DBMS PROJECT REPORT

TREATS ON TRACKS

Name	SRN	Section
Vismaya M	PES1UG22CS706	L
Venkata Sravani Revuri	PES1UG22CS685	L

Abstract:

The "Treats on Tracks" project is designed to create a comprehensive database management system (DBMS) that facilitates food ordering for passengers traveling by train. The system provides an intuitive platform for users to register, log in, and browse through a variety of restaurants and their menus. Passengers can place orders directly from their seats and make payments seamlessly.

Introduction

The application is built with a dual-user perspective, catering to both regular passengers and administrative users. Passengers can manage their profiles, view their order history, and receive updates on their order status. Administrators have the capability to manage restaurant listings, update menu items, and oversee order transactions. This project not only enhances the travel experience by providing easy access to food options but also streamlines the ordering process, ensuring that passengers can enjoy their meals without the hassle of leaving their seats.

- **User Requirements:**

1. Passenger Requirements:
    - Registration and Login: Users should be able to create an account by providing personal details and secure login credentials. Password recovery options should also be available.
    - Profile Management: Users should be able to view and update their personal information such as name, phone number, and email address.
    - Menu Browsing: Users should be able to view a list of available restaurants and their respective menu items, including prices and descriptions.
    - Order Placement: Users should be able to select items from the menu, specify quantities, and place an order. A confirmation of the order should be provided.
    - Payment Processing: Users should have multiple payment options (e.g., credit/debit card, e-wallet) and receive a receipt upon successful payment.

- Order Tracking: Users should receive notifications regarding the status of their orders, including confirmation and estimated delivery time.

2. Admin Requirements:
    - User Management: Admins should be able to view, edit, and delete passenger accounts as necessary.
    - Restaurant Management: Admins should have the ability to add new restaurants, update existing restaurant information, and delete restaurants that are no longer in operation.
    - Menu Management: Admins should be able to add new menu items, update item details (name, price, availability), and remove items from the menu.
    - Order Management: Admins should be able to view all orders placed, track their statuses, and manage any issues related to order fulfillment.
    - Reporting: Admins should have access to reports on sales, user activity, and inventory levels to make informed decisions.

Functional Requirements

1. User Registration and Login: Allow users to create accounts and log in.
2. Profile Management: Enable users to view and update personal information.
3. Menu Browsing: Provide a list of restaurants and their menu items.
4. Order Placement: Allow users to select items and place orders.
5. Payment Processing: Support multiple payment options with receipts.
6. Order Tracking: Notify users about order status and updates.
7. Admin Management: Enable admins to manage users, restaurants, and menu items.

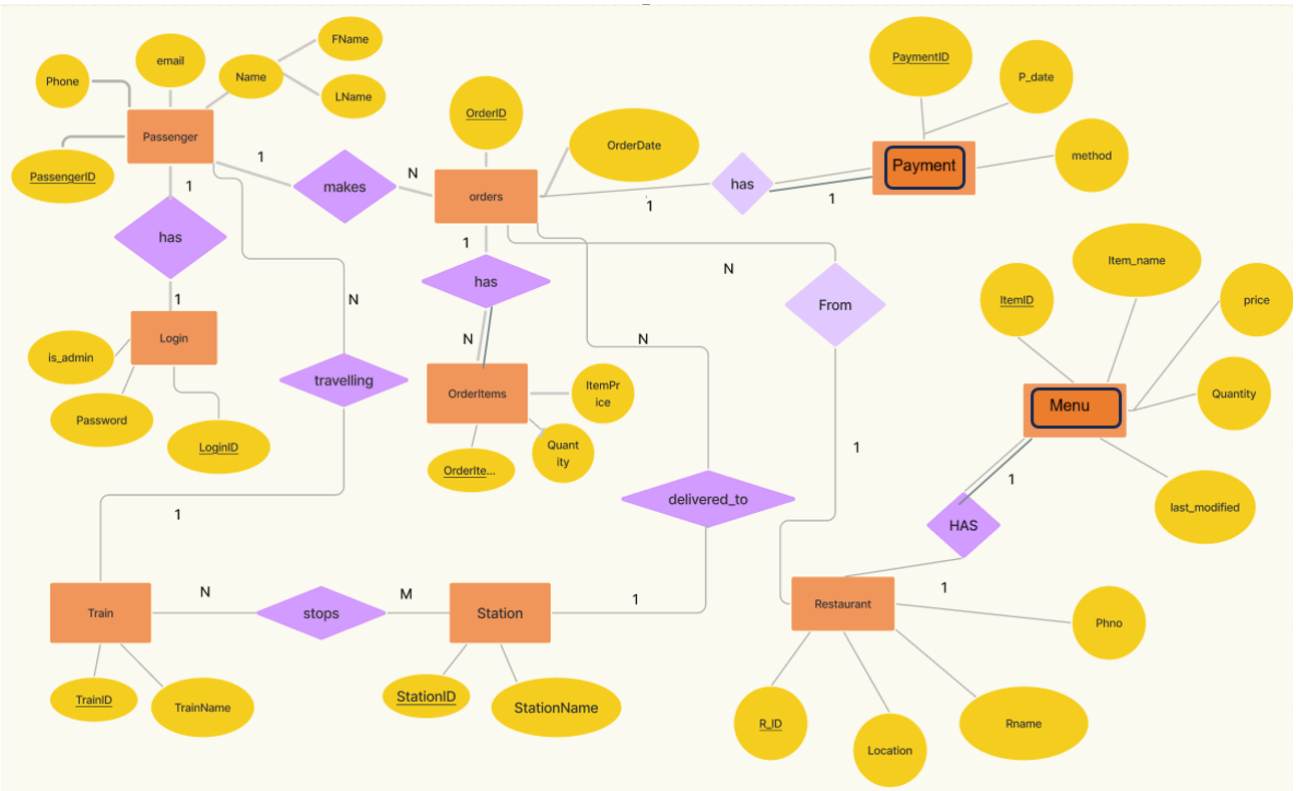
Non-Functional Requirements

1. Performance: The application should load quickly and handle multiple users simultaneously.
2. Security: User data and transactions must be securely handled and protected.
3. Usability: The interface should be user-friendly and easy to navigate.
4. Scalability: The system should accommodate future growth in user base and data.
5. Reliability: The application should be available and functional with minimal downtime.
6. Maintainability: The codebase should be organized for easy updates and modifications.

List of Programming Languages Used

- HTML:
  Used for structuring the web application and creating user interfaces, including forms for registration and order placement.
- CSS:
  Employed for styling the web application to enhance visual appeal and user experience through layout design and formatting.
- JavaScript:
  Utilized for adding interactivity to the application, including form validation and dynamic content updates to improve responsiveness.
- MySQL:
  A relational database management system used to store and manage data, including user accounts, restaurant details, and orders efficiently.
- Flask:
  A lightweight Python web framework that handles backend logic, processes user requests, and connects the frontend to the MySQL database for data operations.

E-R DIAGRAM



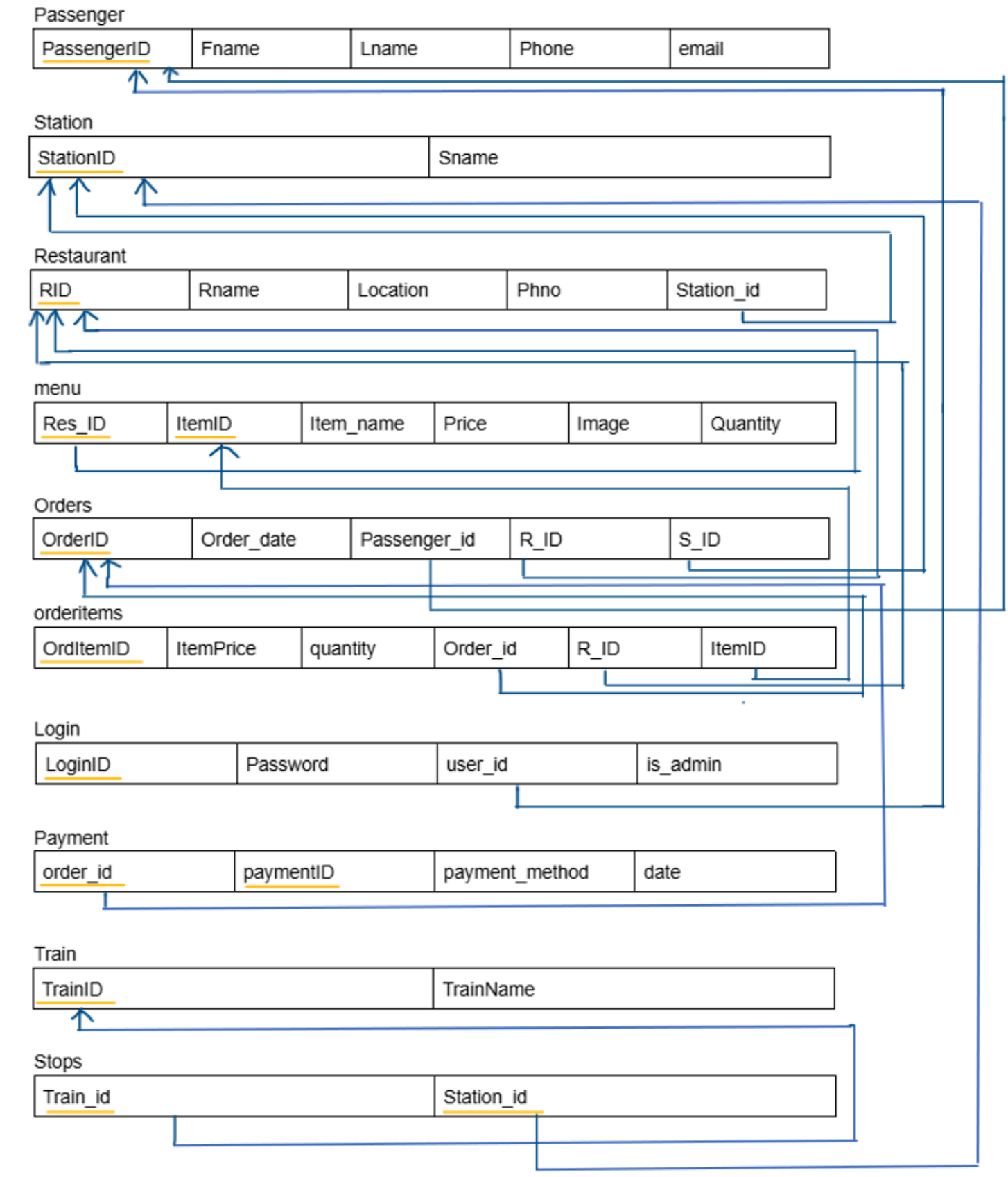
Entities:

- **Passenger:** Represents a person traveling on the train.
  - Attributes: PassengerID, Name, Email, Phone
- **Payment:** Represents a payment made by a passenger.
  - Attributes: PaymentID, Amount, Date, Method
- **Menu:** Represents the food and beverage menu available on the train.
  - Attributes: MenuID, ItemName, Price, Quantity
- **Station:** Represents a train station.
  - Attributes: StationID, StationName, Location
- **Train:** Represents a train.
  - Attributes: TrainID, TrainName, DepartureTime, ArrivalTime

Relationships:

- **Passenger - Payment:** One-to-many. A passenger can make multiple payments.
- **Payment - Menu:** One-to-one. A payment is for one menu item.
- **Menu - Train:** One-to-one. A menu is associated with one train.
- **Station - Train:** Many-to-many. A train can stop at multiple stations, and a station can be used by multiple trains.

RELATIONAL SCHEMA

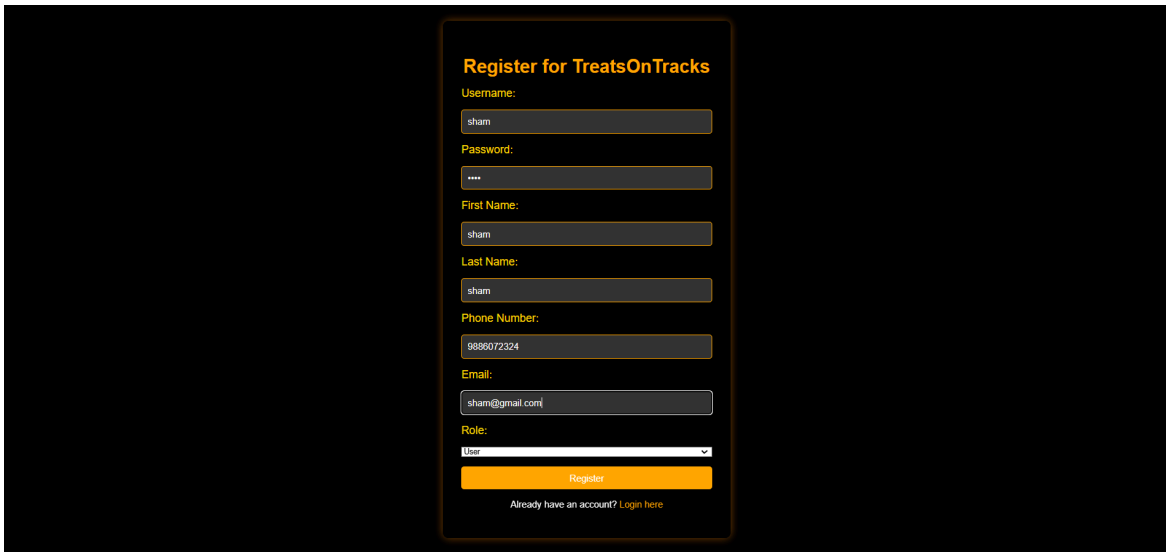


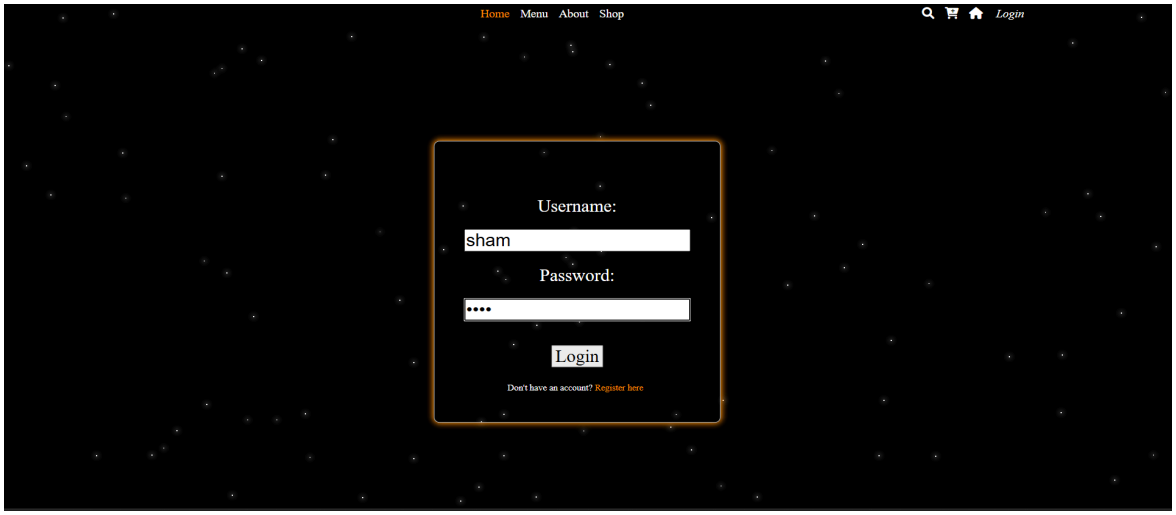
Create Operations

1. **Insert Passenger Details and Login Credentials During Registration**
   `INSERT INTO passenger(PassengerID, Fname, Lname, Phone, email) VALUES (%s, %s, %s, %s, %s)`
   - Adds a new passenger's personal details (first name, last name, phone, and email) to the `passenger` table.

`INSERT INTO Login (LoginID, Password, user_id, is_admin) VALUES (%s, %s, %s, %s)`

   - Adds login details for a passenger, including their role (admin or user), in the `Login` table.





2. **Create MySQL User and Grant Privileges**

```
CREATE USER IF NOT EXISTS %s@'localhost' IDENTIFIED BY %s;
```

- Creates a new MySQL user if they don't already exist.

```
GRANT ALL PRIVILEGES ON *.* TO %s@'localhost' WITH GRANT OPTION;
```

- Grants full privileges to admin users.

```
GRANT SELECT, INSERT ON *.* TO %s@'localhost';

GRANT UPDATE ON passenger TO %s@'localhost';
```

- Grants limited privileges (select, insert) to regular users, with update privileges only on the `passenger` table.

3. **Place an Order**

   ```
   INSERT INTO orders (OrderID, Order_date, R_ID, S_ID,
   Passenger_id) VALUES (%s, %s, %s, %s, %s)
   ```

   - Inserts a new order into the `orders` table with details like order ID, date, restaurant ID, station ID, and passenger ID.

   ```
   INSERT INTO orderitems (OrdItemID, ItemPrice, quantity, Order_id,
   R_ID, ItemID) VALUES (%s, %s, %s, %s, %s, %s)
   ```

   - Adds each ordered item's price, quantity, order ID, restaurant ID, and item ID into `orderitems`.

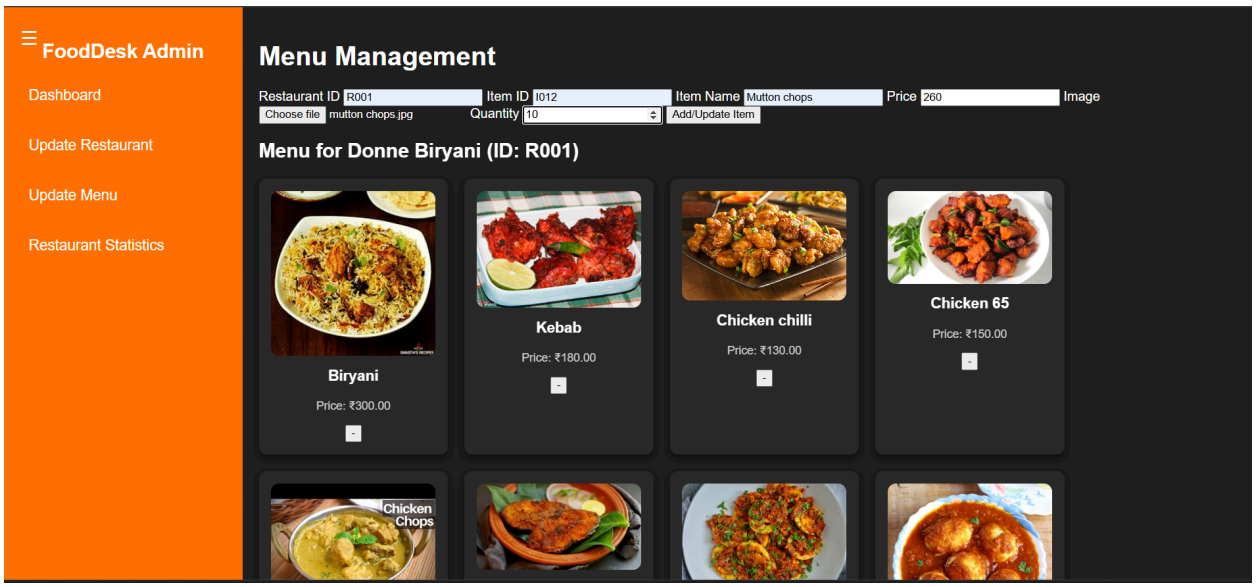
```
mysql> select * from orders;
+----------+------------+--------------+------+------+
| OrderID  | Order_date | Passenger_id | R_ID | S_ID |
+----------+------------+--------------+------+------+
| 6208e637 | 2024-11-12 | M7S8B19Y48   | R001 | S001 |
| a1dbea0c | 2024-11-12 | M7S8B19Y48   | R001 | S001 |
| ae5e6c3a | 2024-11-12 | M7S8B19Y48   | R001 | S001 |
+----------+------------+--------------+------+------+
3 rows in set (0.00 sec)

mysql> select * from orderitems;
+-----------+-----------+----------+----------+------+--------+
| OrdItemID | ItemPrice | quantity | Order_id | R_ID | ItemID |
+-----------+-----------+----------+----------+------+--------+
| cde06704  |    300.00 |        2 | a1dbea0c | R001 | I001   |
| e528edb3  |    260.00 |       10 | 6208e637 | R001 | I012   |
| ec8145e1  |    180.00 |        3 | ae5e6c3a | R001 | I005   |
+-----------+-----------+----------+----------+------+--------+
3 rows in set (0.00 sec)
```

(BASED ON USER SELECTION)

4. **Record Payment Details**
   `INSERT INTO payment (order_id, paymentID, payment_method, date) VALUES (%s, %s, %s, %s)`
   - Inserts a new payment record with details like order ID, payment ID, method, and date.

```
mysql> select * from payment;
+----------+-----------+----------------+------------+
| order_id | paymentID | payment_method | date       |
+----------+-----------+----------------+------------+
| 6208e637 | 4757b95f  | Cash           | 2024-11-12 |
| a1dbea0c | b8709214  | Cash           | 2024-11-12 |
| ae5e6c3a | e736f7af  | DebitCard      | 2024-11-12 |
+----------+-----------+----------------+------------+
3 rows in set (0.00 sec)
```

5. **Add New Menu Item**
   `INSERT INTO menu (Res_ID, ItemID, Item_name, Price, Image, Quantity) VALUES (%s, %s, %s, %s, %s, %s)`
   - Inserts a new menu item with details like restaurant ID, item ID, name, price, image, and quantity.



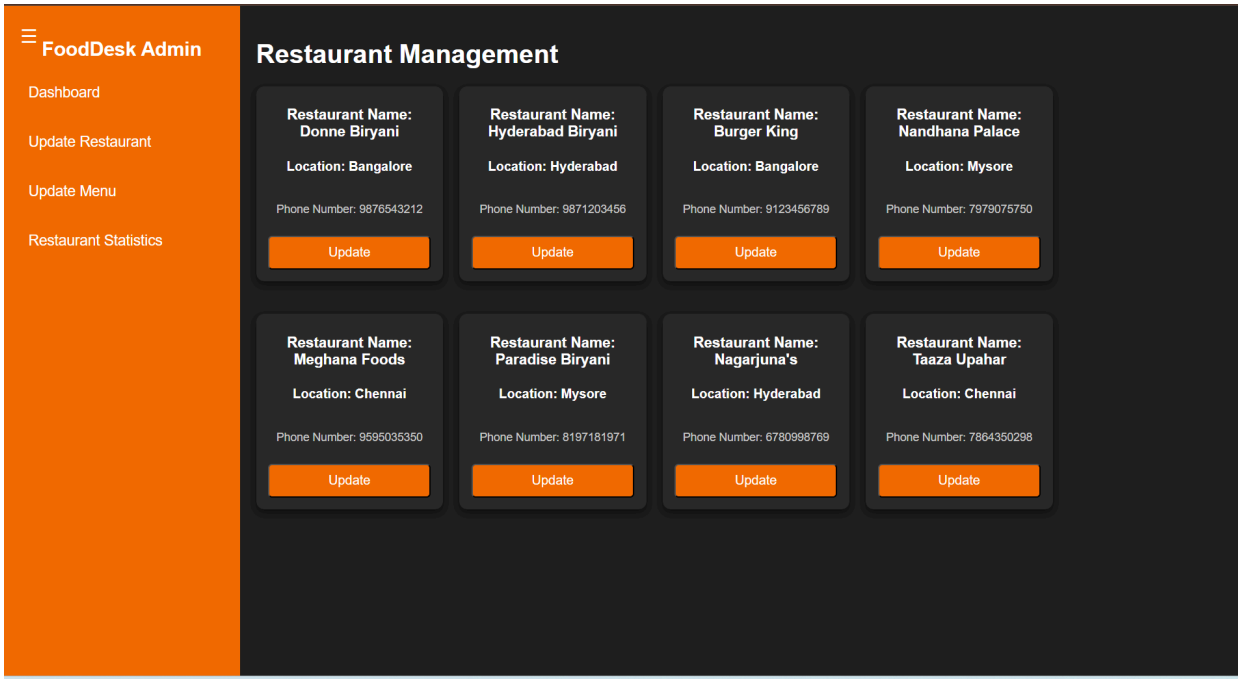
Read Operations

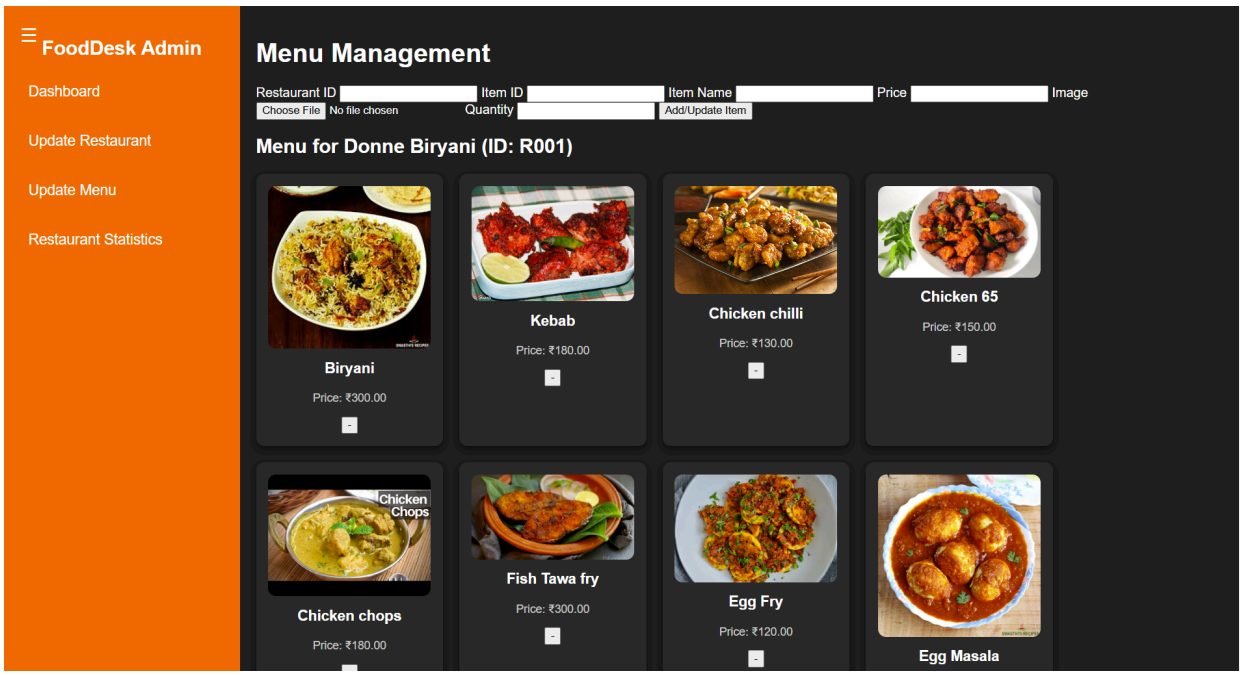
1. **Login Query**
   `SELECT user_id, is_admin FROM Login WHERE LoginID = %s AND Password = %s`
   - Verifies login credentials and retrieves `user_id` and role (`is_admin`) for a user.

```
mysql> select * from login;
+---------+----------+------------+----------+
| LoginID | Password | user_id    | is_admin |
+---------+----------+------------+----------+
| a       | a        | DNLOIF26ID |        1 |
| s       | s        | M7S8B19Y48 |        0 |
| sravani | ss       | AB8XVZXZ2P |        1 |
+---------+----------+------------+----------+
3 rows in set (0.00 sec)
```

```
mysql> SELECT user_id, is_admin
    -> FROM Login
    -> WHERE LoginID = 'sravani' AND Password = 'ss';
+------------+----------+
| user_id    | is_admin |
+------------+----------+
| AB8XVZXZ2P |        1 |
+------------+----------+
1 row in set (0.00 sec)
```

2. **Fetch Restaurants and Menu Items for Admin Dashboard**
   `SELECT Rname, Location, Phno, RID FROM restaurant`
   - Retrieves all restaurants for display in the admin dashboard.






```
SELECT Item_name, Price, Image FROM menu WHERE Res_ID = %s
```

- Retrieves all menu items for each restaurant.

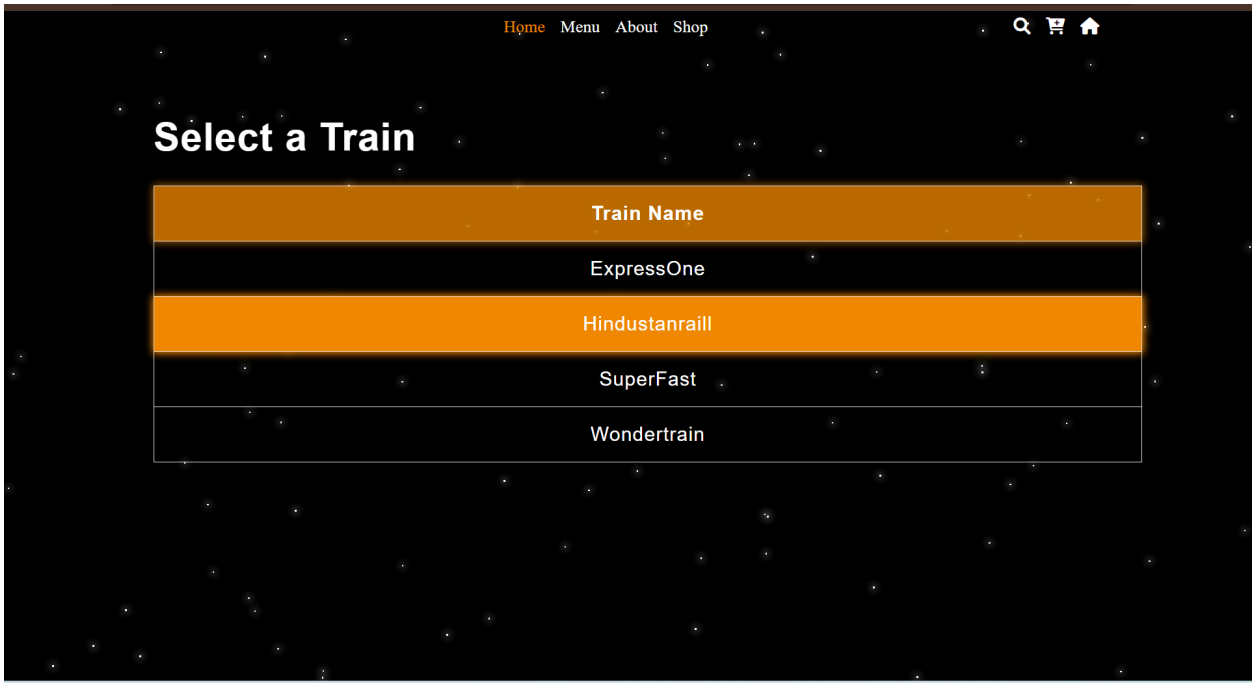
```
mysql> SELECT Item_name, Price FROM menu WHERE Res_ID = 'R001'
    -> ;
+----------------+--------+
| Item_name      | Price  |
+----------------+--------+
| Biryani        | 300.00 |
| Kebab          | 180.00 |
| Chicken chilli | 130.00 |
| Chicken 65     | 150.00 |
| Chicken chops  | 180.00 |
| Fish Tawa fry  | 300.00 |
| Egg Fry        | 120.00 |
| Egg Masala     | 100.00 |
| Mutton chops   | 260.00 |
+----------------+--------+
9 rows in set (0.00 sec)
```

3. **Fetch User Profile Information**
   ```
   SELECT PassengerID, Fname, Lname, Phone, email FROM passenger
   WHERE PassengerID = %s
   ```
   - Retrieves profile details of a passenger based on their ID.

```
mysql> select * from passenger;
+-------------+---------+--------+------------+--------------+
| PassengerID | Fname   | Lname  | Phone      | email        |
+-------------+---------+--------+------------+--------------+
| AB8XVZXZ2P  | sravani | revuri | 9886531909 | rv@gmail.com |
| DNLOIF26ID  | a       | a      | 9886531906 | a@gmail.com  |
| M7S8B19Y48  | s       | s      | 9886531900 | s@gmail.com  |
+-------------+---------+--------+------------+--------------+
3 rows in set (0.00 sec)
```

4. **Fetch Train Stops Based on Train ID**
   `SELECT s.Station_id, st.Sname FROM stops s JOIN Station st ON s.Station_id = st.StationID WHERE s.Train_id = %s`
   - Retrieves all stops of a train with station IDs and names.

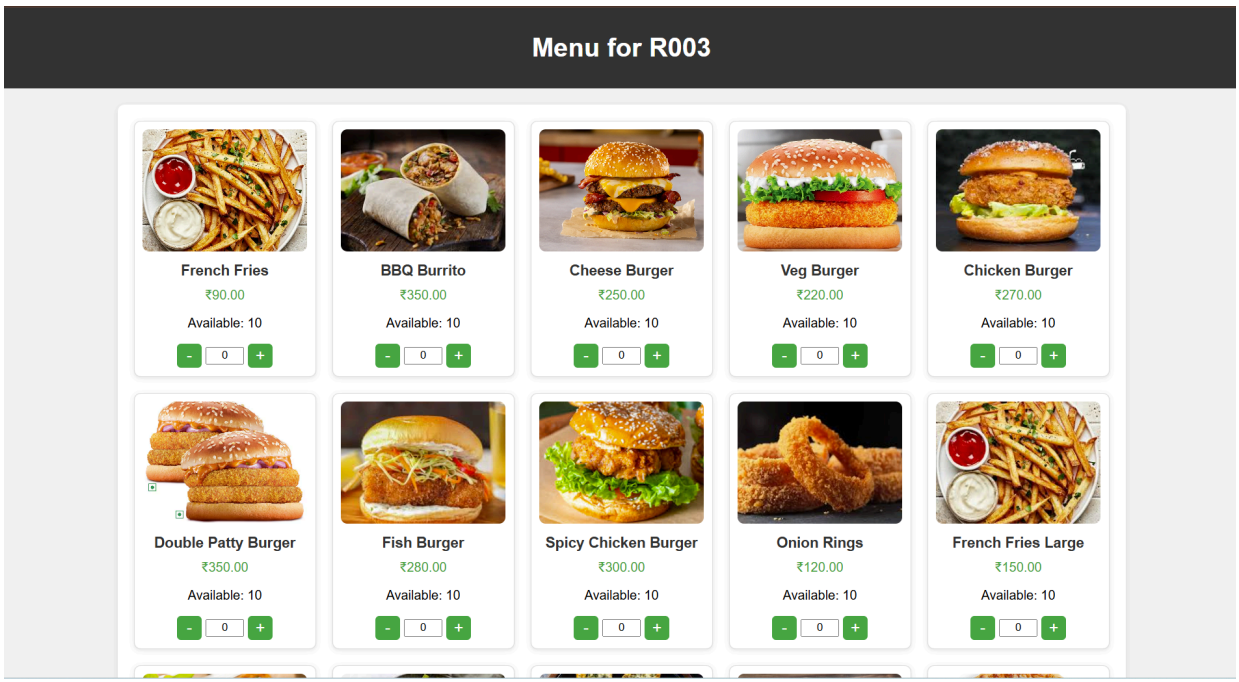
5. **Fetch All Trains**
   `SELECT TrainID, TrainName FROM train`
   - Retrieves all train IDs and names.



6. **Fetch Restaurants at a Specific Station**
   `SELECT Rname, RID, Phno FROM restaurant WHERE Station_id = %s`
   - Fetches all restaurants available at a specified station.

7. **Fetch Menu Items at a Restaurant**

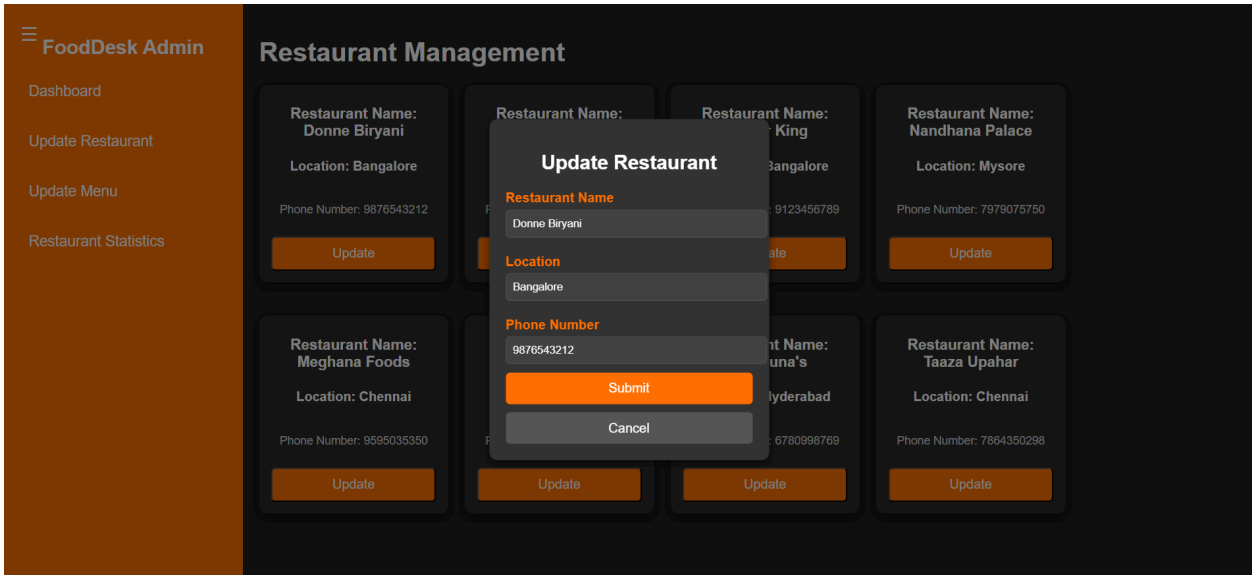
```
SELECT Item_name, Price, Image, quantity FROM menu WHERE Res_ID = %s
```



- Retrieves menu items available at a selected restaurant.

Update Operations

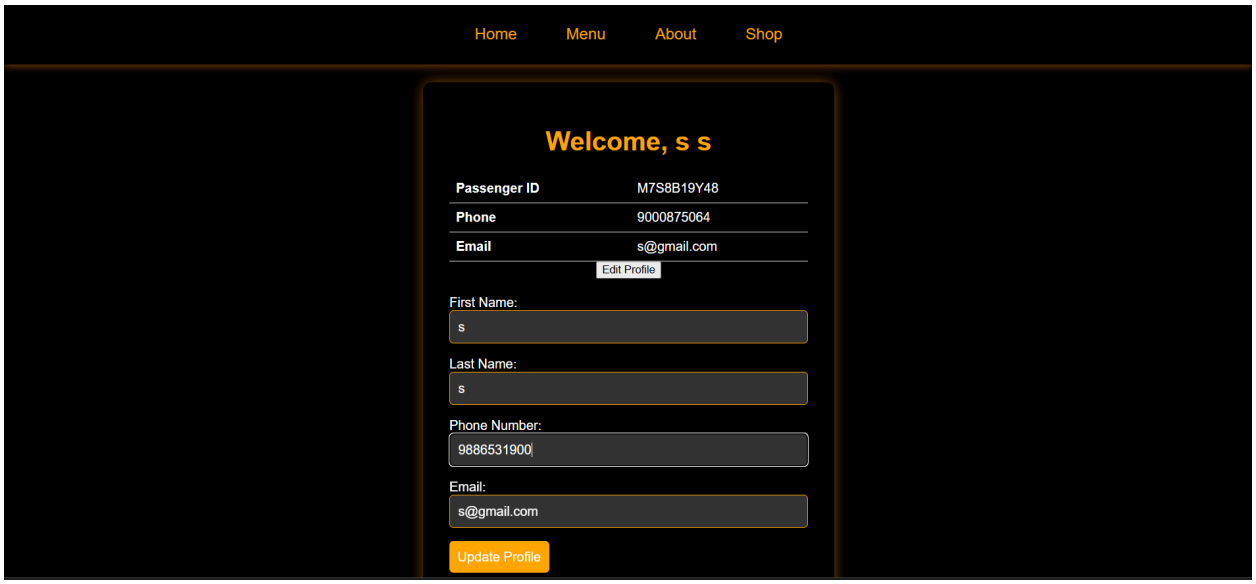
1. **Update Restaurant Details**
   `UPDATE restaurant SET Rname = %s, Location = %s, Phno = %s WHERE RID = %s`
   - Updates the details of a restaurant (name, location, phone) based on its ID.

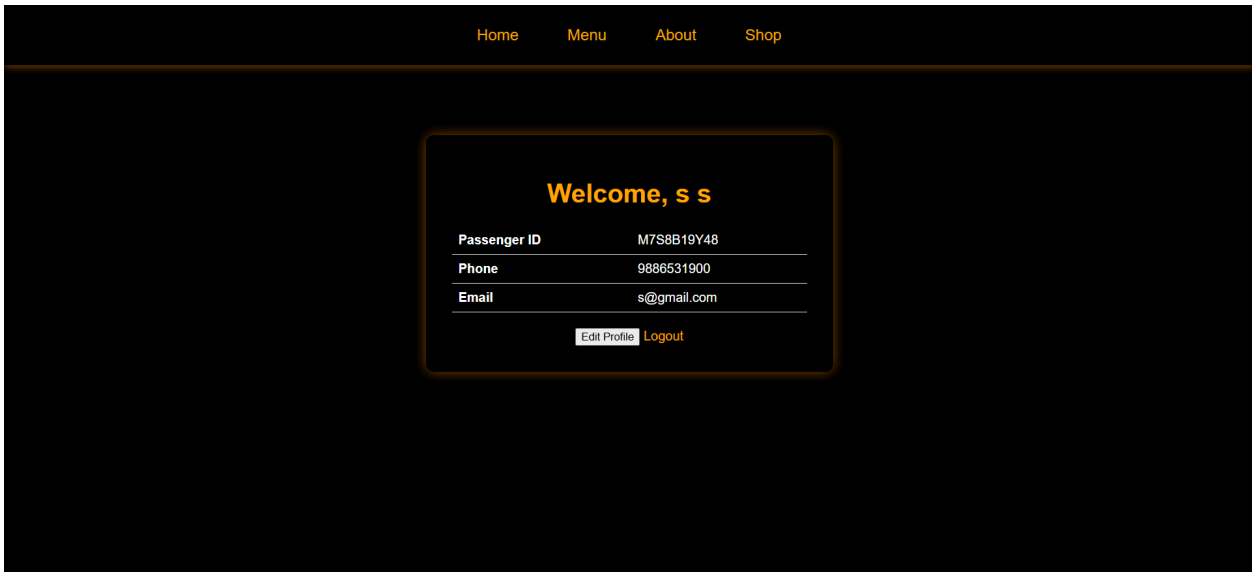


2. **Update User Profile**

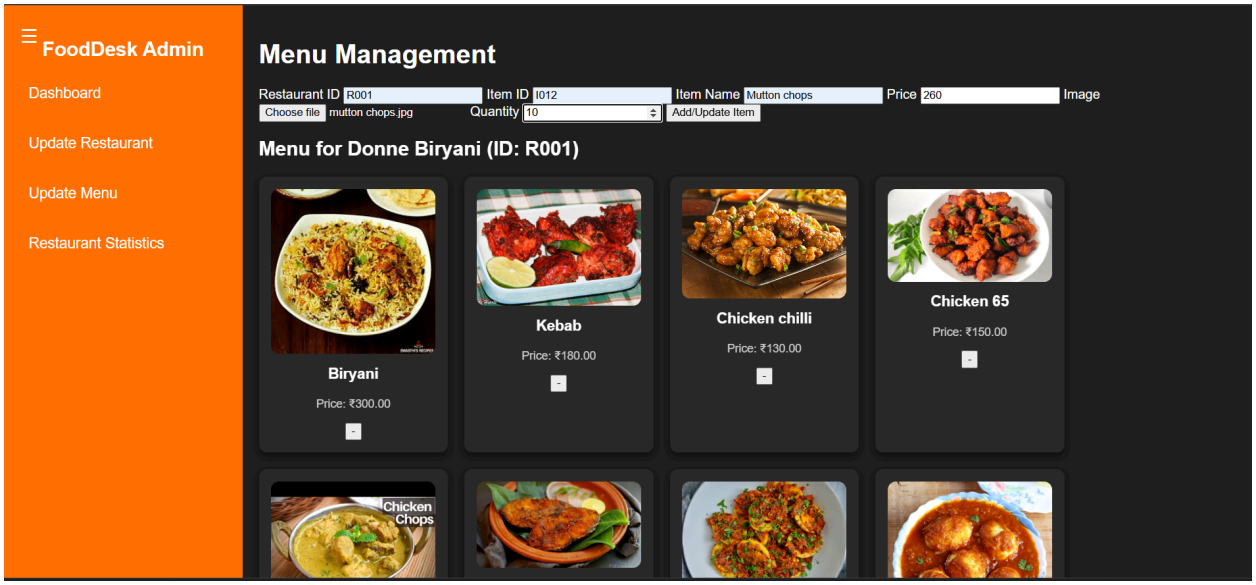
   ```
   UPDATE passenger SET Fname = %s, Lname = %s, Phone = %s, email = %s WHERE PassengerID = %s
   ```

   - Updates a passenger's profile details (first name, last name, phone, and email).





3. **Update Menu Item**
   `SELECT ItemID FROM menu WHERE Res_ID = %s AND ItemID = %s`
   - Checks if an item exists in a restaurant's menu by item ID.



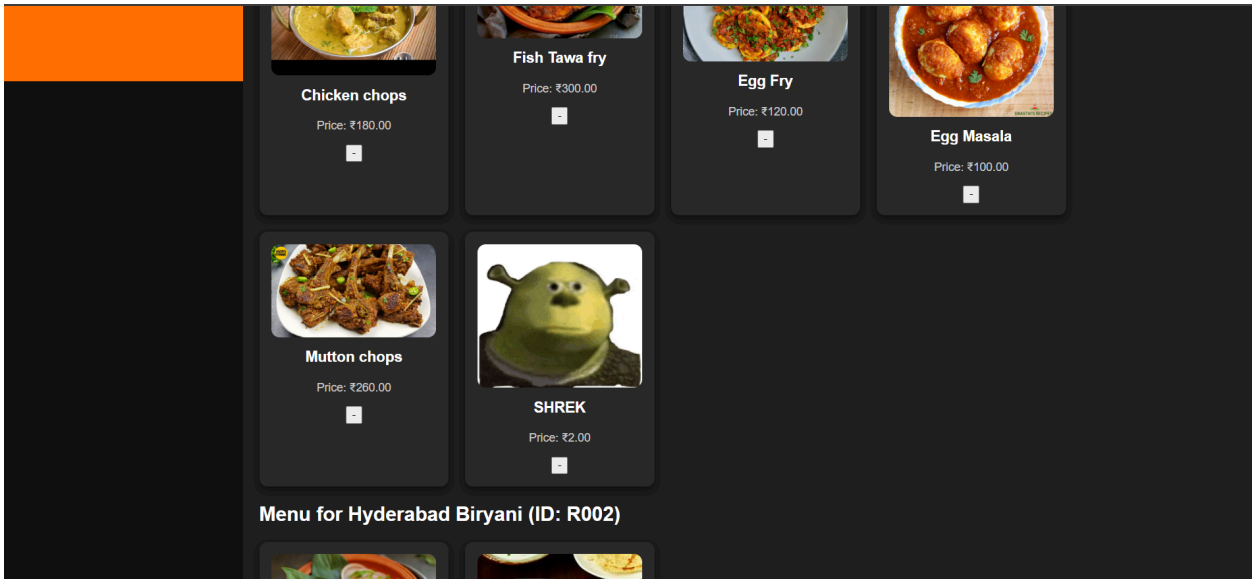

```
UPDATE menu SET Item_name = %s, Price = %s, Image = %s, Quantity = %s
WHERE Res_ID = %s AND ItemID = %s
```

   - Updates details of an existing menu item (name, price, image, quantity).

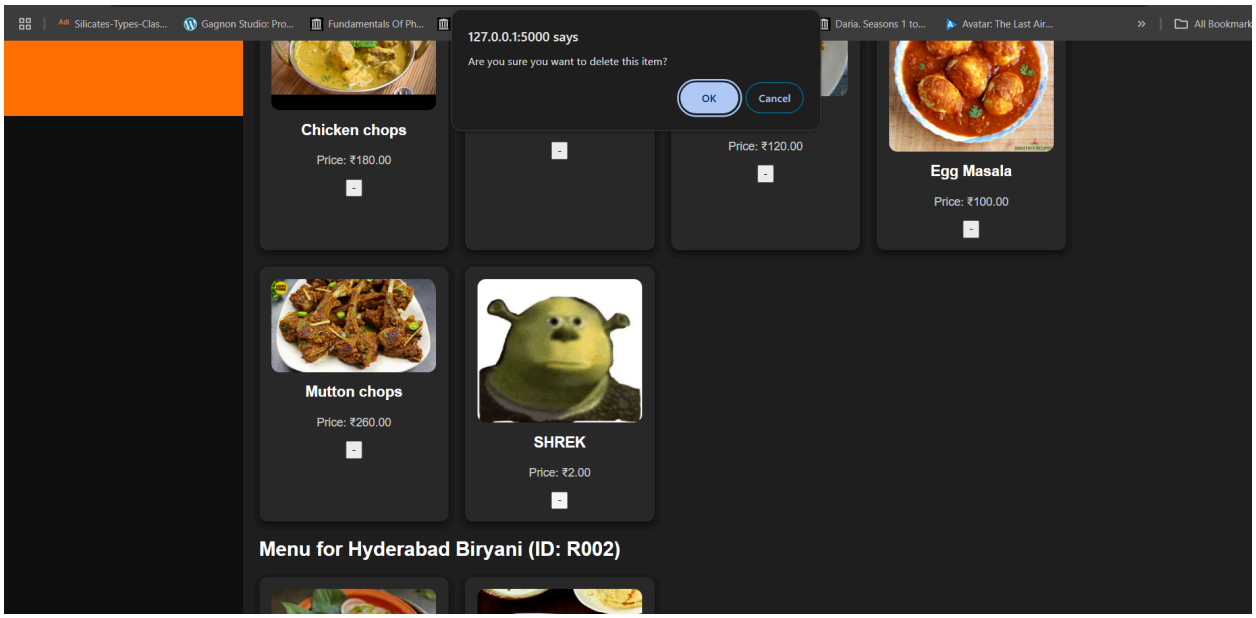
Delete Operations

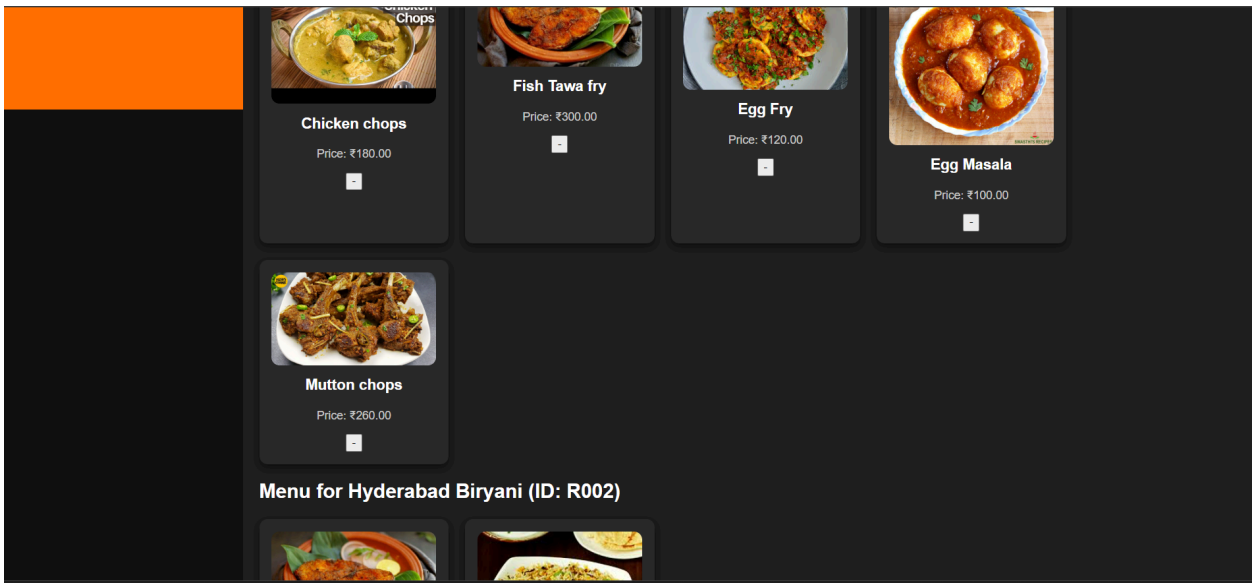
1. **Delete a Menu Item**
   `DELETE FROM menu WHERE Res_ID = %s AND Item_name = %s AND Price = %s`
   - Deletes a specific menu item from a restaurant's menu based on restaurant ID, item name, and price.

     Trying to delete shrek from menu by pressing the '-' button



After delete





FUNCTIONS:

The `CalculateTotalPrice` function calculates the total price for an order by summing the cost of each item's quantity and price.

```
DELIMITER $$

DROP FUNCTION IF EXISTS CalculateTotalPrice $$

CREATE FUNCTION CalculateTotalPrice(order_id VARCHAR(20))
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    DECLARE total_amount DECIMAL(10,2) DEFAULT 0;

    SELECT SUM(oi.quantity * m.Price)
    INTO total_amount
    FROM orderitems oi
    JOIN menu m ON oi.R_ID = m.Res_ID AND oi.ItemID = m.ItemID
    WHERE oi.Order_id = order_id;

    RETURN total_amount;
END $$

DELIMITER ;
```

Declare Variable: Initializes `total_amount` to store the total cost.

Calculate Total:

- Joins `orderitems` (`oi`) and `menu` (`m`) on `R_ID` and `ItemID` to match restaurant and item IDs.
- Multiplies each item's `quantity` by its `price` and sums the results for the specified `order_id`.

Return Total: Returns `total_amount` as the total order cost.

```
mysql> select * from orders;
+----------+------------+--------------+------+------+
| OrderID  | Order_date | Passenger_id | R_ID | S_ID |
+----------+------------+--------------+------+------+
| a1dbea0c | 2024-11-12 | M7S8B19Y48   | R001 | S001 |
| ae5e6c3a | 2024-11-12 | M7S8B19Y48   | R001 | S001 |
+----------+------------+--------------+------+------+
2 rows in set (0.00 sec)

mysql> SELECT CalculateTotalPrice('a1dbea0c');
+---------------------------------+
| CalculateTotalPrice('a1dbea0c') |
+---------------------------------+
|                          600.00 |
+---------------------------------+
1 row in set (0.00 sec)
```

TRIGGER:

```
DROP TRIGGER IF EXISTS update_menu_quantity;

DELIMITER //

CREATE TRIGGER update_menu_quantity
AFTER INSERT ON orderitems
FOR EACH ROW
BEGIN
    DECLARE current_quantity INT;

    -- Fetch the current stock of the ordered item
    SELECT quantity INTO current_quantity
    FROM menu
    WHERE ItemID = NEW.ItemID AND Res_ID = NEW.R_ID;
    -- Validate stock availability
    IF current_quantity >= NEW.quantity THEN
        -- Deduct ordered quantity from stock
```

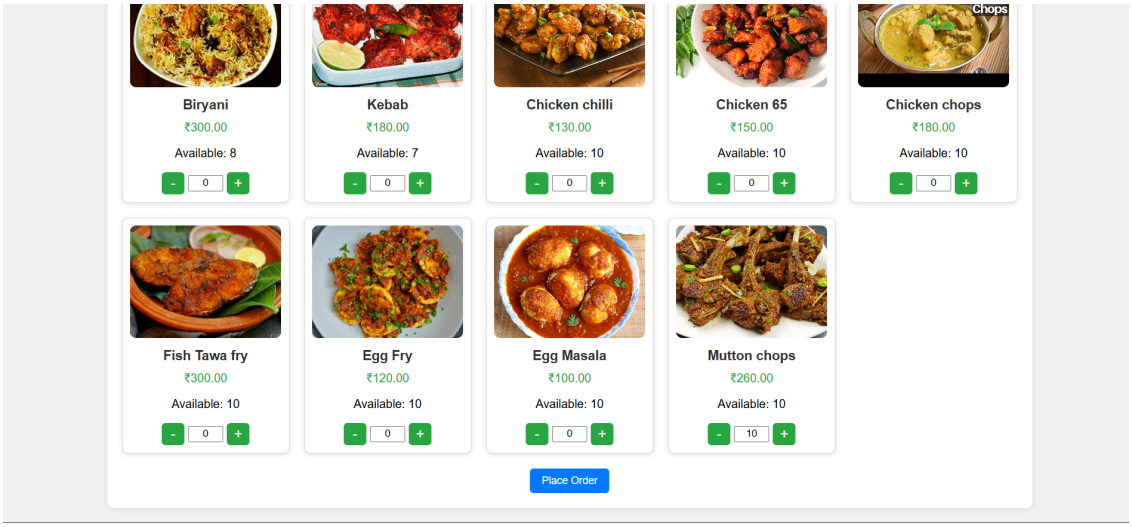
```
        UPDATE menu
        SET quantity = quantity - NEW.quantity
        WHERE ItemID = NEW.ItemID AND Res_ID = NEW.R_ID;
    ELSE
        -- Raise error if stock is insufficient
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Insufficient stock for the requested
quantity.';
    END IF;
END //

DELIMITER ;
```

This trigger, `update_menu_quantity`, is triggered after a new row is inserted into the `orderitems` table:

- **Purpose**: It checks if there's enough stock in the `menu` table for the ordered item. If there is, it reduces the item's quantity in the `menu` table accordingly. If not, it raises an error to indicate insufficient stock.
- **Key Steps**:
    - The `current_quantity` of the item is fetched from the `menu` table.
    - The trigger checks if this stock is enough for the new order's quantity.
    - If sufficient, the `menu` quantity is decreased; if insufficient, an error message is raised with `SIGNAL`.

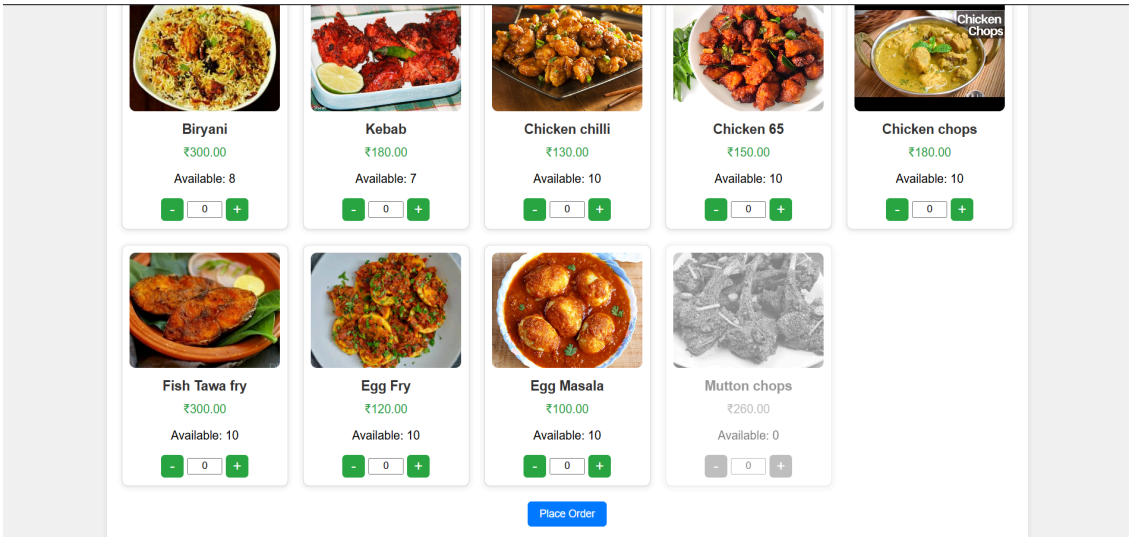
FRONTEND:

Food Item Mutton Chops is being ordered (10 quantity which is equal to available amount also reflected in backend)



```
mysql> select Res_ID,ItemID,Item_name,Price,quantity from menu
+--------+--------+----------------------+--------+----------+
| Res_ID | ItemID | Item_name            | Price  | quantity |
+--------+--------+----------------------+--------+----------+
| R001   | I001   | Biryani              | 300.00 |        8 |
| R001   | I005   | Kebab                | 180.00 |        7 |
| R001   | I006   | Chicken chilli       | 130.00 |       10 |
| R001   | I007   | Chicken 65           | 150.00 |       10 |
| R001   | I008   | Chicken chops        | 180.00 |       10 |
| R001   | I009   | Fish Tawa fry        | 300.00 |       10 |
| R001   | I010   | Egg Fry              | 120.00 |       10 |
| R001   | I011   | Egg Masala           | 100.00 |       10 |
| R001   | I012   | Mutton chops         | 260.00 |       10 |
```

AFTER ORDER IS PLACED



BACKEND:

```
mysql> select Res_ID,ItemID,Item_name,Price,quantity from menu;
+--------+--------+----------------------+--------+----------+
| Res_ID | ItemID | Item_name            | Price  | quantity |
+--------+--------+----------------------+--------+----------+
| R001   | I001   | Biryani              | 300.00 |        8 |
| R001   | I005   | Kebab                | 180.00 |        7 |
| R001   | I006   | Chicken chilli       | 130.00 |       10 |
| R001   | I007   | Chicken 65           | 150.00 |       10 |
| R001   | I008   | Chicken chops        | 180.00 |       10 |
| R001   | I009   | Fish Tawa fry        | 300.00 |       10 |
| R001   | I010   | Egg Fry              | 120.00 |       10 |
| R001   | I011   | Egg Masala           | 100.00 |       10 |
| R001   | I012   | Mutton chops         | 260.00 |        0 |
```

2)

```
DELIMITER $$

CREATE TRIGGER update_last_modified
BEFORE UPDATE ON menu
FOR EACH ROW
BEGIN
   SET NEW.last_modified = CURRENT_TIMESTAMP;
END$$

DELIMITER ;
```

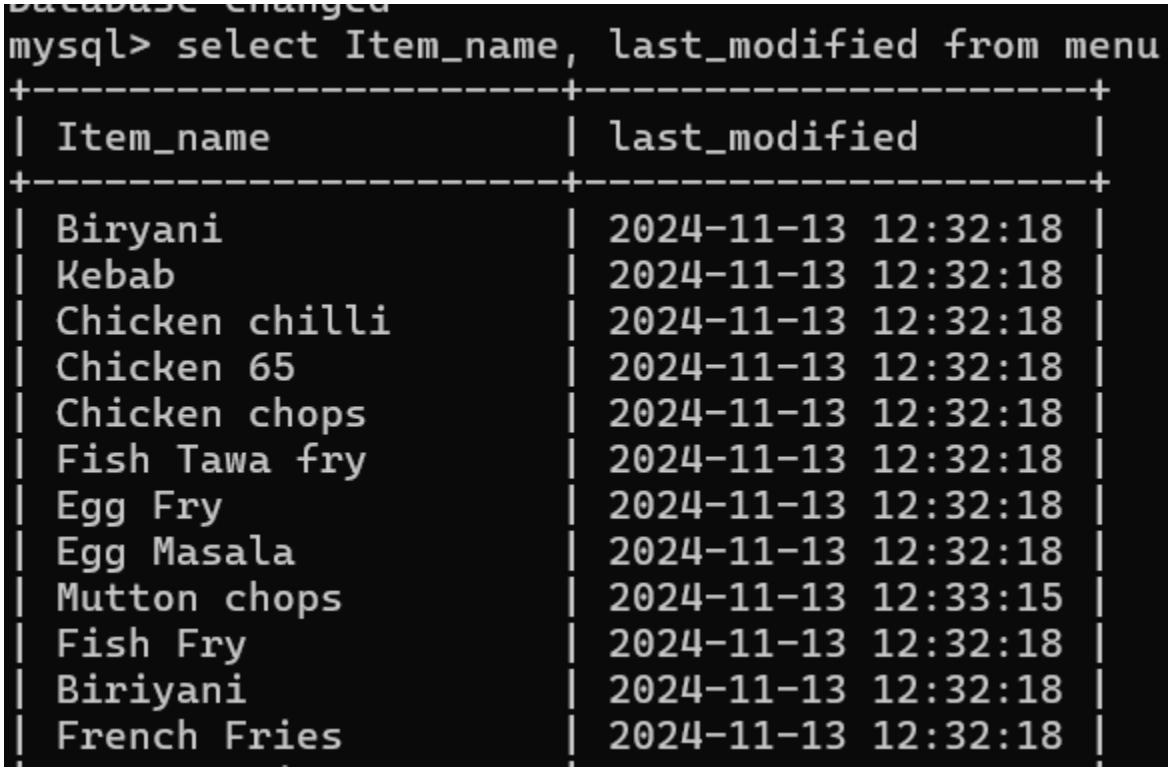
This trigger automatically updates the last_modified column to the current timestamp whenever a row in the menu table is updated.

It is a BEFORE UPDATE trigger, meaning it fires before any update occurs.

The trigger sets the last_modified field to CURRENT_TIMESTAMP for the updated row.

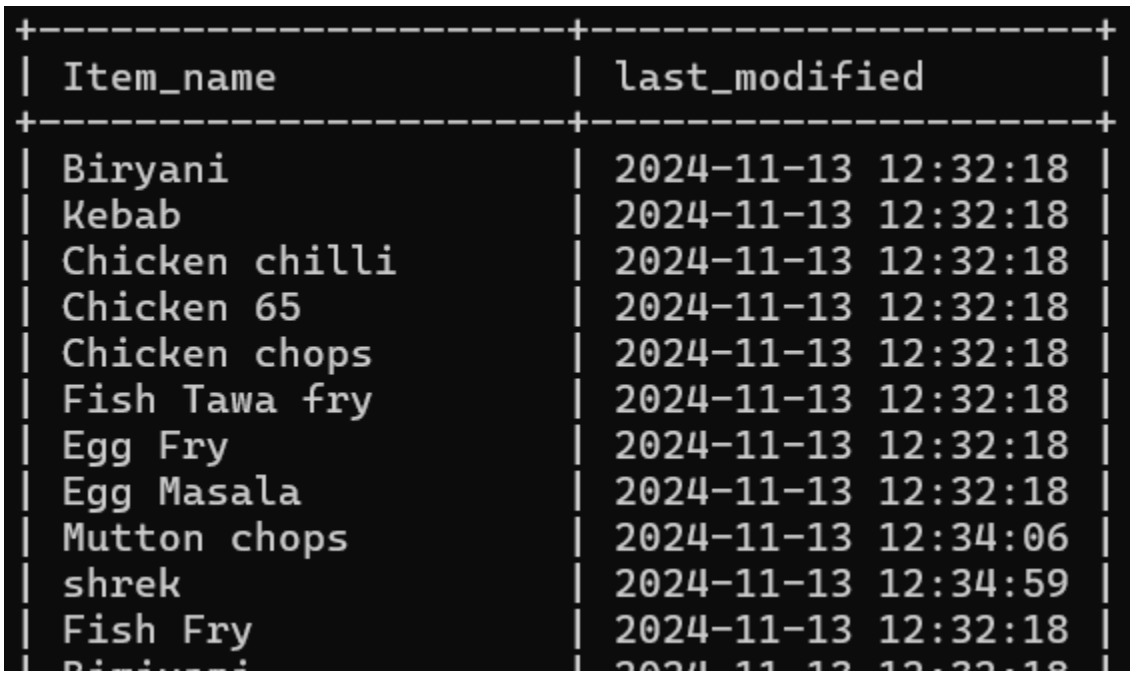
This ensures that the last_modified column is always up-to-date with the current time whenever a row in the menu table is modified.

BEFORE UPDATE:



```
mysql> select Item_name, last_modified from menu
+----------------------+---------------------+
| Item_name            | last_modified       |
+----------------------+---------------------+
| Biryani              | 2024-11-13 12:32:18 |
| Kebab                | 2024-11-13 12:32:18 |
| Chicken chilli       | 2024-11-13 12:32:18 |
| Chicken 65           | 2024-11-13 12:32:18 |
| Chicken chops        | 2024-11-13 12:32:18 |
| Fish Tawa fry        | 2024-11-13 12:32:18 |
| Egg Fry              | 2024-11-13 12:32:18 |
| Egg Masala           | 2024-11-13 12:32:18 |
| Mutton chops         | 2024-11-13 12:33:15 |
| Fish Fry             | 2024-11-13 12:32:18 |
| Biriyani             | 2024-11-13 12:32:18 |
| French Fries         | 2024-11-13 12:32:18 |
```

AFTER UPDATE:



Item_name	last_modified
Biryani	2024-11-13 12:32:18
Kebab	2024-11-13 12:32:18
Chicken chilli	2024-11-13 12:32:18
Chicken 65	2024-11-13 12:32:18
Chicken chops	2024-11-13 12:32:18
Fish Tawa fry	2024-11-13 12:32:18
Egg Fry	2024-11-13 12:32:18
Egg Masala	2024-11-13 12:32:18
Mutton chops	2024-11-13 12:34:06
shrek	2024-11-13 12:34:59
Fish Fry	2024-11-13 12:32:18

NESTED QUERY:

```
SELECT m.Item_name, m.Price, r.Rname

FROM menu m

JOIN restaurant r ON m.Res_ID = r.RID

WHERE m.Price > (

   SELECT AVG(Price)

   FROM menu

   WHERE Res_ID = %s

)

AND m.Res_ID = %s
```

```
mysql> SELECT m.Item_name, m.Price, r.Rname
    -> FROM menu m
    -> JOIN restaurant r ON m.Res_ID = r.RID
    -> WHERE m.Price > (
    ->     SELECT AVG(Price)
    ->     FROM menu
    ->     WHERE Res_ID = 'R001'
    -> )
    -> AND m.Res_ID = 'R001';
+---------------+--------+---------------+
| Item_name     | Price  | Rname         |
+---------------+--------+---------------+
| Biryani       | 300.00 | Donne Biryani |
| Fish Tawa fry | 300.00 | Donne Biryani |
| Mutton chops  | 260.00 | Donne Biryani |
+---------------+--------+---------------+
3 rows in set (0.00 sec)
```

This query retrieves the names, prices, and restaurant names of items from a restaurant's menu where the item's price is greater than the restaurant's average menu price.

1. **Subquery**:
    - `SELECT AVG(Price) FROM menu WHERE Res_ID = %s`: Calculates the average price of all menu items for a specific restaurant.
2. **Outer Query**:
    - `SELECT m.Item_name, m.Price, r.Rname`: Selects the item name, price, and restaurant name.
    - `WHERE m.Price > (...) AND m.Res_ID = %s`: Filters items with a price greater than the average price calculated by the subquery for the given restaurant ID.

In short, it fetches items that are more expensive than the average price of that restaurant.

AGGREGATE QUERIES

1. Calculating the Average Price for a Specific Restaurant using 'AVG'

   SELECT AVG(Price) FROM menu WHERE Res_ID = rid

- **Aggregate Function Used:** `AVG()`
- **Purpose:** The `AVG()` function calculates the average price of all menu items for a specific restaurant, identified by `Res_ID`.

```python
def restaurant_items_above_average():
    cursor = db.cursor()

    items = []
    average_price = None  # Initialize the average price
    if request.method == 'POST':
        rid = request.form.get('rid')
        if rid:
            # Query to get the average price for the specified restaurant
            cursor.execute("""
                SELECT AVG(Price) FROM menu WHERE Res_ID = %s
            """, (rid,))
            average_price = cursor.fetchone()[0]

            # Query to get items whose price is greater than the average price of items in the specified restaurant
            cursor.execute("""
                SELECT m.Item_name, m.Price, r.Rname
                FROM menu m
                JOIN restaurant r ON m.Res_ID = r.RID
                WHERE m.Price > %s
                  AND m.Res_ID = %s
            """, (average_price, rid))
            items = cursor.fetchall()

    cursor.close()
    db.close()
```

The query retrieves the average price of items from the `menu` table where the `Res_ID` matches the given `rid`.

It retrieves the details of items whose prices are greater than the average price calculated for a specific restaurant.

2. Calculating the total price of an order using 'SUM'

   SELECT SUM(oi.quantity * m.Price)

   INTO total_amount

   FROM orderitems oi

   JOIN menu m ON oi.R_ID = m.Res_ID AND oi.ItemID = m.ItemID

   WHERE oi.Order_id = order_id;

- **Aggregate Function Used:** `SUM()`
- **Purpose:** The `SUM()` function calculates the total amount for an order by adding up the cost of each item in the order.

The query multiplies the `quantity` of each item (`oi.quantity`) in the `orderitems` table by the `Price` of the item (`m.Price`) from the `menu` table.

The resulting sum, representing the total price of all items in the specified order

TABLES

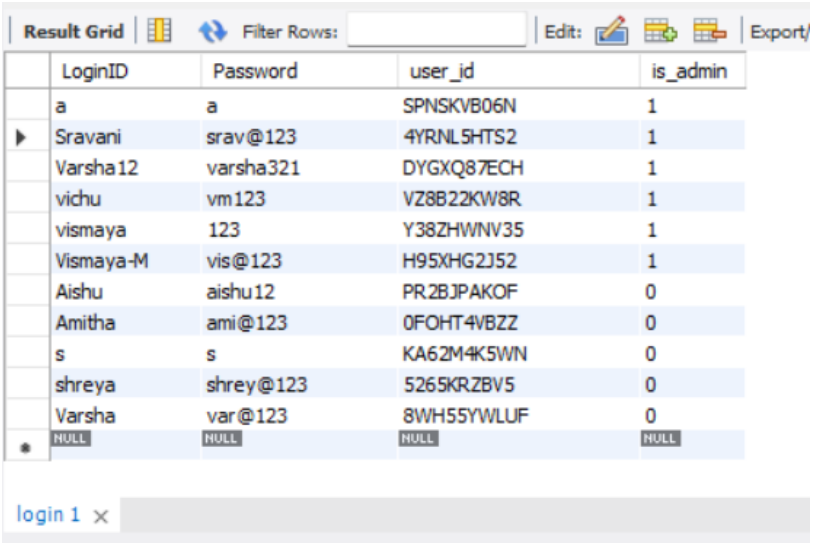
```
mysql> use treatsontracks;
Database changed
mysql> show tables;
+--------------------------+
| Tables_in_treatsontracks |
+--------------------------+
| login                    |
| menu                     |
| orderitems               |
| orders                   |
| passenger                |
| payment                  |
| restaurant               |
| station                  |
| stops                    |
| train                    |
+--------------------------+
10 rows in set (0.00 sec)
```

PASSENGER

PassengerID	Fname	Lname	Phone	email
0FOHT4VBZZ	Amitha	Reddy	8988445734	ami@gmail.com
4YRNL5HTS2	Sravani	Revuri	7865478654	dbms@gmail.com
5265KRZBV5	Shreya	Srinivasan	8794378376	shr@gmail.com
5WTLIXH7MZ	f	f	0	f@gmail.com
8WH55YWLUF	Varsha	Badri	9056765789	varsha@gmail.com
9M0QLU4B12	Vismaya	M	9975057977	vi@gmail.com
B3WMTSNEFE	c	c	7975050977	vip@gmail.com
DYGXQ87ECH	Varsha	Jyothsna	8786746542	jyo@gmail.com
H95XHG2J52	Vismaya	Murali	8884888902	sql@gmail.com
KA62M4K5WN	s	s	9000875064	s@gmail.com
PR2BJPAKOF	Aishwa...	Chandana	8983465237	ais@gmail.com
SPNSKVB06N	a	a	9535023936	a@gmail.com
UYCOTRNF8R	abc	abc	7975057976	vi5@gmail.com

passenger 1 ×

LOGIN



Result Grid | Filter Rows: | Edit: | Export/

LoginID	Password	user_id	is_admin
a	a	SPNSKVB06N	1
Sravani	srav@123	4YRNL5HTS2	1
Varsha12	varsha321	DYGXQ87ECH	1
vichu	vm123	VZ8B22KW8R	1
vismaya	123	Y38ZHWNV35	1
Vismaya-M	vis@123	H95XHG2J52	1
Aishu	aishu12	PR2BJPAKOF	0
Amitha	ami@123	0FOHT4VBZZ	0
s	s	KA62M4K5WN	0
shreya	shrey@123	5265KRZBV5	0
Varsha	var@123	8WH55YWLUF	0
NULL	NULL	NULL	NULL

login 1 ×

MENU TABLE

Res_ID	ItemID	Item_name	Price	Image	quantity	last_modified
R001	I001	Biryani	300.00	BLOB	2	2024-11-13 12:32:18
R001	I005	Kebab	180.00	BLOB	5	2024-11-13 12:32:18
R001	I006	Chicken chilli	130.00	BLOB	8	2024-11-13 12:32:18
R001	I007	Chicken 65	150.00	BLOB	10	2024-11-13 12:32:18
R001	I008	Chicken chops	180.00	BLOB	10	2024-11-13 12:32:18
R001	I009	Fish Tawa fry	300.00	BLOB	10	2024-11-13 12:32:18
R001	I010	Egg Fry	120.00	BLOB	10	2024-11-13 12:32:18
R001	I011	Egg Masala	100.00	BLOB	9	2024-11-13 12:32:18
R001	I012	Mutton chops	260.00	BLOB	6	2024-11-13 12:34:06
R001	I088	shrek	2.00	BLOB	20	2024-11-13 12:34:59
R002	I002	Fish Fry	450.00	BLOB	10	2024-11-13 12:32:18
R002	I005	Biriyani	300.00	BLOB	10	2024-11-13 12:32:18

ORDERS TABLE

OrderID	Order_date	Passenger_id	R_ID	S_ID
22a6ff73	2024-11-12	KA62M4K5WN	R001	S001
3c916d7f	2024-11-11	KA62M4K5WN	R004	S004
5152b62f	2024-11-12	KA62M4K5WN	R001	S001
53aa5981	2024-11-12	KA62M4K5WN	R001	S001
6166ff5b	2024-11-12	KA62M4K5WN	R001	S001
8acf27da	2024-11-12	KA62M4K5WN	R001	S001
936db8bc	2024-11-12	KA62M4K5WN	R001	S001
ad7da0f3	2024-11-12	KA62M4K5WN	R001	S001
cf24e4c3	2024-11-12	KA62M4K5WN	R004	S004
dab37cd8	2024-11-12	KA62M4K5WN	R001	S001
dafbbd85	2024-11-10	KA62M4K5WN	R003	S001
efdd2e47	2024-11-12	KA62M4K5WN	R001	S001
NULL	NULL	NULL	NULL	NULL

orders 1 ×

PAYMENT TABLE

order_id	paymentID	payment_method	date
22a6ff73	d85467dc	DebitCard	2024-11-12
3c916d7f	c6e6aa4c	DebitCard	2024-11-11
5152b62f	55e94d94	BankTransfer	2024-11-12
6166ff5b	32fbd300	CreditCard	2024-11-12
936db8bc	8db48b79	BankTransfer	2024-11-12
cf24e4c3	3616b8f6	DebitCard	2024-11-12
dab37cd8	af29197c	Cash	2024-11-12
dafbbd85	53ba67eb	DebitCard	2024-11-10
NULL	NULL	NULL	NULL

RESTAURANTS

RID	Rname	Location	Phno	Station_id
R001	Donne Biryani	Bangalore	9876543212	S001
R002	Hyderabad Biryani	Hyderabad	9871203456	S003
R003	Burger King	Bangalore	9123456789	S001
R004	Nandhana Palace	Mysore	7979075750	S004
R005	Meghana Foods	Chennai	9595035350	S002
R006	Paradise Biryani	Mysore	8197181971	S004
R007	Nagarjuna's	Hyderabad	6780998769	S003
R008	Taaza Upahar	Chennai	7864350298	S002
NULL	NULL	NULL	NULL	NULL

STATIONS

StationID	Sname
S001	Bangalore
S002	Chennai
S003	Hyderabad
S004	Mysore
NULL	NULL

TRAINS

TrainID	TrainName
T001	ExpressOne
T004	Hindustanraill
T002	SuperFast
T003	Wondertrain
NULL	NULL

STOPS

Train_id	Station_id
T001	S001
T002	S001
T003	S001
T004	S001
T001	S002
T004	S002
T002	S003
T003	S003
T002	S004
T003	S004
T004	S004
NULL	NULL

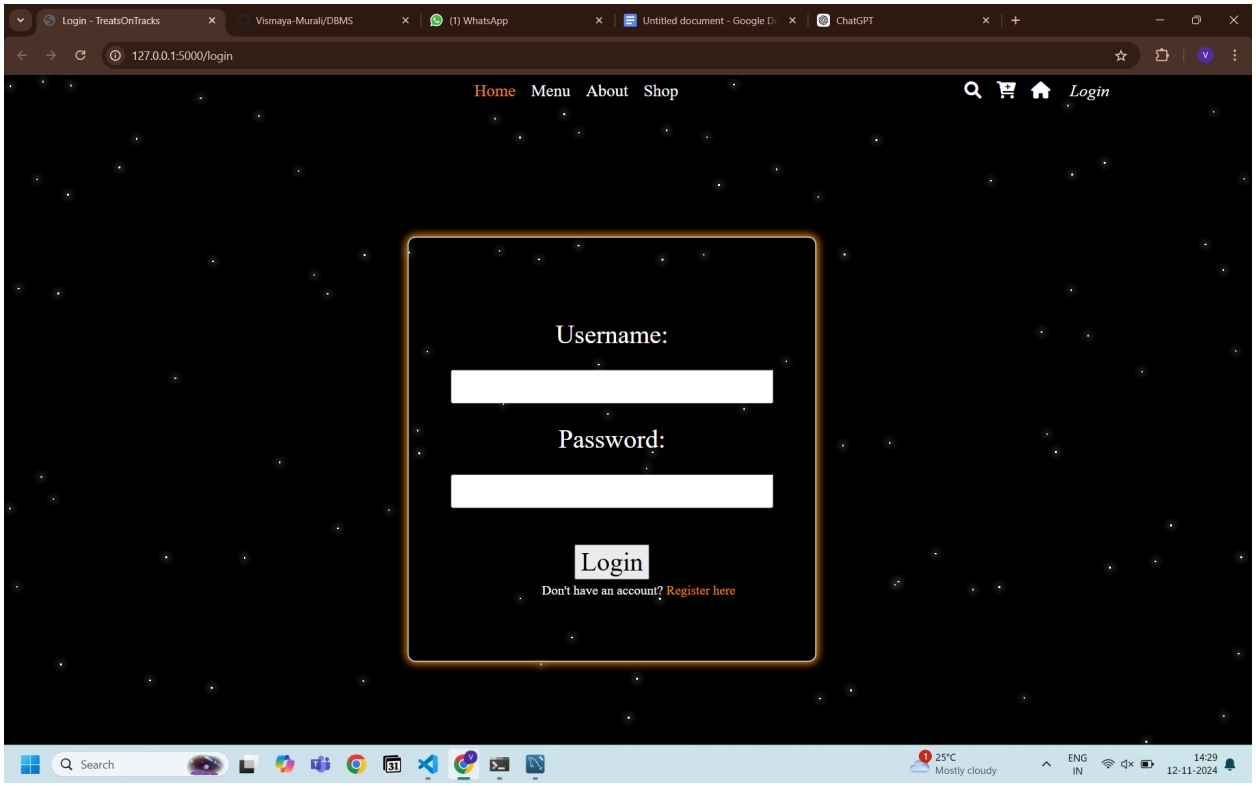
stops 1

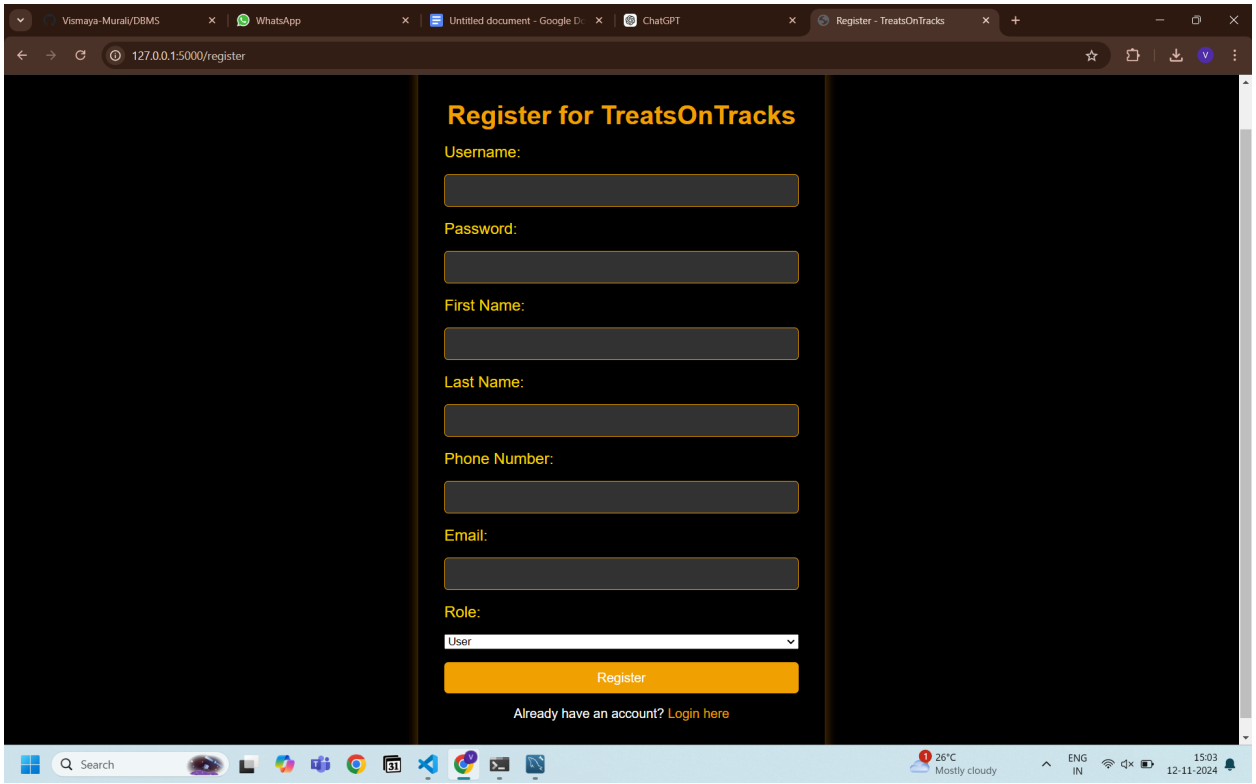
ORDER_ITEMS

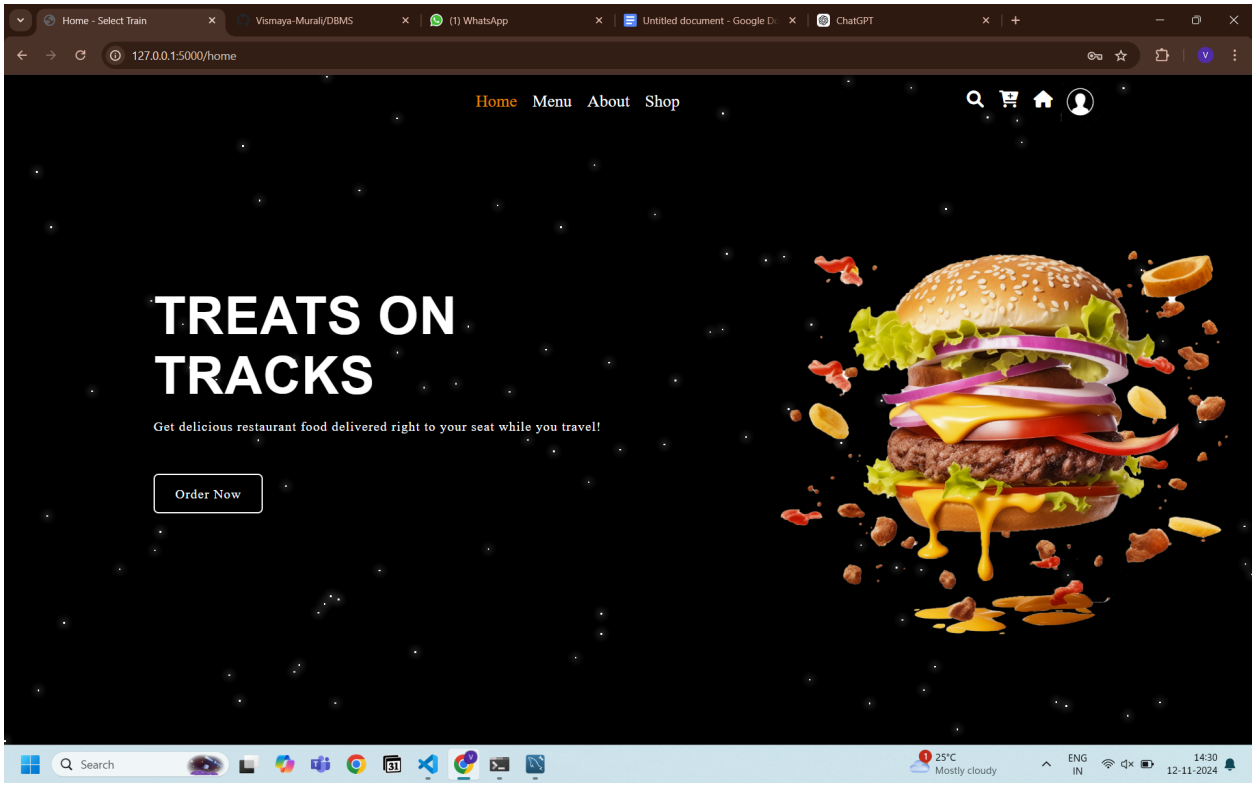
OrdItemID	ItemPrice	quantity	Order_id	R_ID	ItemID
0557dafb	300.00	8	8acf27da	R001	I001
0c519111	300.00	1	58b0cb6e	R001	I009
382af58b	180.00	1	cb91dfd2	R001	I005
39479bc6	300.00	1	53aa5981	R001	I001
3a362e80	300.00	1	fe1a86f9	R001	I009
3ec8f621	250.00	2	3c916d7f	R004	I032
4dc470a7	130.00	1	efdd2e47	R001	I006
57b8059a	180.00	3	5152b62f	R001	I005
5934a100	350.00	1	cf24e4c3	R004	I033
599182e2	130.00	1	01735ed9	R001	I006
5a6471a4	300.00	1	03e96a1b	R001	I009
6ebc8bf1	350.00	1	3c916d7f	R004	I033
742e0df9	150.00	2	53aa5981	R001	I007

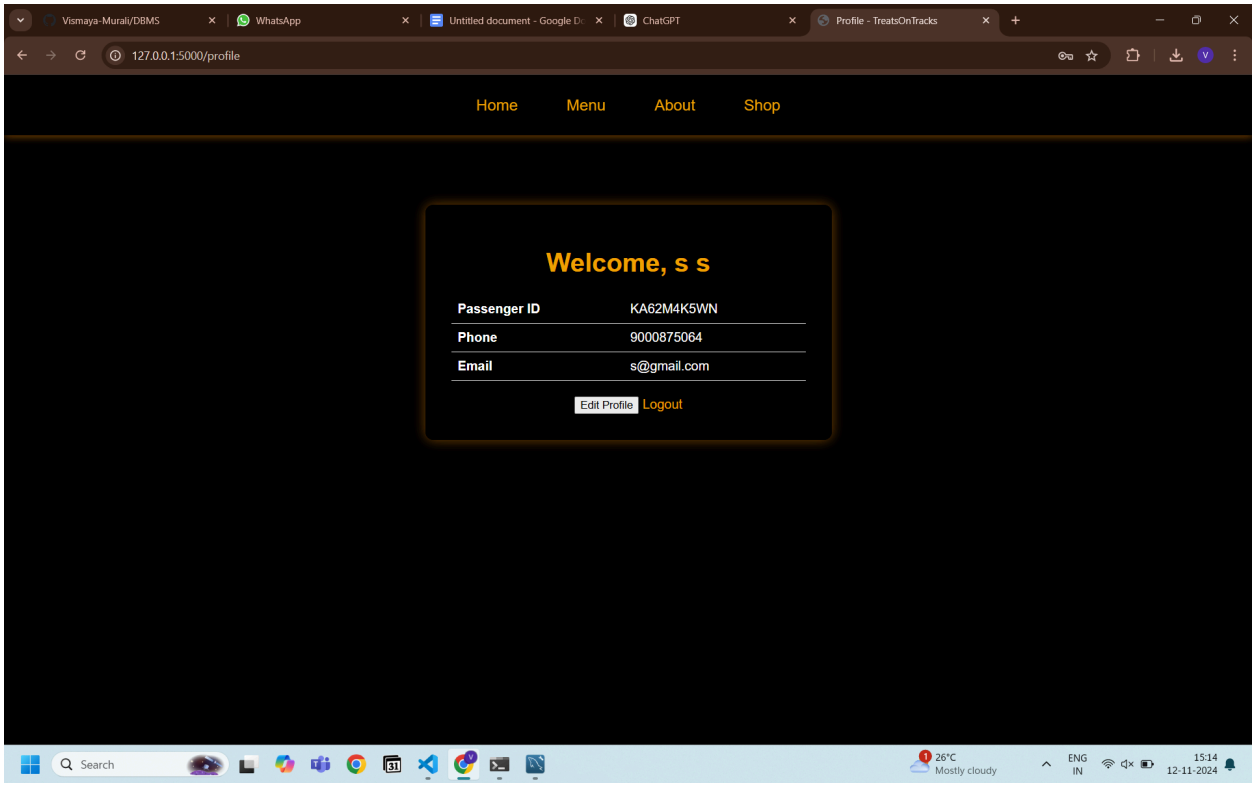
orderitems 1

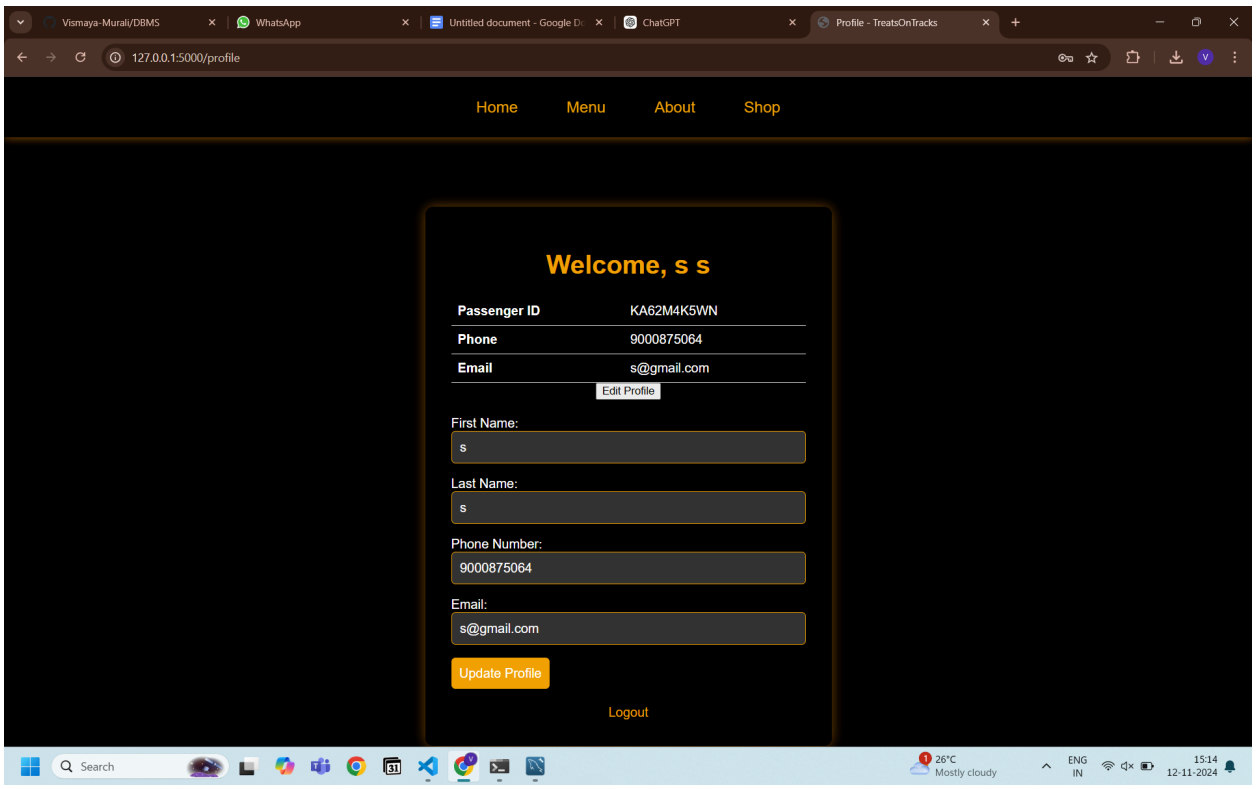
FRONTEND

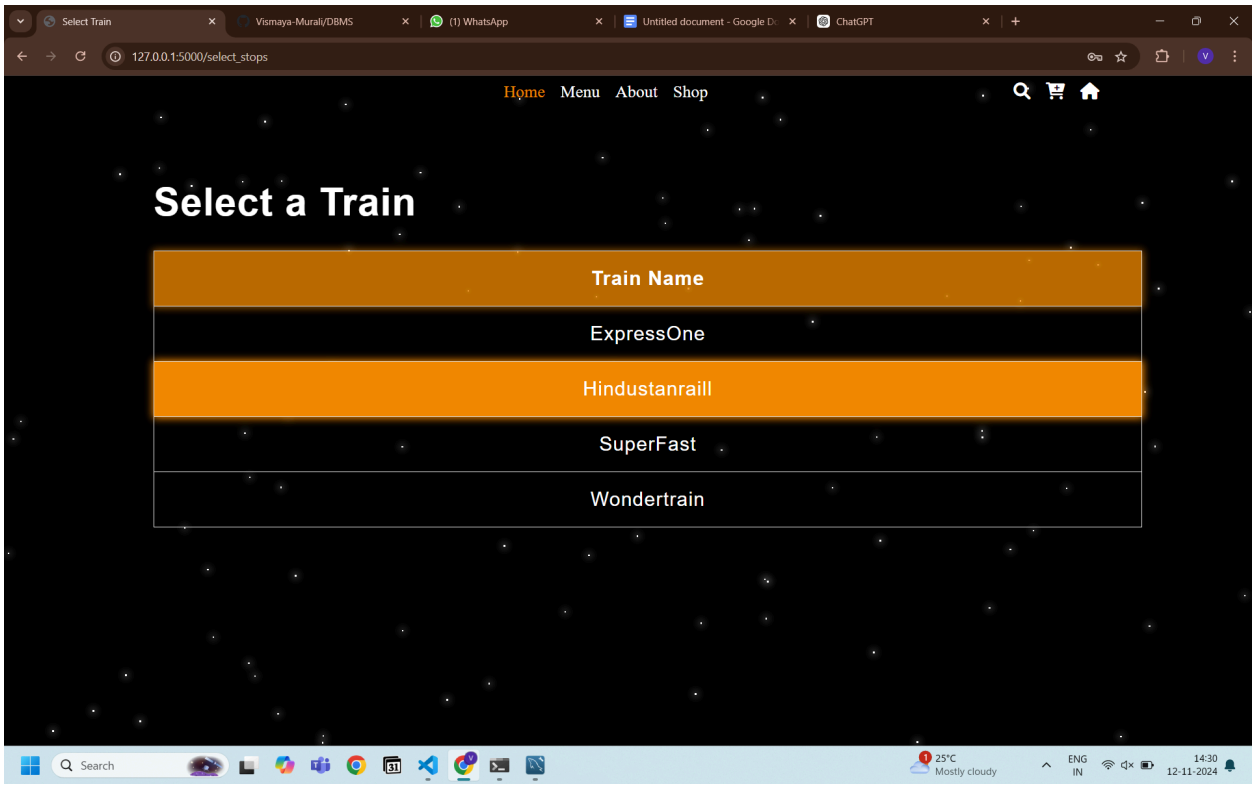

Login - TreatsOnTracks
Vismaya-Murali/DBMS
(1) WhatsApp
Untitled document - Google Dc
ChatGPT
127.0.0.1:5000/login
Home
Menu
About
Shop
Login
Username:
Password:
Login
Don't have an account? Register here
Search
25°C
Mostly cloudy
ENG
IN
14:29
12-11-2024

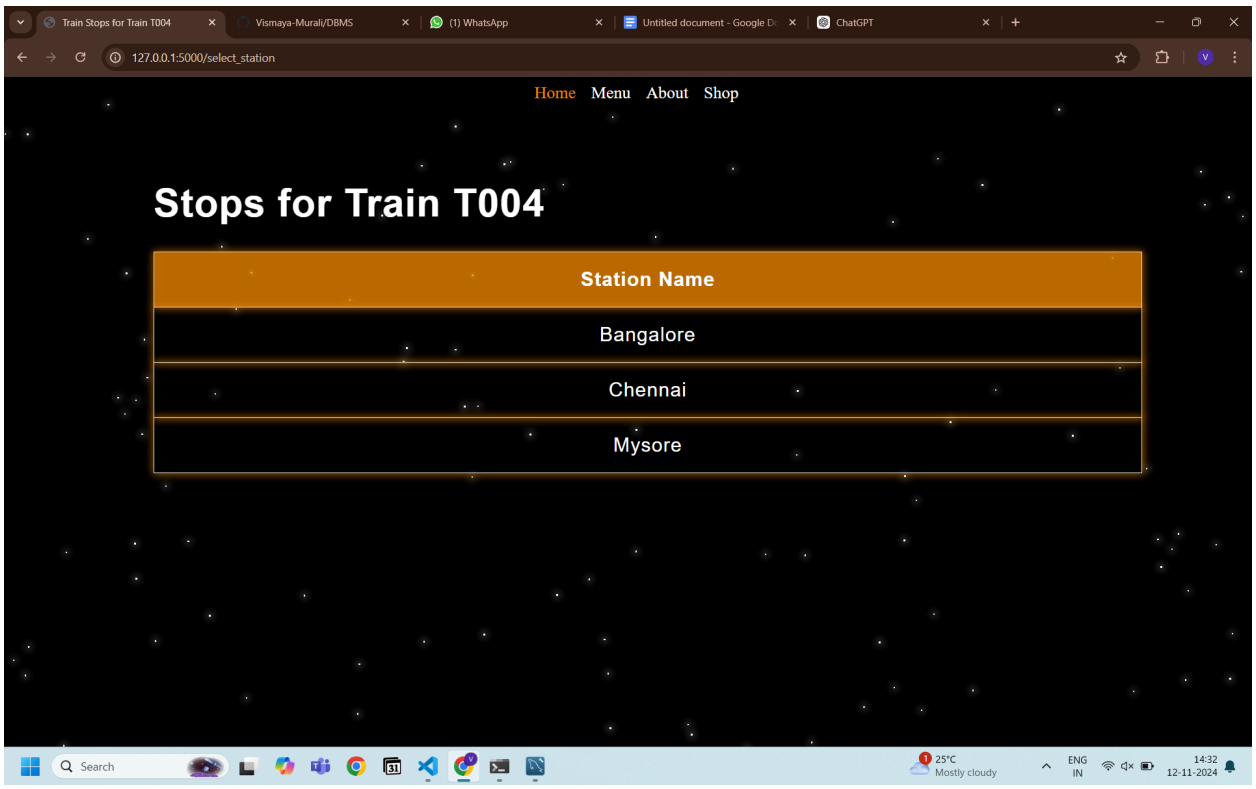

Vismaya-Murali/DBMS
WhatsApp
Untitled document - Google Dc
ChatGPT
Register - TreatsOnTracks
127.0.0.1:5000/register
Register for TreatsOnTracks
Username:
Password:
First Name:
Last Name:
Phone Number:
Email:
Role:
User
Register
Already have an account? Login here
Search
26°C
Mostly cloudy
ENG
IN
15:03
12-11-2024

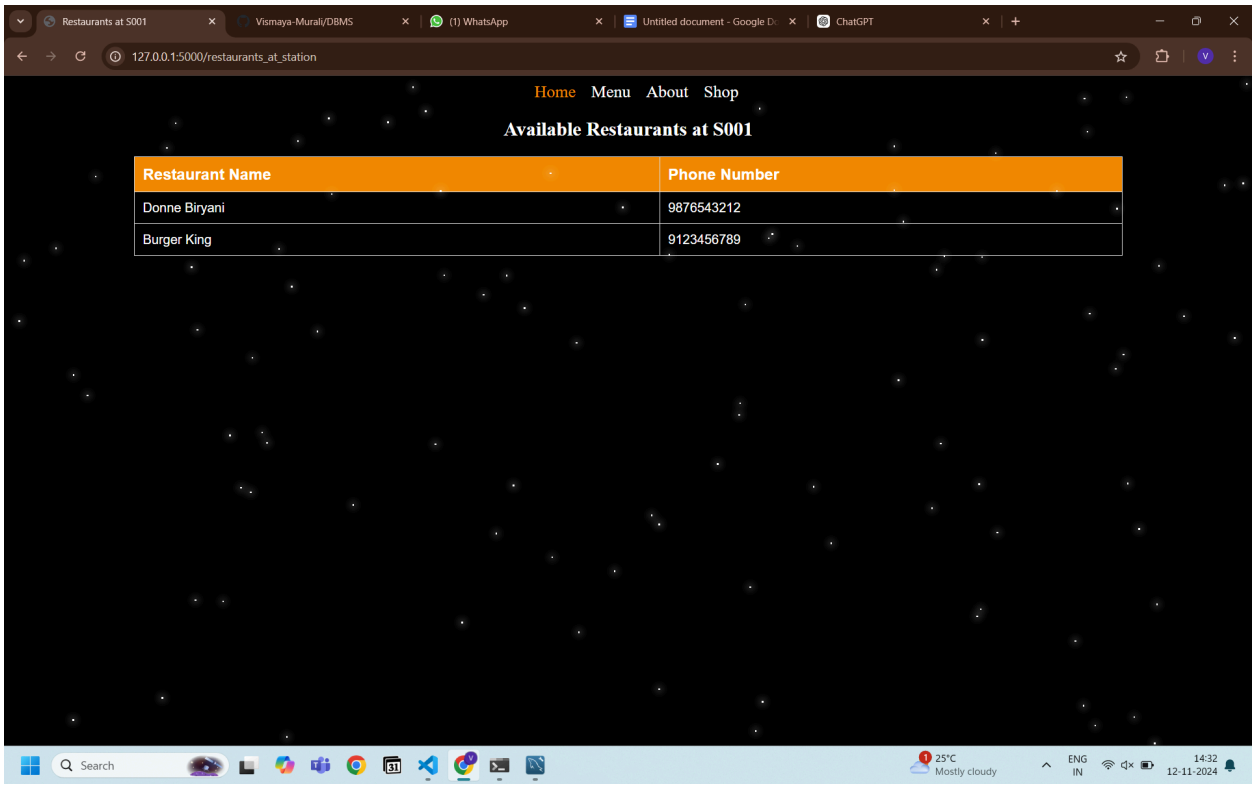

Home - Select Train
Vismaya-Murali/DBMS
(1) WhatsApp
Untitled document - Google D
ChatGPT
127.0.0.1:5000/home
Home Menu About Shop
TREATS ON TRACKS
Get delicious restaurant food delivered right to your seat while you travel!
Order Now
Search
25°C
Mostly cloudy
ENG
IN
14:30
12-11-2024

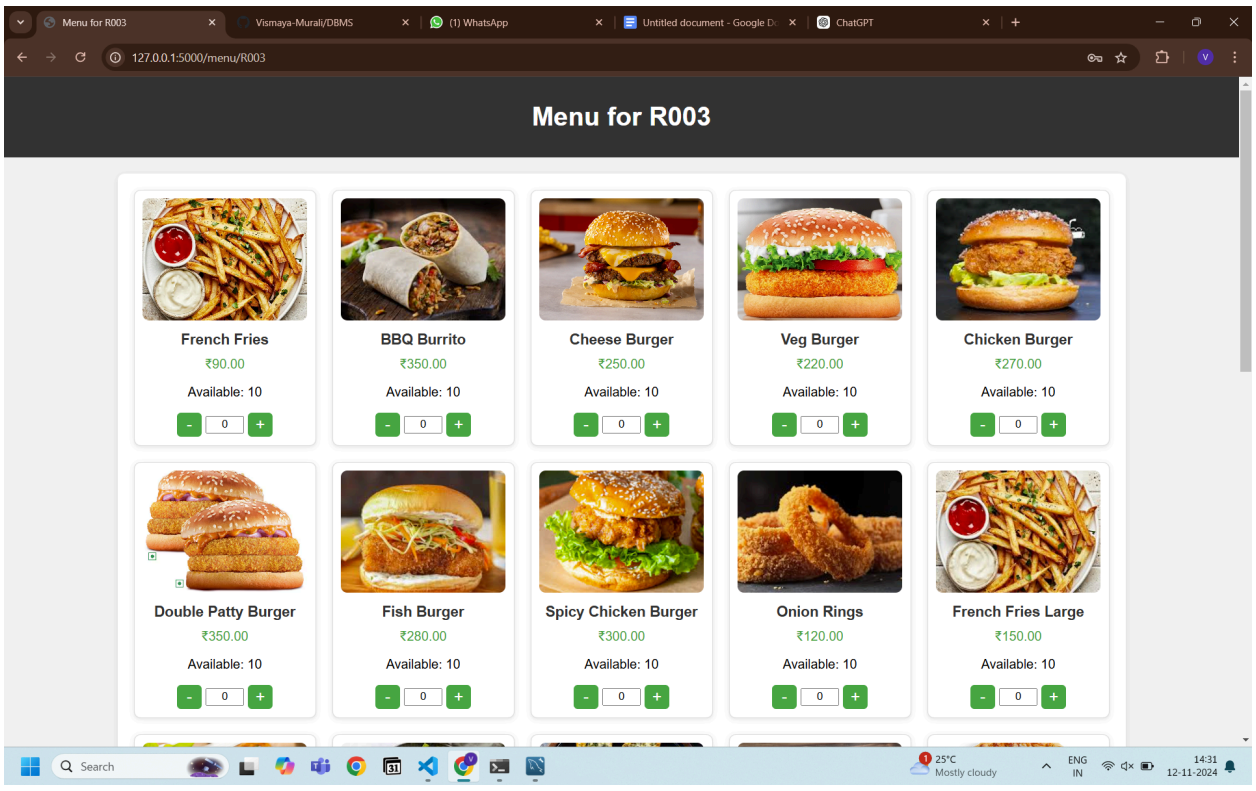

Vismaya-Murali/DBMS
WhatsApp
Untitled document - Google D
ChatGPT
Profile - TreatsOnTracks
127.0.0.1:5000/profile
Home Menu About Shop
Welcome, s s
Passenger ID KA62M4K5WN
Phone 9000875064
Email s@gmail.com
Edit Profile Logout
Search
26°C
Mostly cloudy
ENG
IN
15:14
12-11-2024

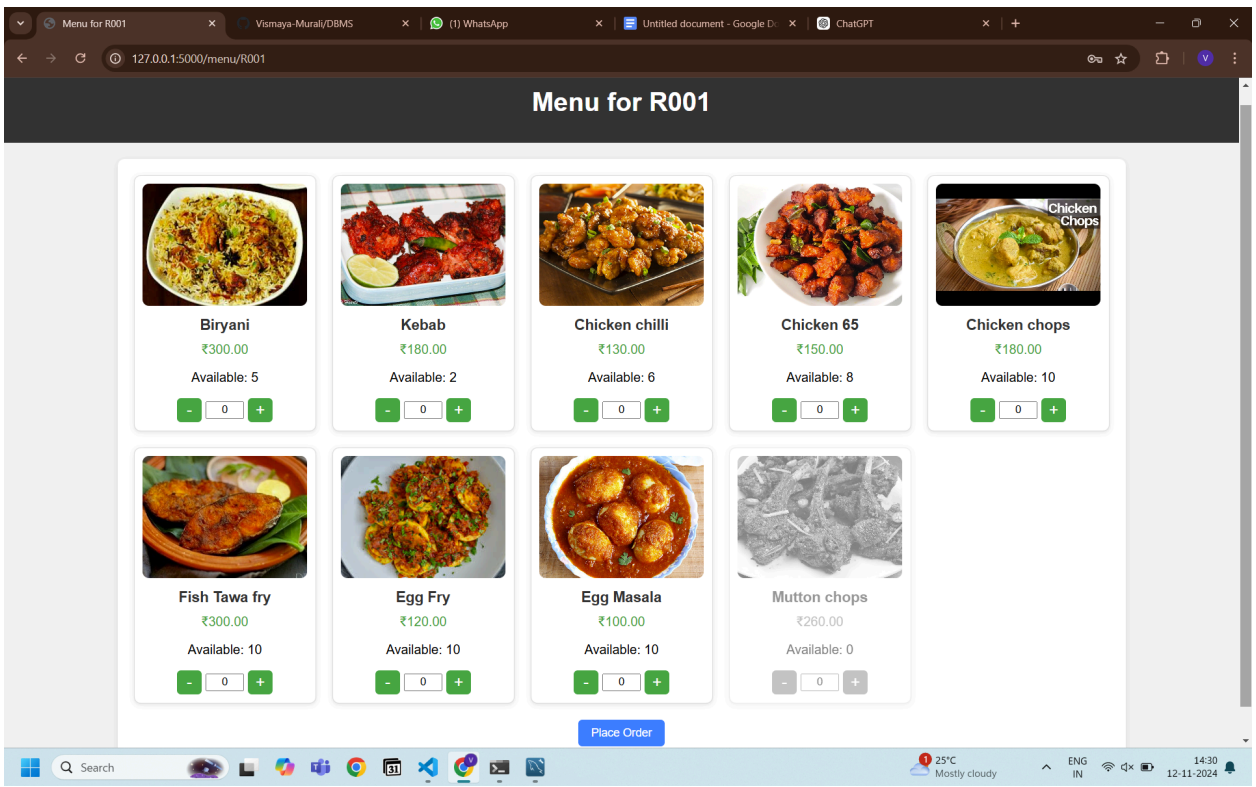

Vismaya-Murali/DBMS
WhatsApp
Untitled document - Google Dc
ChatGPT
Profile - TreatsOnTracks
127.0.0.1:5000/profile
Home
Menu
About
Shop
Welcome, s s
Passenger ID
KA62M4K5WN
Phone
9000875064
Email
s@gmail.com
Edit Profile
First Name:
s
Last Name:
s
Phone Number:
9000875064
Email:
s@gmail.com
Update Profile
Logout
Search
26°C
Mostly cloudy
ENG
IN
15:14
12-11-2024

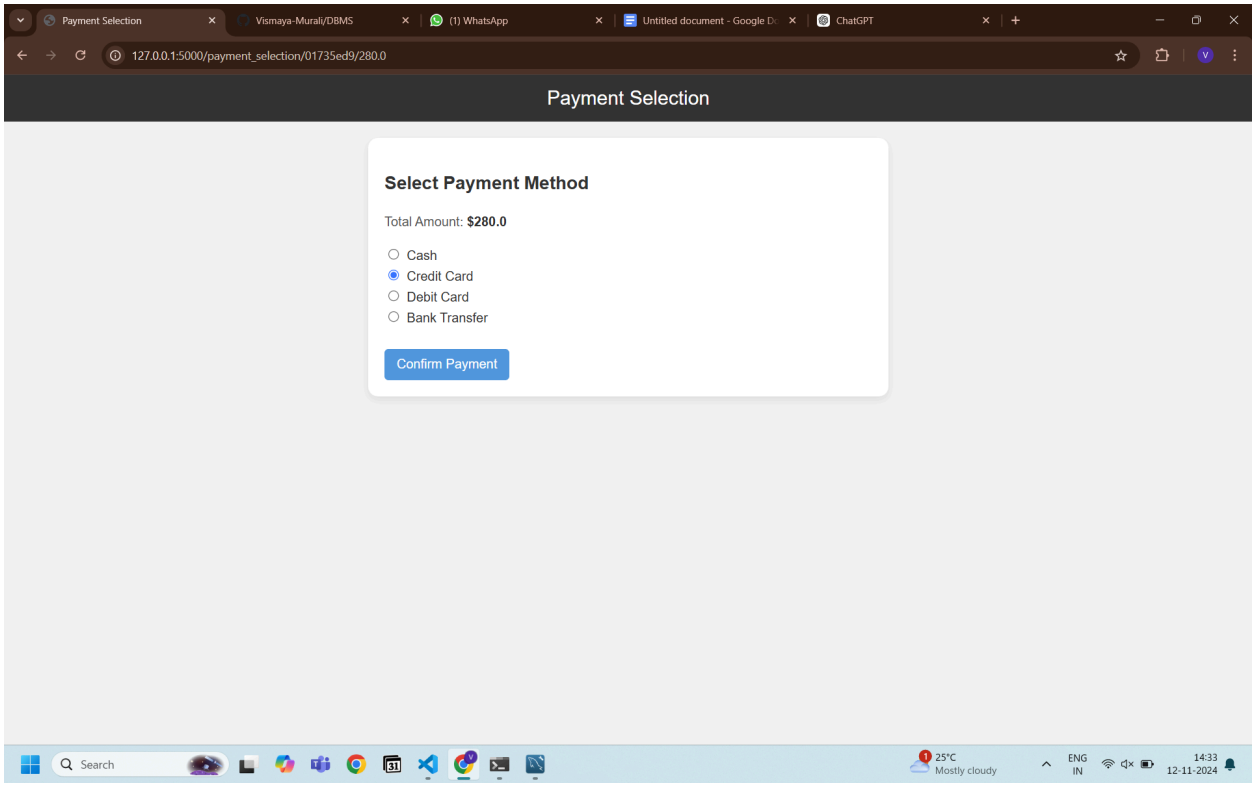

Select Train
Vismaya-Murali/DBMS
(1) WhatsApp
Untitled document - Google Dc
ChatGPT
127.0.0.1:5000/select_stops
Home
Menu
About
Shop
Select a Train
Train Name
ExpressOne
Hindustanraill
SuperFast
Wondertrain
Search
25°C
Mostly cloudy
ENG
IN
14:30
12-11-2024

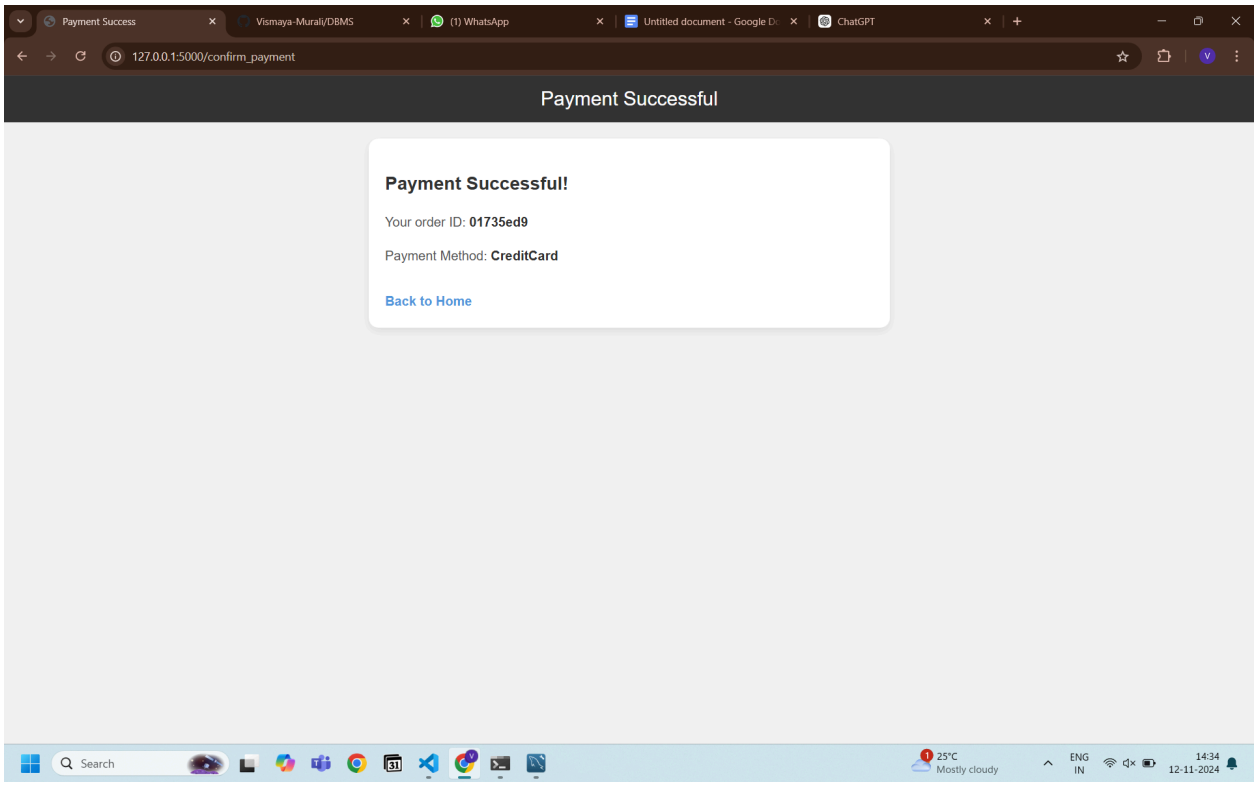

Train Stops for Train T004
Vismaya-Murali/DBMS
(1) WhatsApp
Untitled document - Google D
ChatGPT
127.0.0.1:5000/select_station
Home Menu About Shop
Stops for Train T004
Station Name
Bangalore
Chennai
Mysore
Search
25°C
Mostly cloudy
ENG
IN
14:32
12-11-2024


Restaurants at S001
Vismaya-Murali/DBMS
(1) WhatsApp
Untitled document - Google D
ChatGPT
127.0.0.1:5000/restaurants_at_station
Home Menu About Shop
Available Restaurants at S001
Restaurant Name
Phone Number
Donne Biryani
9876543212
Burger King
9123456789
Search
25°C
Mostly cloudy
ENG
IN
14:32
12-11-2024

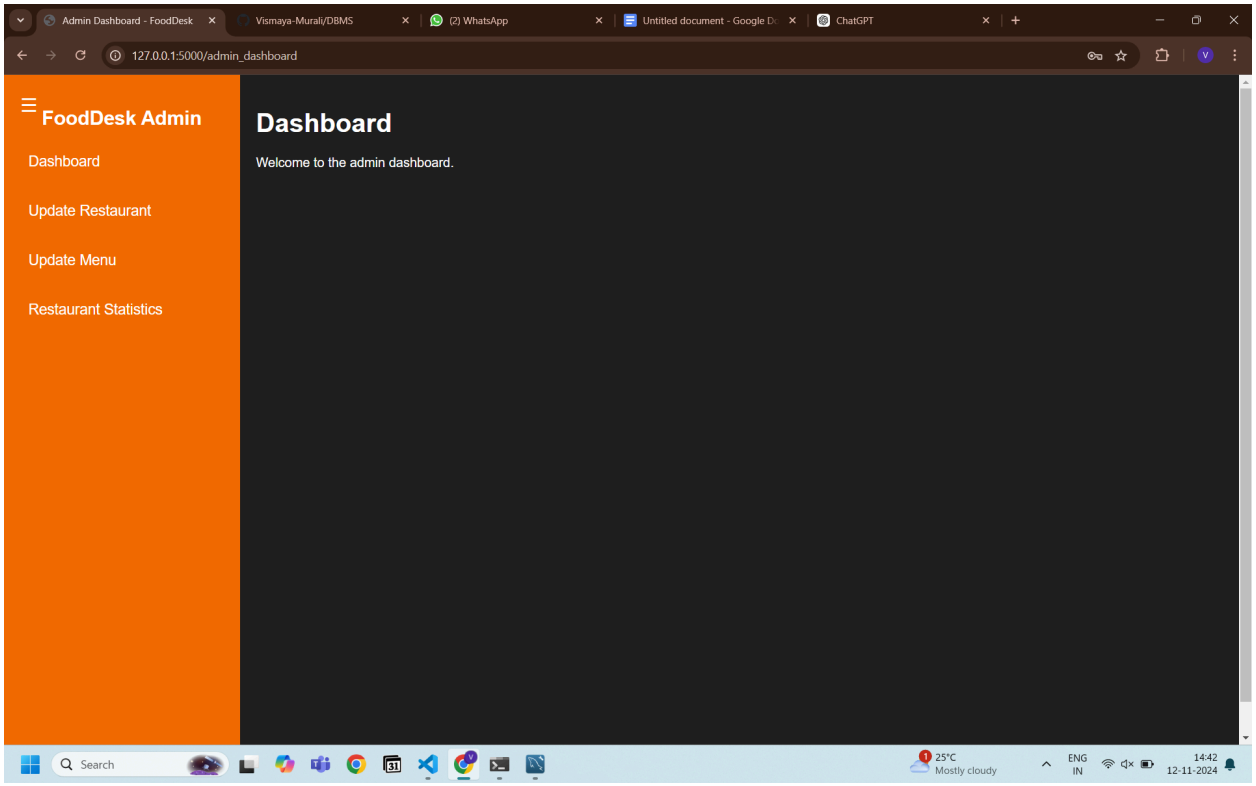

Menu for R003
127.0.0.1:5000/menu/R003
Menu for R003
French Fries
₹90.00
Available: 10
BBQ Burrito
₹350.00
Available: 10
Cheese Burger
₹250.00
Available: 10
Veg Burger
₹220.00
Available: 10
Chicken Burger
₹270.00
Available: 10
Double Patty Burger
₹350.00
Available: 10
Fish Burger
₹280.00
Available: 10
Spicy Chicken Burger
₹300.00
Available: 10
Onion Rings
₹120.00
Available: 10
French Fries Large
₹150.00
Available: 10

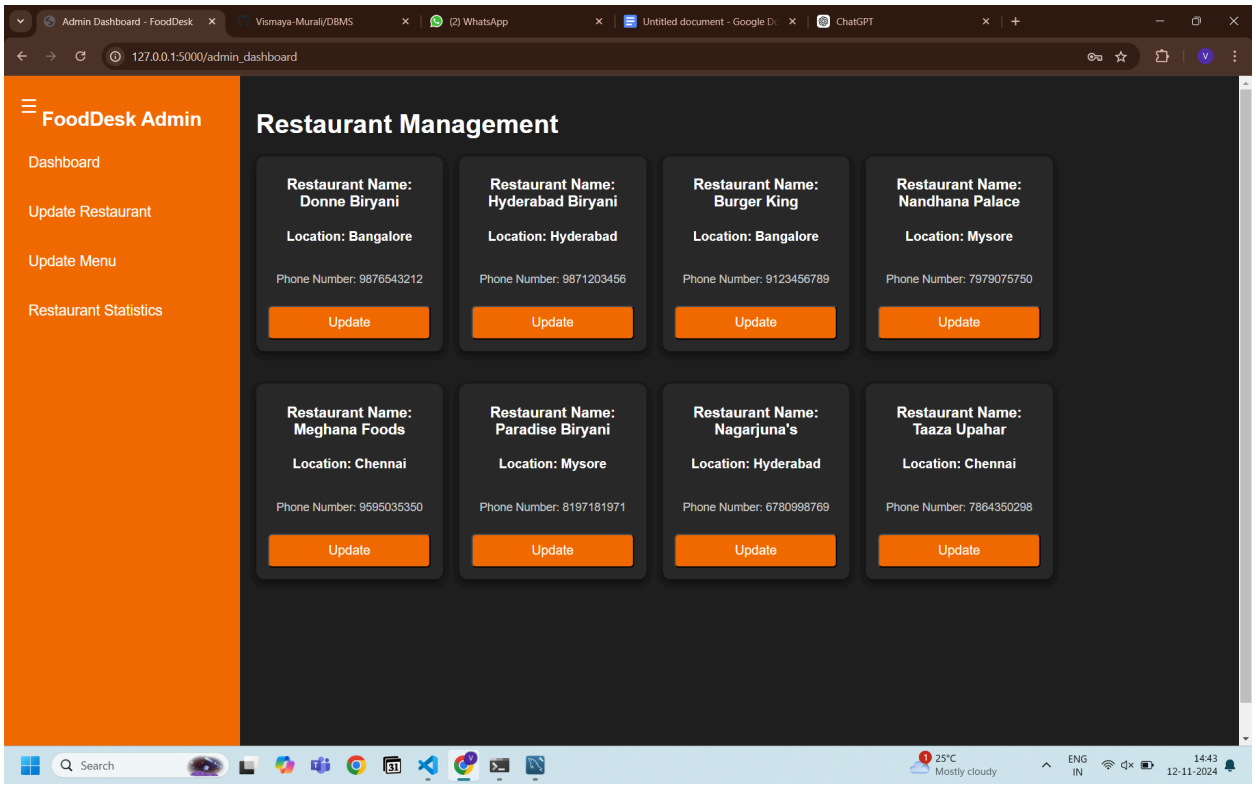

Menu for R001
127.0.0.1:5000/menu/R001
Menu for R001
Biryani
₹300.00
Available: 5
Kebab
₹180.00
Available: 2
Chicken chilli
₹130.00
Available: 6
Chicken 65
₹150.00
Available: 8
Chicken chops
₹180.00
Available: 10
Fish Tawa fry
₹300.00
Available: 10
Egg Fry
₹120.00
Available: 10
Egg Masala
₹100.00
Available: 10
Mutton chops
₹260.00
Available: 0
Place Order

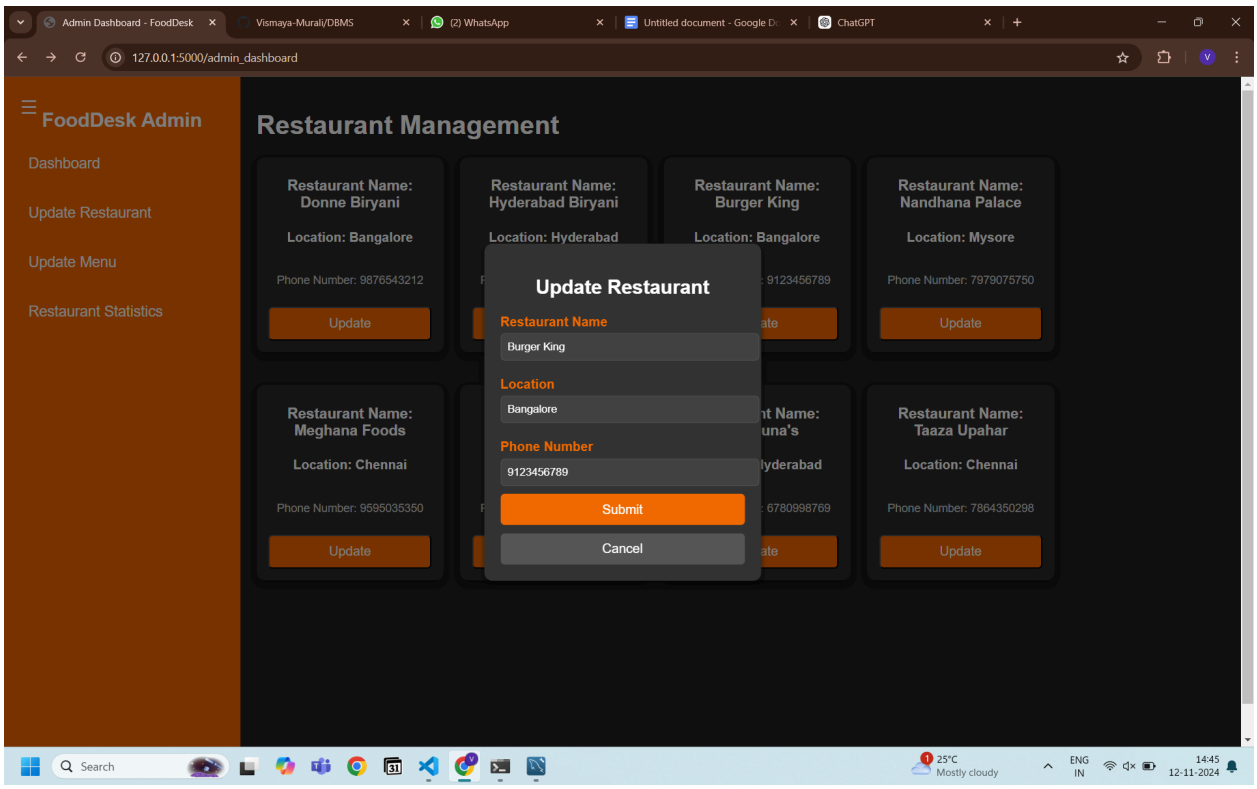

Payment Selection
Vismaya-Murali/DBMS
(1) WhatsApp
Untitled document - Google D
ChatGPT
127.0.0.1:5000/payment_selection/01735ed9/280.0
Payment Selection
Select Payment Method
Total Amount: $280.0
Cash
Credit Card
Debit Card
Bank Transfer
Confirm Payment
Search
25°C
Mostly cloudy
ENG
IN
14:33
12-11-2024

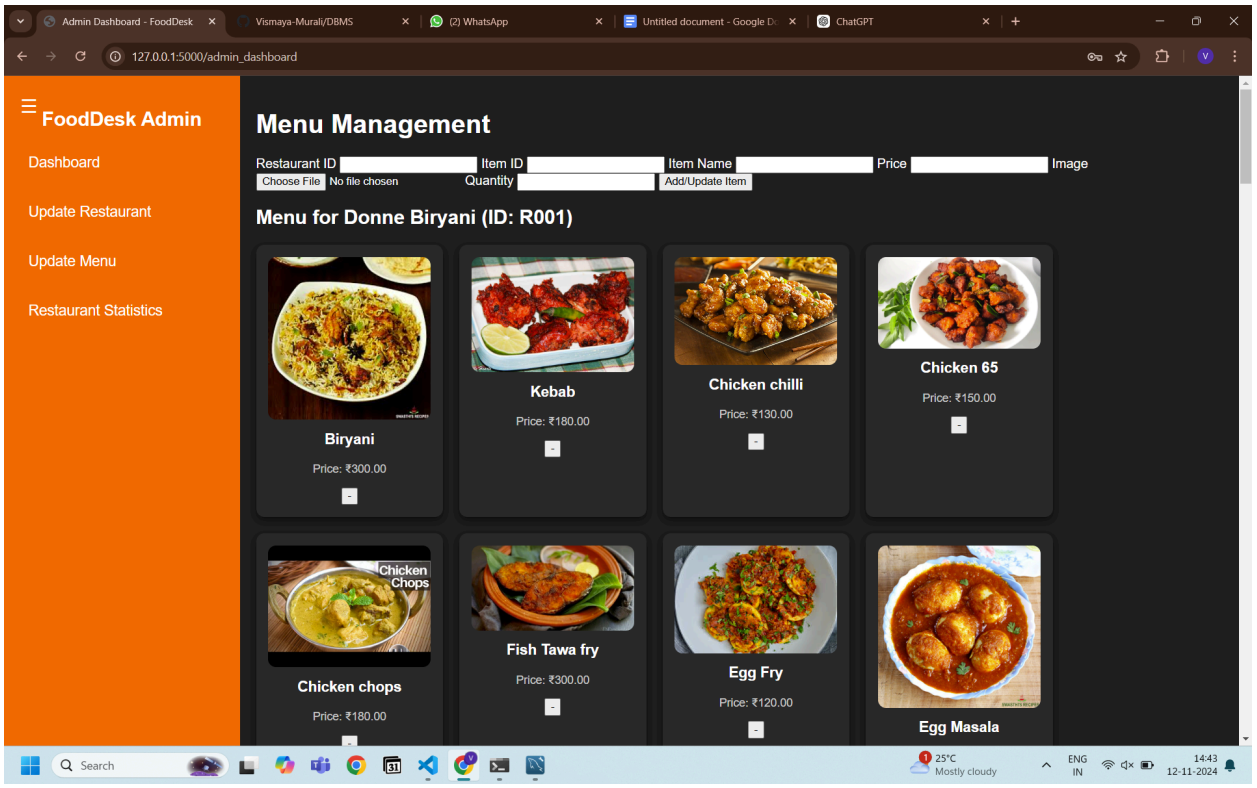

Payment Success
Vismaya-Murali/DBMS
(1) WhatsApp
Untitled document - Google D
ChatGPT
127.0.0.1:5000/confirm_payment
Payment Successful
Payment Successful!
Your order ID: 01735ed9
Payment Method: CreditCard
Back to Home
Search
25°C
Mostly cloudy
ENG
IN
14:34
12-11-2024

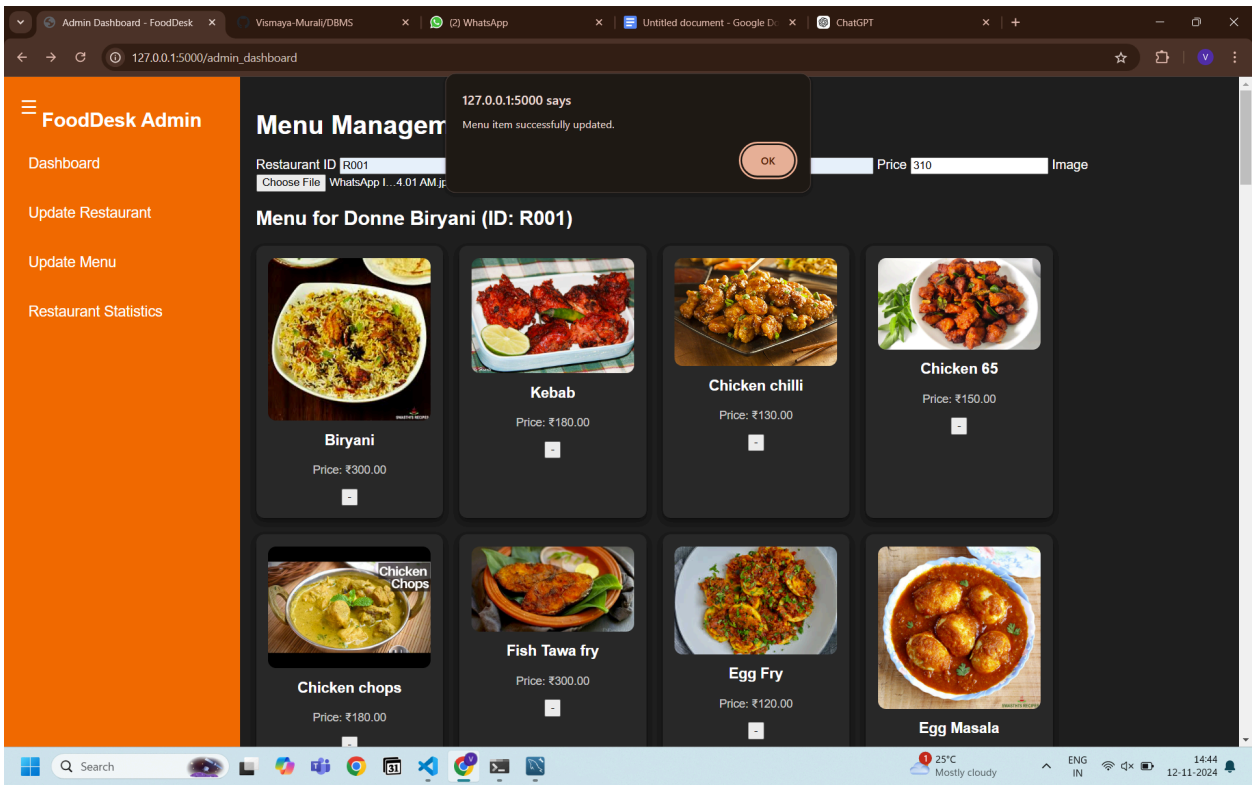
ADMIN

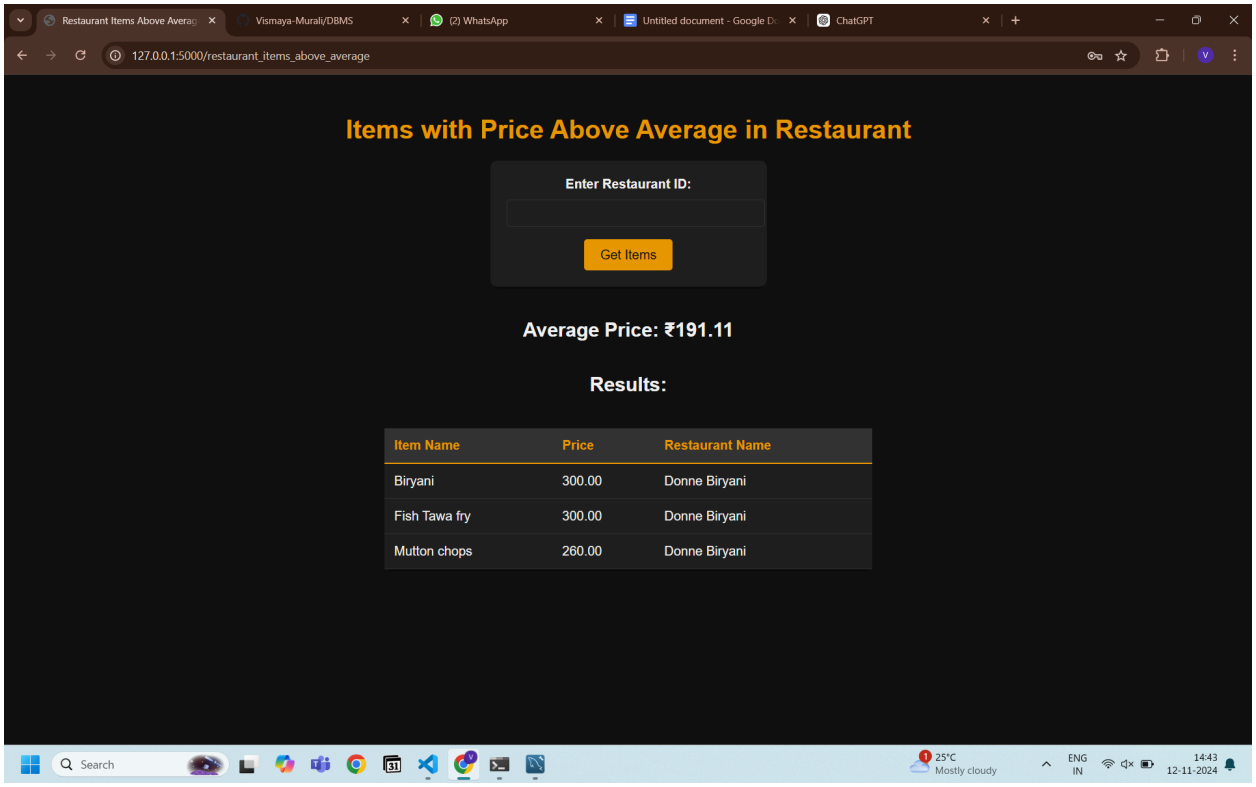

Admin Dashboard - FoodDesk
Vismaya-Murali/DBMS
(2) WhatsApp
Untitled document - Google Dr
ChatGPT
127.0.0.1:5000/admin_dashboard
FoodDesk Admin
Dashboard
Update Restaurant
Update Menu
Restaurant Statistics
Dashboard
Welcome to the admin dashboard.
Search
25°C
Mostly cloudy
ENG
IN
14:42
12-11-2024


Admin Dashboard - FoodDesk
Vismaya-Murali/DBMS
(2) WhatsApp
Untitled document - Google Dr
ChatGPT
127.0.0.1:5000/admin_dashboard
FoodDesk Admin
Dashboard
Update Restaurant
Update Menu
Restaurant Statistics
Restaurant Management
Restaurant Name: Donne Biryani
Location: Bangalore
Phone Number: 9876543212
Update
Restaurant Name: Hyderabad Biryani
Location: Hyderabad
Phone Number: 9871203456
Update
Restaurant Name: Burger King
Location: Bangalore
Phone Number: 9123456789
Update
Restaurant Name: Nandhana Palace
Location: Mysore
Phone Number: 7979075750
Update
Restaurant Name: Meghana Foods
Location: Chennai
Phone Number: 9595035350
Update
Restaurant Name: Paradise Biryani
Location: Mysore
Phone Number: 8197181971
Update
Restaurant Name: Nagarjuna's
Location: Hyderabad
Phone Number: 6780998769
Update
Restaurant Name: Taaza Upahar
Location: Chennai
Phone Number: 7864350298
Update
Search
25°C
Mostly cloudy
ENG
IN
14:43
12-11-2024


Admin Dashboard - FoodDesk
127.0.0.1:5000/admin_dashboard
FoodDesk Admin
Dashboard
Update Restaurant
Update Menu
Restaurant Statistics
Restaurant Management
Restaurant Name: Donne Biryani
Location: Bangalore
Phone Number: 9876543212
Update
Restaurant Name: Hyderabad Biryani
Location: Hyderabad
Restaurant Name: Burger King
Location: Bangalore
Restaurant Name: Nandhana Palace
Location: Mysore
Phone Number: 7979075750
Update
Update Restaurant
Restaurant Name
Burger King
Location
Bangalore
Phone Number
9123456789
Submit
Cancel
Restaurant Name: Meghana Foods
Location: Chennai
Phone Number: 9595035350
Update
Restaurant Name: Taaza Upahar
Location: Chennai
Phone Number: 7864350298
Update


Admin Dashboard - FoodDesk
127.0.0.1:5000/admin_dashboard
FoodDesk Admin
Dashboard
Update Restaurant
Update Menu
Restaurant Statistics
Menu Management
Restaurant ID
Item ID
Item Name
Price
Image
Choose File
No file chosen
Quantity
Add/Update Item
Menu for Donne Biryani (ID: R001)
Biryani
Price: ₹300.00
Kebab
Price: ₹180.00
Chicken chilli
Price: ₹130.00
Chicken 65
Price: ₹150.00
Chicken chops
Price: ₹180.00
Fish Tawa fry
Price: ₹300.00
Egg Fry
Price: ₹120.00
Egg Masala


Admin Dashboard - FoodDesk
127.0.0.1:5000/admin_dashboard
FoodDesk Admin
Dashboard
Update Restaurant
Update Menu
Restaurant Statistics
127.0.0.1:5000 says
Menu item successfully updated.
OK
Menu Managem
Restaurant ID R001
Price 310
Image
Menu for Donne Biryani (ID: R001)
Biryani
Price: ₹300.00
Kebab
Price: ₹180.00
Chicken chilli
Price: ₹130.00
Chicken 65
Price: ₹150.00
Chicken chops
Price: ₹180.00
Fish Tawa fry
Price: ₹300.00
Egg Fry
Price: ₹120.00
Egg Masala


Restaurant Items Above Averag
127.0.0.1:5000/restaurant_items_above_average
Items with Price Above Average in Restaurant
Enter Restaurant ID:
Get Items
Average Price: ₹191.11
Results:
Item Name
Price
Restaurant Name
Biryani
300.00
Donne Biryani
Fish Tawa fry
300.00
Donne Biryani
Mutton chops
260.00
Donne Biryani