DBMS PROJECT

RESTAURANT MANAGEMENT SYSTEM

FLAVOURS AND FABLES

Team members:

NIKKATH ZABEEN – 2023BCS0060

S S PAVAN CHARAN – 2023BCS0078

HARSHITHA – 2023BCS0069

ROSHITHA – 2023BCS0105

MAMALESH – 2023BCS0081

ROHITH – 2023BCY0030

TASK: To provide a user-friendly platform for a Restaurant to manage orders, Table-reservations, cart-items, user-Authentication and manage Employee-Authentication and Employee task.

Abstraction:

This platform is a comprehensive solution for restaurant management, designed to integrate multiple key functions into a single, user-friendly interface. It addresses the operational needs of both customers and employees, providing seamless management of orders, table reservations, cart items, and secure authentication for both users and staff. The system enhances efficiency, reduces manual errors, and creates a smooth dining experience through the following features:

1. **Order Management:**

- Allows restaurant staff to efficiently handle and track customer orders in real-time.
- Includes features for marking orders as completed and updating the order status instantly.
- Offers employees an easy-to-navigate interface for managing active and completed orders, reducing delays and errors.

2. **Table Reservation System:**

- Enables customers to reserve tables ahead of time through an intuitive booking system.
- Allows restaurant staff to view, manage, and update reservations, ensuring proper table allocation and smooth customer seating.
- Integrates with the order management system, ensuring orders are linked with specific table reservations.

3. **Cart System:**

- Provides customers with an easy-to-use cart interface for adding items and making final submissions.
- Enhances user experience by offering real-time updates to cart contents and order summaries.

4. **User Authentication:**

- Ensures secure customer login and registration using robust authentication mechanisms.
- Generates unique sessions for each user, safeguarding personal data and providing a personalized experience.
- Supports user data retention, allowing returning customers to view past orders, preferences, and active reservations.

5. **Employee Authentication and Task Management:**

- Implements secure employee login and role-based access to the platform.
- Employees can manage tasks such as processing orders and updating reservation statuses.

- Ensures only authorized staff can access sensitive operations, protecting the integrity of the platform.

6. Efficiency and Automation:

- The platform automates routine tasks such as updating order statuses, managing reservations, and organizing employee workflows.
- By reducing manual input, it minimizes human errors and optimizes the time required to serve customers, resulting in faster service.
- All features are integrated into a single, cohesive system, ensuring smooth communication between customers, staff, and management.

Objective:

The main objective of this platform is to create an efficient and user-friendly system that simplifies restaurant management and enhances both customer and staff experiences. It aims to automate critical tasks such as order processing, table reservations, cart management, and secure authentication while optimizing staff workflow. The extended objectives of this platform include:

1. **Streamline Order Management:**

- Provide a centralized interface for restaurant staff to manage all customer orders efficiently.
- Ensure real-time updates on order status, allowing staff to track the progress of orders and handle them in a timely manner.
- Enable easy completion of orders, reducing confusion in busy kitchen and service areas.

2. **Enhance Table Reservation Process:**

- Allow customers to reserve tables directly through the platform, reducing the need for manual reservation processes.
- Automatically link table reservations with orders, ensuring that customers receive the appropriate service upon arrival.

3. **Improve Cart and Checkout System:**

- Provide customers with a simple, intuitive cart interface that allows them to browse the menu, add items, and place orders with ease.
- Allow customers to modify their orders before checkout, improving user satisfaction by offering flexibility.

4. **Strengthen User Authentication:**

- Automatically generate unique user sessions upon login, providing a personalized experience with access to previous orders and preferences.
- Offer an easy registration process to encourage new users to sign up, while maintaining high levels of security.

5. **Secure Employee Authentication and Role Management:**

- Secure employee login with individual credentials, safeguarding sensitive restaurant data from unauthorized access.
- Allow employees to view and manage their tasks easily, streamlining their workflow and improving task accountability

6. **Foster Scalability and Flexibility:**

- Design the platform to be scalable, ensuring it can handle increased customer and employee traffic as the restaurant grows.
- Ensure flexibility in system features so that the platform can be adapted to different types of restaurants, from small cafes to larger establishments.
- Support future expansions such as adding delivery services, loyalty programs, or advanced analytics without disrupting current operations

Introduction

Managing the operations of a restaurant requires seamless coordination between various functions such as order management, table reservations, customer service, and employee task assignments. In a fast-paced restaurant environment, manually handling these tasks can lead to inefficiencies, errors, and customer dissatisfaction. To address these challenges, this platform offers a comprehensive solution that integrates order processing, table reservations, cart management, and secure user and employee authentication into one unified system.

The platform aims to optimize restaurant workflows by automating routine tasks, improving communication between the kitchen and service staff, and enhancing the overall customer experience. By incorporating features such as table booking, cart functionality, and role-based access for employees, the system ensures that restaurants can efficiently manage operations even during peak hours. Ultimately, this platform is designed to streamline restaurant management processes, reduce manual workloads, and improve customer satisfaction.

Methodology:

The development of this platform follows a structured approach to

ensure that all functional requirements are met while maintaining an intuitive user experience. The methodology is divided into the following stages:

1. **Requirements Gathering and Analysis**:

- A detailed analysis of restaurant management needs was focused, focusing on key areas such as order handling, table reservations, customer and employee authentication, and task assignments.
- Specific use cases were developed to outline the system's functionality, including scenarios for order management, reservation handling, and user authentication.

2. **System Design**:

- The platform was designed using a modular approach, allowing each feature (orders, reservations, cart, authentication, etc.) to function independently yet integrate seamlessly.
- A user-friendly interface was prioritized, ensuring both employees and customers can interact with the system effortlessly.
- Database design involved creating a relational schema that securely stores user, employee, order, and reservation data. Efficient indexing and query optimization were implemented to ensure fast response times, even under high traffic.

3. **Development**:

- The platform was developed using a combination of **Node.js** for the backend and **Express** to handle server-side operations. **MySQL** was used as the database to manage all records efficiently.
- **EJS** templates were used to render the frontend, ensuring dynamic content generation, while **Bootstrap** was employed to create a responsive, mobile-friendly interface.
- A robust authentication system was implemented, including role-based access for employees and secure session management for both users and employees.
- Real-time features, such as order status updates and table availability tracking, were developed to provide immediate feedback to both customers and staff.

4. **Future Enhancements**:

- The platform is designed with scalability in mind, allowing future expansion to include additional features like delivery services, loyalty programs, and advanced data analytics for better decision-making.
- Based on ongoing feedback from users and staff, new features and improvements will be integrated into the system to meet evolving restaurant management needs.

By following this structured methodology, the platform ensures a high level of reliability, efficiency, and user satisfaction, making it an indispensable tool for modern restaurant management.

DATABASE:

- **ER-DIAGRAM:**

- **DATABASE CREATION:**

I. User Table

Where user details stores when the sign-up
And this table is used for used functionality purposes.

```
CREATE TABLE users (
    user_id VARCHAR(50) PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    email VARCHAR(50) NOT NULL,
    password VARCHAR(255) NOT NULL
);
```

Field	Type	Null	Key	Default	Extra
user_id	varchar(50)	NO	PRI	NULL	
username	varchar(50)	NO		NULL	
email	varchar(50)	NO		NULL	
password	varchar(255)	NO		NULL	

II. Items table

Here in this table all the items will be stored in this.
This table is used for viewing menu

```
CREATE TABLE items (
    item_id INT PRIMARY KEY AUTO_INCREMEN
    title VARCHAR(255),    T,
    description TEXT,
    image_url VARCHAR(255),
    price DECIMAL(10, 2)
);
```

Field	Type	Null	Key	Default	Extra
item_id	int	NO	PRI	NULL	auto_increment
title	varchar(255)	YES		NULL	
description	text	YES		NULL	
image_url	varchar(255)	YES		NULL	
price	decimal(10,2)	YES		NULL	

III. Orders table

Here in this table all the which were placed by user will be stored in this.
The is_active is handled by employee

```sql
CREATE TABLE orders (
    order_id INT PRIMARY KEY AUTO_INCREMENT,
    user_id VARCHAR(36),  --For UUID
    total_amount DECIMAL(10, 2),
    order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

ALTER TABLE orders ADD COLUMN employee_id INT;

ALTER TABLE orders ADD COLUMN is_active BOOLEAN DEFAULT TRUE;
```

Field	Type	Null	Key	Default	Extra
order_id	int	NO	PRI	NULL	auto_increment
user_id	varchar(36)	YES		NULL	
total_amount	decimal(10,2)	YES		NULL	
order_date	timestamp	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED
is_active	tinyint(1)	YES		1	
employee_id	int	YES		NULL	

IV. Order_items table

Here in order_items what are all the items been placed will be stored by his and order_id created in the session.
This table is used for cart

```sql
CREATE TABLE order_items
    order_item_id INT PRIMARY KEY AUTO_INCREMENT,
    order_id INT,
    item_id INT,
    quantity INT,
    price DECIMAL(10, 2),
    FOREIGN KEY (order_id) REFERENCES orders(order_id),
    FOREIGN KEY (item_id) REFERENCES items(item_id)
);
```

Field	Type	Null	Key	Default	Extra
order_item_id	int	NO	PRI	NULL	auto_increment
order_id	int	YES	MUL	NULL	
item_id	int	YES	MUL	NULL	
quantity	int	YES		NULL	
price	decimal(10,2)	YES		NULL	

V. Reservations table

When user reserves table the details will be stored in this table
Further references to orders table

```sql
CREATE TABLE reservations (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id VARCHAR(36) NOT NULL, -- Foreign key to link with the use
    reservation_date DATE NOT NULL,
    reservation_hour TIME NOT NULL,
    phone_number VARCHAR(15) NOT NULL,
    occasion VARCHAR(50) NOT NULL,
    num_persons INT NOT NULL CHECK (num_persons > 0 AND num_persons <= 8),
    FOREIGN KEY (user_id) REFERENCE  users(user_id) -- Assuming a `users` table exist
);                        S                        s
ALTER TABLE reservations ADD table_number INT;
```

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
user_id	varchar(36)	NO	MUL	NULL	
reservation_date	date	NO		NULL	
reservation_hour	time	NO		NULL	
phone_number	varchar(15)	NO		NULL	
occasion	varchar(50)	NO		NULL	
num_persons	int	NO		NULL	
table_number	int	YES		NULL	

VI. Employee table

Pre existed details will be given in the employees table

```sql
CREATE TABLE employees (
    emp_id INT PRIMARY KEY AUTO_INCREMEN
    emp_name VARCHAR(100) NOT NULL,
    emp_email VARCHAR(100) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL
);
```

Field	Type	Null	Key	Default	Extra
emp_id	int	NO	PRI	NULL	auto_increment
emp_name	varchar(100)	NO		NULL	
emp_email	varchar(100)	NO	UNI	NULL	
passcode	varchar(255)	NO		NULL	

VII. Feedback table

Here feedback is stored where user is given and will this feedback in his employee authentication page.

```sql
CREATE TABLE feedback (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL,
    message TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
name	varchar(255)	NO		NULL	
email	varchar(255)	NO		NULL	
message	text	NO		NULL	
created_at	timestamp	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED

- **Queries used:**

❖ **User signup query:**

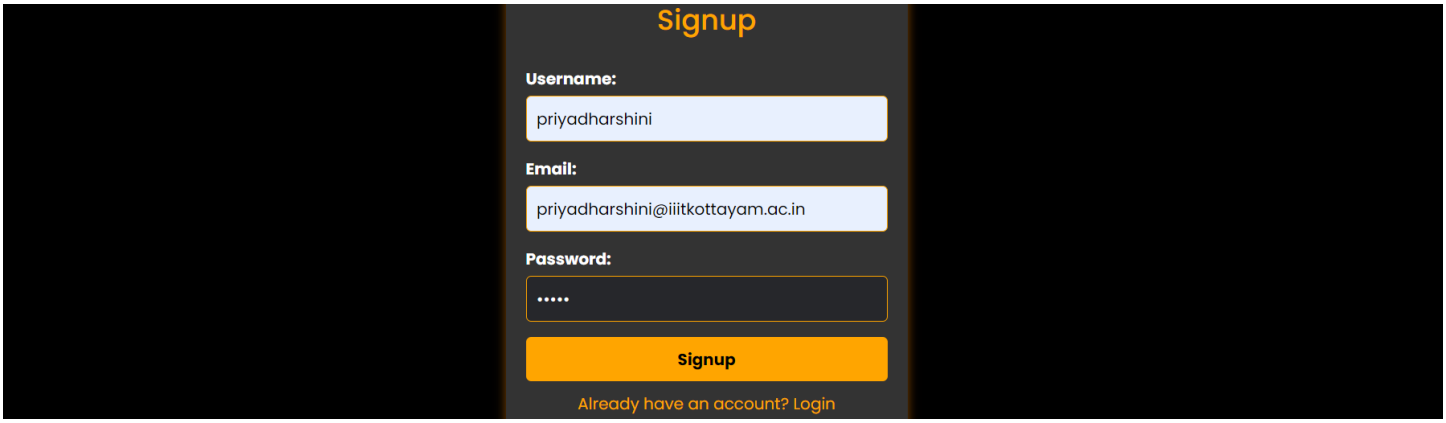
```
const connectio  = await pool.getConnectio ();
      ntry {                  n
            const [existingUse ] = await connectio .execute(
                'SELECT * FROM users WHERE username = ? OR email =   ,
                ?username, email]
[
);
            if (existingUse .length > 0) {
                // User with the same username or email already exist
                req.flash("error", "Username or email already registere   ); // Use flash messag
                 return res.redirect("/login/use  ); // Redirect to the signup pag
                                    r'              e
}
            // Hash the passwor
            donst hashedPasswor  = await bcrypt.hash(password, 10);
                  d
            // Generate a unique userI
            donst userId = uuidv4(); // Generate a UUI
                                    D
            // Insert the user into the databas
            await connectio .execute(
                'INSERT INTO users (user_id, username, email, password) VALUES (?, ?, ?,    ,
                ?userId, username, email, hashedPasswor ] // Include userId in the quer
[                                       d               y
);
            req.flash("success", "Registration successful! Please log i   ); // Use flash message for succes
            res.redirect('/login/use  ); // Redirecting to the login page after successful registratio
        } catch (error) {r'              n
            console.error('Error checking for existing user or inserting use   , error);
            req.flash("error", "Database error occurre   );
            res.redirect('/login/signu  ); // Redirect to the signup page on erro
        } finally {      p'                r
            connectio .release(); // Always release the connectio
        }   n                    n
    }
```

User first creates his account with his/her name , email, password here if the same username or email is already existed in the database his/her account can't be created.

Here are the sql queries for the above scenario

Here when user creates his account the details are stored in the database.

Testing:



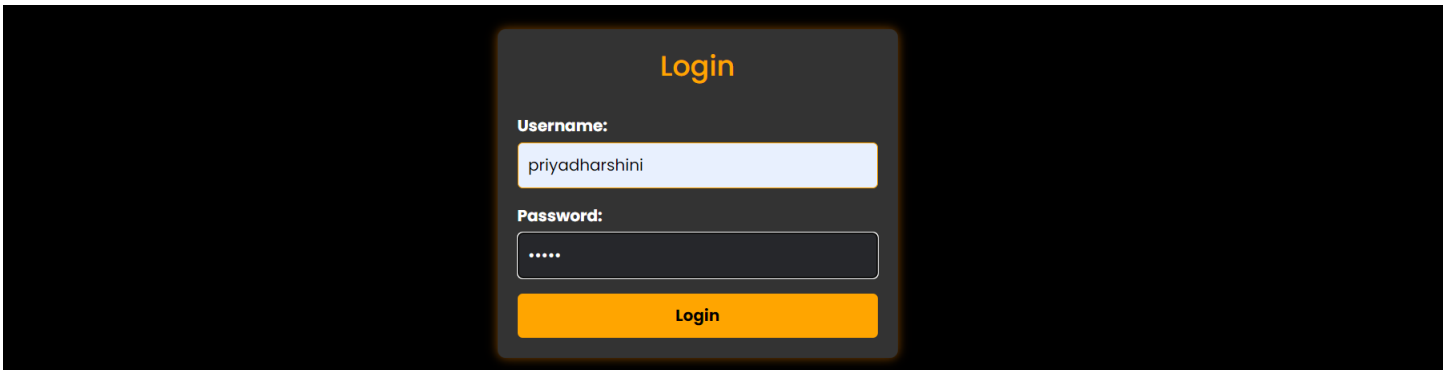
user_id	username	email	password
1e501b43-8ad3-41cb-8f5a-6454a471c64a	roshitha	roshi@gmail.com	$2b$10$CqH0gFtjkE9iabEXLRwWJOC7nrVnlFZnFdJ6nmNz8TgxykDXFYtsS
5849da06-7f68-47ed-8c56-e920fb7c9f07	vaishu	vaishu@gmaii.com	$2b$10$yiEp8q/.QVWaLfNiEIgqbeO0wIwX0/pqGCe2r311JUQKqzW6Redn6
914716ef-3e82-4ccd-8f94-2e9a53681317	priyadharshini	priyadharshini@iiitkottayam.ac.in	$2b$10$agqRTFg4DtVSPCSvnijkhe2F8fOv/5hLgyMleKSKg6qGJsrKhV10.
a31182a0-7e5d-49b4-bd88-9b5f1f7ccc7e	pavan	pavan@gmail.com	$2b$10$eLPv3x56tIJB5ium8XGF4.xhilRhCqukud5zzg3dSourDPNtBr3zO
aa824429-6090-4595-9e3e-9cb0e42876c1	nikkath	trabbitco@gmail.com	$2b$10$zKPEBv07KhPE4x9VIUDIiu6XdOC5aC3mvMO2w8VS6XoBfNkW60r12
c5d0ed0c-5b1b-42cb-801f-9607f4717bdf	qwerty	qwerty@gmail.com	$2b$10$Q6XuIqR.TPZx0xsQoxTPuuWxQLEPeWFyLS9ypljCkmE7.mhhGOimi

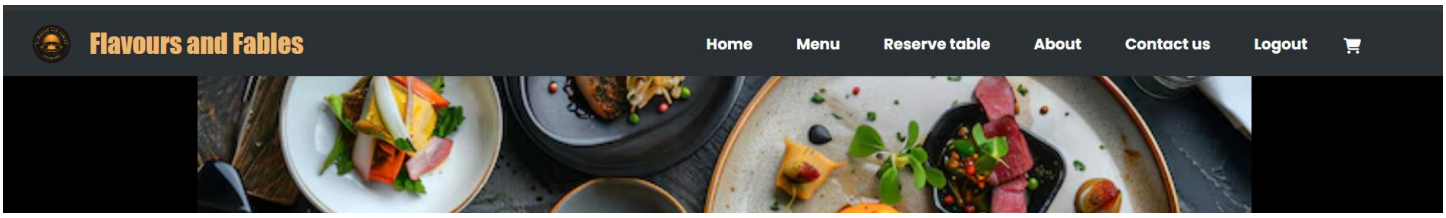
❖ User login query:

```
try {
        // Query the database for the user using connection.execut
        eonst [results] = await db.execute('SELECT * FROM users WHERE username =   , [username]);
                                                  ?'
        // Log the query result
        console.log('Query result   , results);
                    s:'
        // Check if user exist
        if (results.length === 0) {
            req.flash('error', 'Invalid username or passwor  );
            return res.redirect('/login/use  ); // Redirect back to login if username does not exis
        }                       r'              t

        // Compare the password using bcryp
        const user = results[0]; // The user object from the databas
        const passwordMatc  = awaiit bcrypt.compare(password, user.password); // Compare entered password with stored has
              h                                                                h
        if (!passwordMatc ) {
            rbq.flash('error', 'Invalid username or passwor  );
            return res.redirect('/login/use  ); // Redirect back to login if password does not matc
        }                       r'              h
```

Here when ever user tries to login first it will check a particular user has signed in or not, if user doesn't signed in and tries to login it shows pops up to signup or any username or password issues it shows invalid username or password.

Testing:



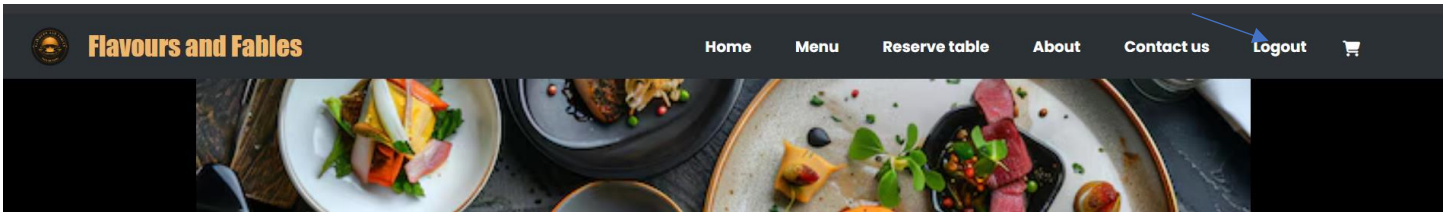


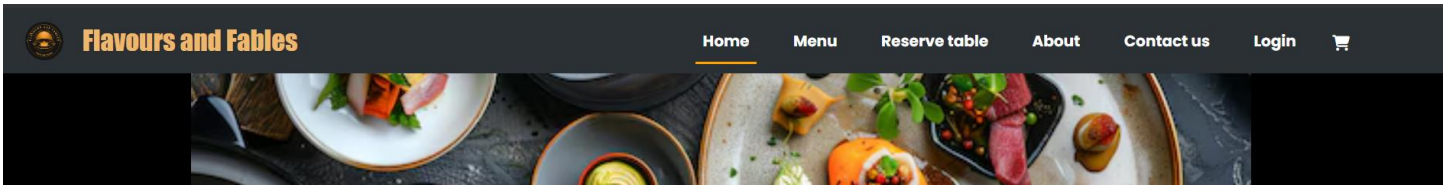
Here user is already sign up so it redirects to home page.

❖ User logout

```
req.session.destroy((err) => {
        if (err) {
            console.log(err);
            return res.redirect('/');
        }
        // Clear the cooki
        res.clearCookie('connect.si  ); // Assuming 'connect.sid' is the default session cookie nam
        // req.flash('sudcess', 'You have been logged ou
        res.redirect('/login'); // Redirect to login page after logou
    });                          t
```

Here when user clicks logout in nav-bar users session will be destroyed





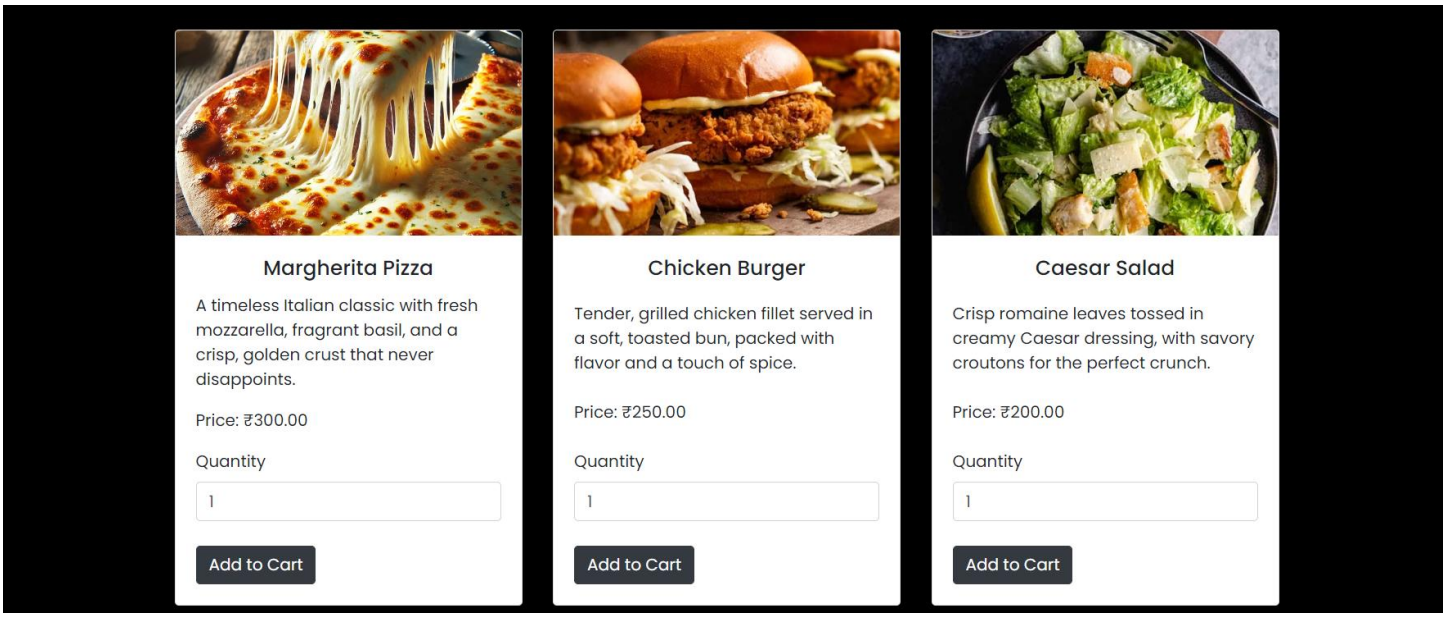
Here session of certain user ends and redirects home page.

❖ Get items query

```
try {
        const [items] = await connectio .execute('SELECT * FROM item  );
        res.render('listings/mnu.ej  , { items });
    }              s'
```

Here items from the items table will be fetched and displayed in the menu page.

Here we used cards to display the menu, in menu we can update the quantity and add to cart functionalities will be there.



There are many rows of items but this is a sample items list from the menu page.

❖ Add to cart query:

```
const [reservationRow ] = await connectio .execute(
        s   `SELECT * FROM reservations WHERE user_id = ? ,
            `userId]
[
);
```

First when user tries to add the item to cart, it checks whether the user has reserved the table.

```
const [itemRows] = await connectio .execute(
            `SELECT item_id, price FROM items WHERE item_id = ? ,
            `itemId]
[
);
```

When user adds the item to cart, fetches the item details of item added to cart and stores in item_id for further functionality.

```
const [orderResult] = await connectio .execute(
                    `INSERT INTO orders (user_id, total_amount) VALUES (?, ?) ,
                    `userId, 0] // Ensure userId is defined her
[                                e
);
```

When item is added to cart a order_id is created in orders table for that order_id the session user_id, total amount will be stored.

```
const [existingItemRow ] = await connectio .execute(
        s         `SELECT order_item_id, quantity FROM order_items
                   WHERE order_id = ? AND item_id = ? ,
`                  orderId, itemId]
[
);
```

After adding item to cart the order_items will be stored in the order_items tables along with their unique order_id fetched from session and quantity, amount of the certain item.

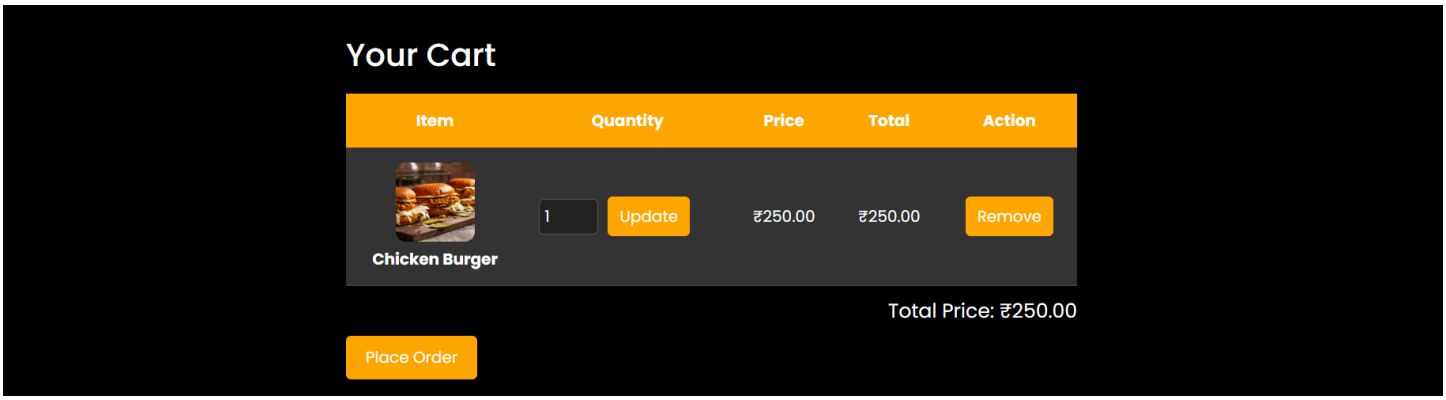
```
await connectio .execute(
      n             `UPDATE order_items
                     SET quantity = ?
                     WHERE order_item_id = ? ,
`                    newQuantity, existingItemRow [0].order_item_i ]
[                                s                    d
);
```

Whenever user tries to add the same item to the cart then the item in the cart will be updated by one.

```
await connectio .execute(
      n             `INSERT INTO order_items (order_id, item_id, quantity, pric
                    eVALUES (?, ?, ?, ?) ,
`                    orderId, itemId, quantity, item.price] // Ensure item.price is define
[                                                           d
);
```

And then all the items will be inserted into order_items table.

Testing:

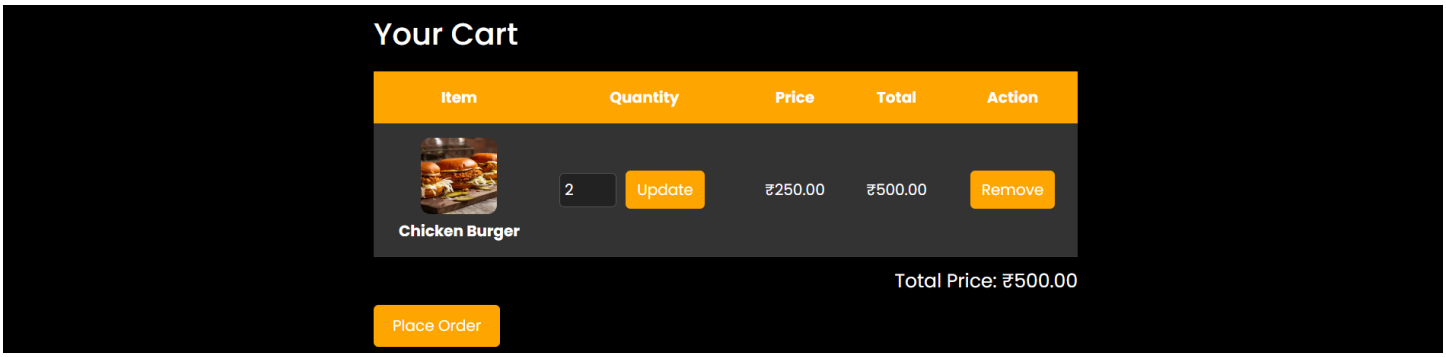


20	c5d0ed0c-5b1b-42cb-801f-9607f4717bd1	780.00	2024-10-21 23:37:29	0	6
21	1e501b43-8ad3-41cb-8f5a-6454a471c64a	800.00	2024-10-22 18:33:58	1	4
22	1e501b43-8ad3-41cb-8f5a-6454a471c64a	0.00	2024-10-22 21:46:23	1	NULL

Here after placing order only a employee will be assigned to particular order.

16	21	2	2	250.00
17	22	2	1	250.00

This is the order_items table whenever user adds to cart the item will be inserted to order_items table with the order_id, quantity.



Here I updated the quantity to 2

16	21	2	2	250.00
17	22	2	2	250.00

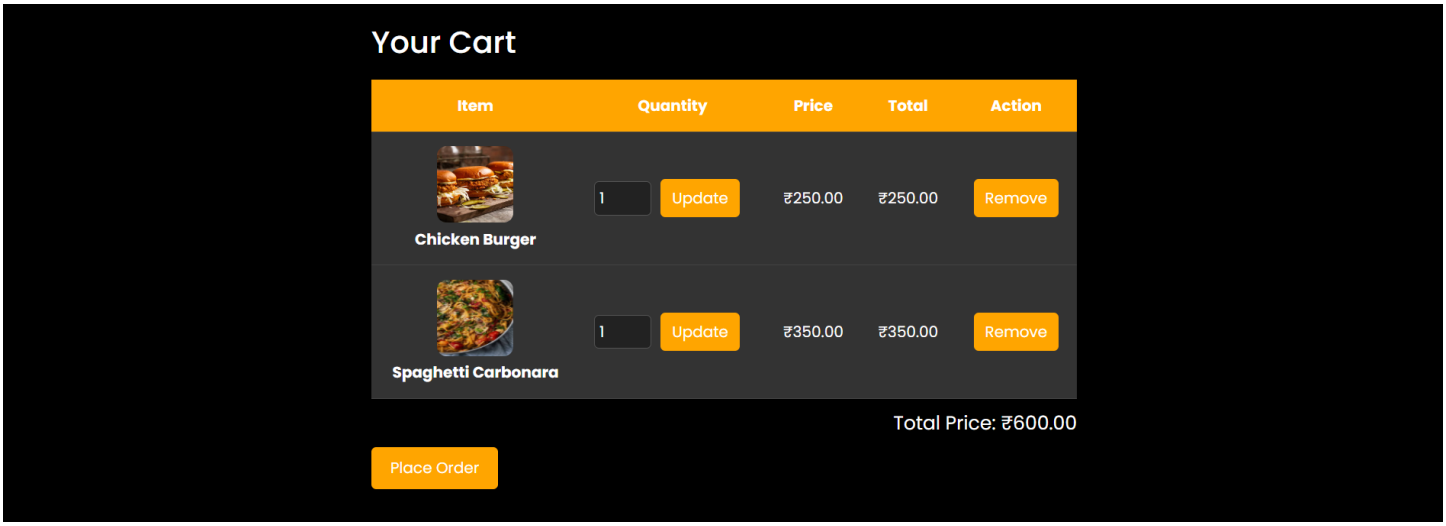
Here is the database change in order_items table too.

❖ View cart:

```
const [cartItems] = await connectio .execute(
            `SELECT oi.ordar_item_id, i.title, i.image_url, oi.quantity, oi.pric
            eFROM order_items o
i            JOIN items i ON oi.item_id = i.item_i
d            WHERE oi.order_id = ? ,
`            req.session.order_id]
[
);
```

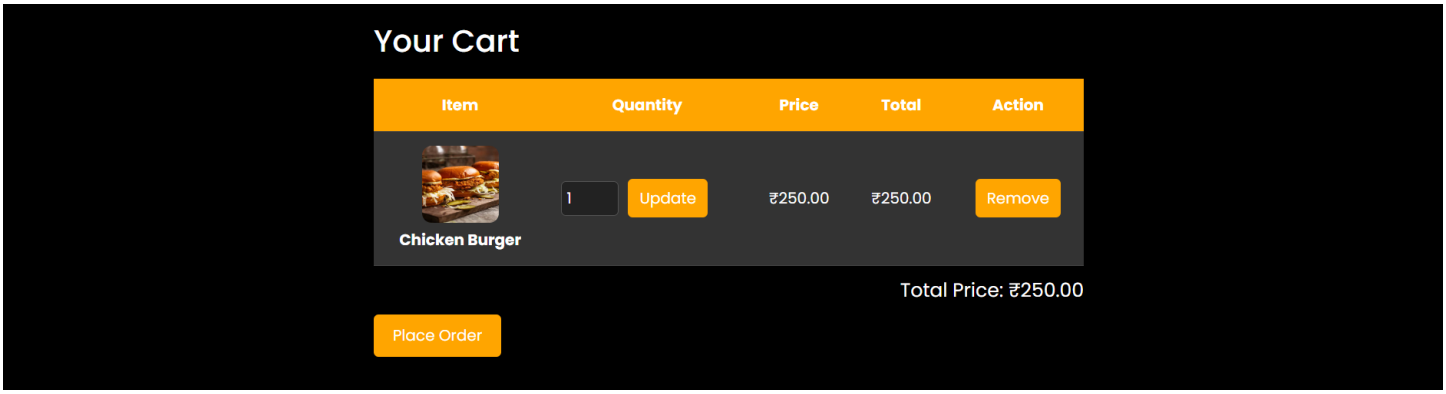
For viewing cart after adding the items.

Testing:

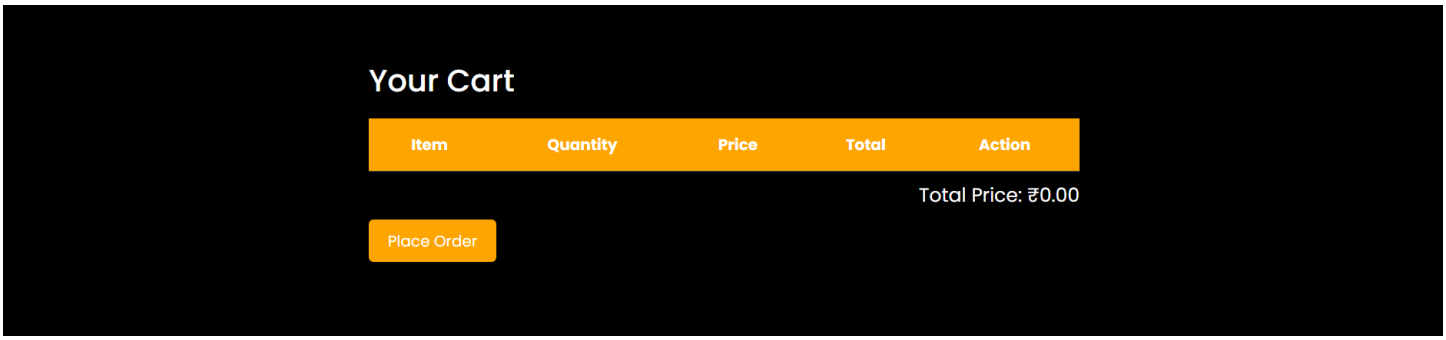


❖ Delete query

```
await connectio .execute(
      n     `DELETE FROM order_items WHERE order_item_id = ? ,
            `orderItemId]
[
);
```



16	21	2	2	250.00
18	22	2	1	250.00



15	21	1	1	300.00
16	21	2	2	250.00

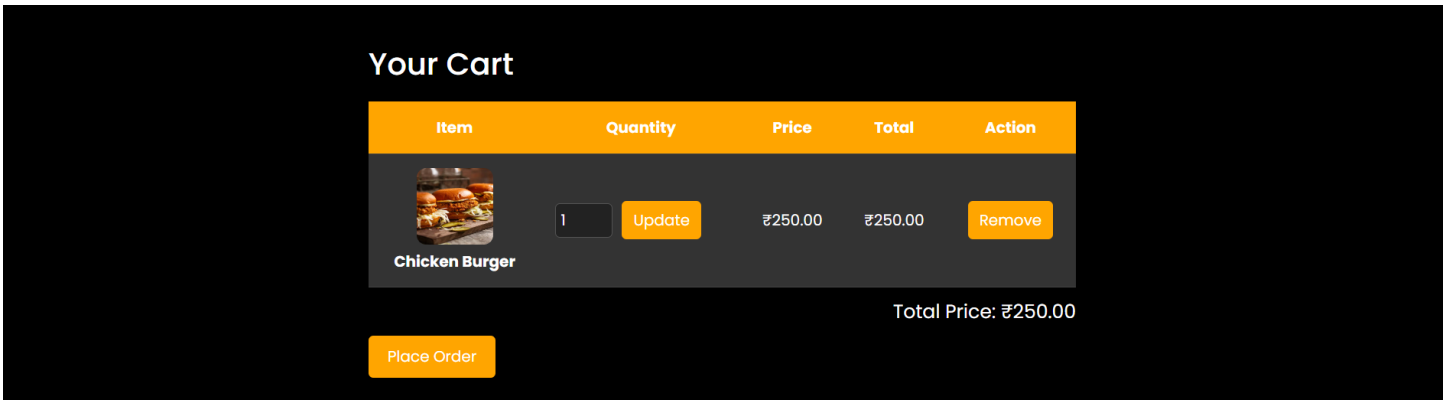
Whenever user removes the item from the cart the item is deleted from the order_items table.

❖ **Place order query:**

```
1  await connectio .execute(
2        n     `UPDATE orders SET total_amount = ?, employee_id = ? WHERE user_id = ? AND total_amount = 0 ,
3              `totalAmount, employeeI , req.session.user_id]
4  [                          d
5  );
6          // Fetch the cart items to display in the order confirmatio
7          const [cartItems] = await connectio .execute(
8              `SELECT i.title, oi.quantit
9              yFROM order_items o
10 i            JOIN items i ON oi.item_id = i.item_i
11 d            WHERE oi.order_id = ? ,
12 `            req.session.order_id]
13 [
   );
```

Here after placing order final updates will be done in orders table and order_items table and here a random emp_id will be given to the order so that a certain employee will takes the order.

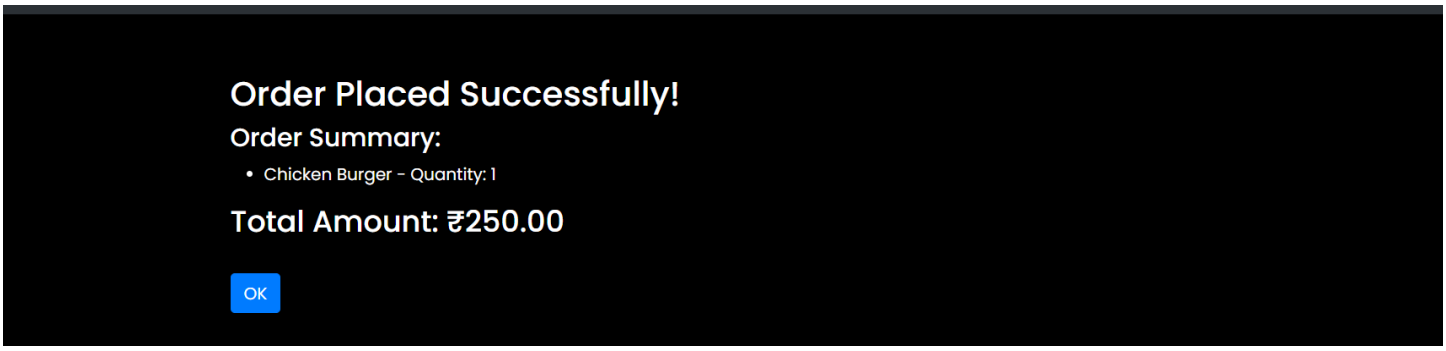
And the total amount will be updated in orders table.



Before placing the ordwe

20	c5d0ed0c-5b1b-42cb-801f-9607f4717bdf	780.00	2024-10-21 23:37:29	0	6
21	1e501b43-8ad3-41cb-8f5a-6454a471c64a	800.00	2024-10-22 18:33:58	1	4
22	1e501b43-8ad3-41cb-8f5a-6454a471c64a	0.00	2024-10-22 21:46:23	1	NULL

This the order confirmation page.



After placing the order, a emp_id is assigned

20	c5d0ed0c-5b1b-42cb-801f-9607f4717bdf	780.00	2024-10-21 23:37:29	0	6
21	1e501b43-8ad3-41cb-8f5a-6454a471c64a	800.00	2024-10-22 18:33:58	1	4
22	1e501b43-8ad3-41cb-8f5a-6454a471c64a	250.00	2024-10-22 21:46:23	1	7

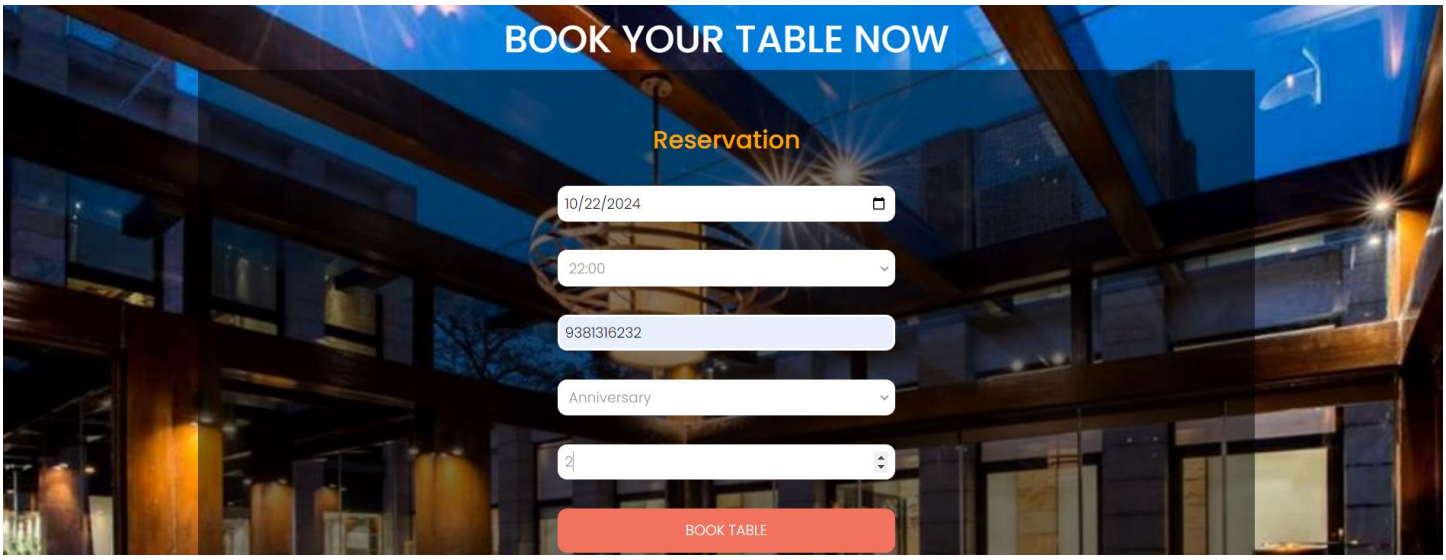
❖ Table reservation query

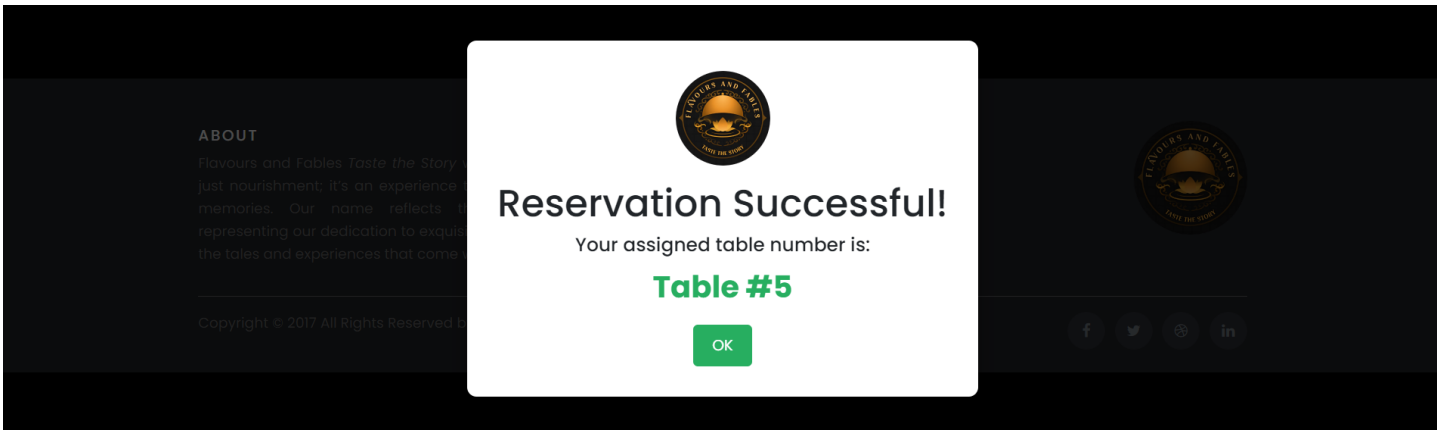
```
const [rows] = await connectio .execute(
            `SELECT table_number FROM reservations WHERE reservation_date = ? AND reservation_hour = ? ,
            `reservation_dat , hours]
[            e
);
        // Get an array of available table numbers from 1 to 1
        const bookedTable  = rows.map(row => row.table_numbe );
        const availableTable  = Array.from({ length: 10 }, (_, i) => i + 1).filter(table => !bookedTable .includes(table));
              s                                                                                                  s
        if (availableTable .length === 0) {
            return res.status(400).send('No tables available for the selected date and tim   );
        }                                e.'

        // Randomly pick an available table numbe
        const randomTableInde  = Math.floor(Math.random() * availableTable .length);
        const tableNumber = availableTable [randomTableInde ];
                            s                x
        // Insert the reservation with the randomly assigned table numbe
        const [result] = await connectio .execute(
            `INSERT INTO reservations (user_id, reservation_date, reservation_hour, phone_number, occasion, num_persons, table_numbe
            VALUES (?, ?, ?, ?, ?, ?, ?) ,
`            req.session.user_id, reservation_dat , hours, phone_numbe , occasions, num_persons, tableNumber]
[                                e                        r
);
        // Render the success page and pass the table number to the templat
        res.render('listings/reservationsucces  , { tableNumber });
                    s'
```

Here user can reserve a table after login

User fills the required details here query checks whether the table is free for the selected day and time slot if the table is free he will be ssigned a table else displays all tables are full for selected date and time slot.

Testing:





When reservation is done a popup will be displayed with the table_no

id	user_id	reservation_date	reservation_hour	phone_number	occasion	num_persons	table_number
9	1e501b43-8ad3-41cb-8f5a-6454a471c64a	2024-10-22	16:00:00	9381316232	dinner	2	10
10	1e501b43-8ad3-41cb-8f5a-6454a471c64a	2024-10-23	12:00:00	9381316232	birthday	3	2
11	1e501b43-8ad3-41cb-8f5a-6454a471c64a	2024-10-22	22:00:00	9381316232	anniversary	2	5

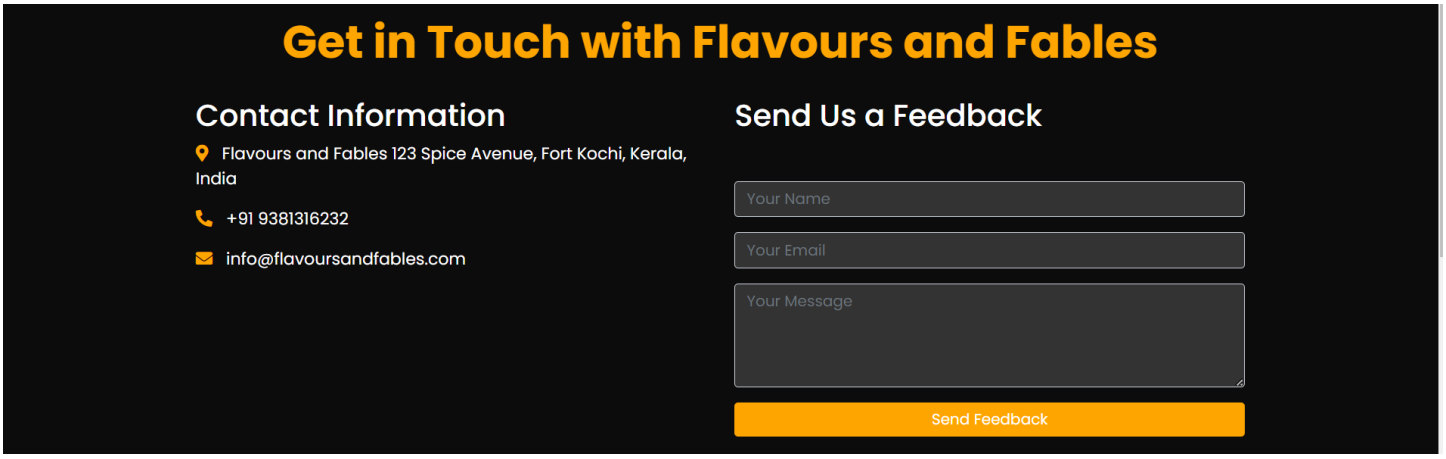
❖ Feedback query:

```
const query = 'INSERT INTO feedback (name, email, message) VALUES (?, ?,    ;
        await connectio .execute(query, [name, email, message]);
              n
        req.flash("success", "Feedback submitted successfull   );
        res.redirect('/'); //you redirect to the home page or contact pag
    }                      e
```

All the feedback details will be stored in feedback form

Testing:

id	name	email	message	created_at
1	Siraparapu Sai Satya Pavan Charan	sspavancharan@gmail.com	my table number gone	2024-10-21 23:37:01



From the contact us feedback will be written and viewed by the employee.

❖ Employee login query

```
const [results] = await db.execute('SELECT * FROM employees WHERE emp_email =   , [email]);
                                   ?'
        if (results.length === 0) {
            // If no matching employee is foun
            req.flash('error', 'Invalid email or passwor   );
            return res.redirect('/login/employeelogi  );
        }                          n'
```

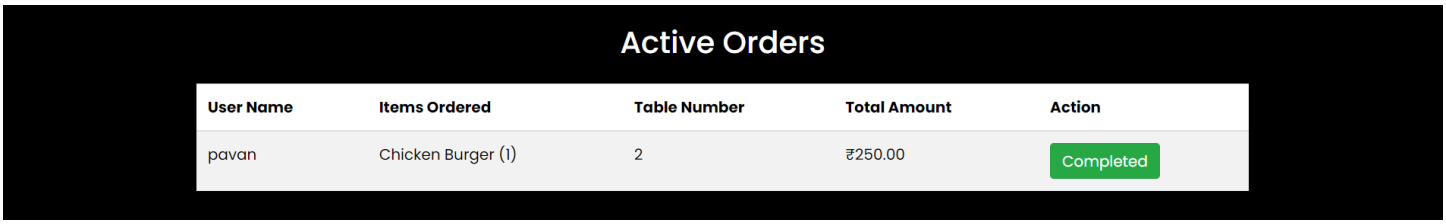
We have given a sample database for employee email and password by that credentials only a certain employee can login

❖ Activeorders query:

```
const [activeOrder ] = await db.execute(`
        s    SELECT o.order_id, u.username AS user_name,
                    GROUP_CONCAT(CONCAT(i.title, ' (', oi.quantity, ')') SEPARATOR ', ') AS items_ordere
d,                  ANY_VALUE(r.table_number) AS table_number,
                    ANY_VALUE(o.total_amount) AS total_amoun
t           FROM orders
o           JOIN users u ON o.user_id = u.user_i
d           JOIN order_items oi ON o.order_id = oi.order_i
d           JOIN items i ON oi.item_id = i.item_i
d           JOIN reservations r ON o.user_id = r.user_i
d           WHERE o.is_active = TRU
E             AND o.employee_id = ?  -- Filter by employee I
D           GROUP BY o.order_i
d        `, [employeeI ]);
            d
```

Here active orders will be fetched from orders table and displayed to particular employee of assigned orders

Testing:



❖ Completed order query:

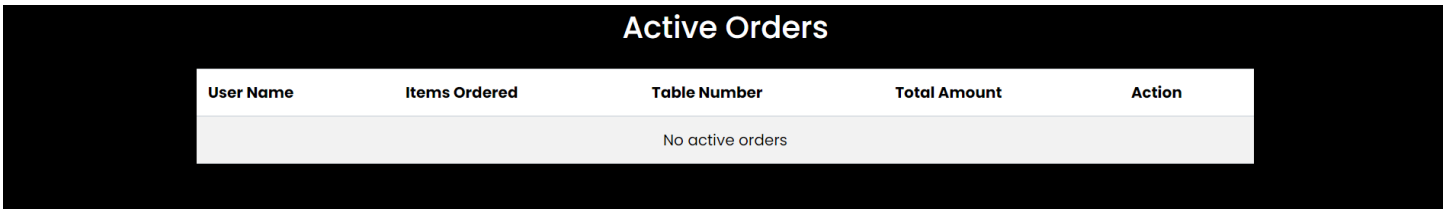
```
const [rows] = await connectio .execute(`
            SELECT r.table_number
            FROM reservations
r           JOIN orders o ON r.user_id = o.user_i
d           WHERE o.order_id =
?       `, [orderId]);

        const tableNumber = rows.length > 0 ? rows[0].table_numbe  : null;
                                                        r
        if (tableNumber) {
            // Remove the specific table number from the reservations tabl
            await connectio .execute('DELETE FROM reservations WHERE table_number =   , [tableNumber]);
        }         n                  ?'

        // Mark the order as inactiv
        await connectio .execute('UPDATE orders SET is_active = FALSE WHERE order_id =   , [orderId]);
```

```
const [activeOrder ] = await db.execute(`
        s    SELECT o.order_id, u.username AS user_name,
                    GROUP_CONCAT(CONCAT(i.title, ' (', oi.quantity, ')') SEPARATOR ', ') AS items_ordere
d,                  ANY_VALUE(r.table_number) AS table_number,
                    ANY_VALUE(o.total_amount) AS total_amoun
t           FROM orders
o           JOIN users u ON o.user_id = u.user_i
d           JOIN order_items oi ON o.order_id = oi.order_i
d           JOIN items i ON oi.item_id = i.item_i
d           JOIN reservations r ON o.user_id = r.user_i
d           WHERE o.is_active = TRU
E           GROUP BY o.order_i
d       `);
```

Here after clicking complete order the user_id related to that order will be fetched and deletes the reservations table of that certain user_id session.

id	user_id	reservation_date	reservation_hour	phone_number	occasion	num_persons	table_number
13	a31182a0-7e5d-49b4-bd88-9b5f1f7ccc7e	2024-10-23	18:00:00	9381316232	anniversary	2	2

Before clicking complete orders




```
mysql> select * from reservations;
Empty set (0.00 sec)

mysql> |
```

After clicking complete orders.

Conclusion:

Finaly this is our website where streamline all the tasks of employee , manage reservations, manage orders, and user, employee authentication.

Thank you