Database Management Systems
Food Ordering System

Project Report

Khizar Asad – 23988

Humayun Raza - 24464

Business Scenario:

Our business scenario is a food delivery web application, which allows listings of multiple restaurants and allows customers to create accounts and place orders from any restaurant of their choice from the listed restaurants.

Business Rules:

Users:

1. **Registration & Login:**
    - **Users must register with valid personal details.**
    - **Registered users can log in using their credentials.**
2. **Ordering:**
    - **Users can browse the restaurant menu.**
    - **Users can add items to their cart.**
    - **Users can place an order from their cart.**
    - **Users can view the status of their orders.**
3. **Profile & History:**
    - **Users can update their personal details.**
    - **Users can view their order history.**

Restaurants:

1. **Registration & Login (If restaurants have their own dashboard/account):**
    - **Restaurants must be added by the admin.**
    - **Restaurants can log in using credentials provided by the admin.**
2. **Orders:**
    - **Restaurants can view orders placed for their establishment.**
    - **Restaurants can accept or reject an order.**
    - **Restaurants can update the status of an order (e.g., "preparing," "out for delivery," "delivered").**
3. **Menu Management:**
    - **Restaurants can add, update, or remove items from their menu.**
    - **Restaurants can update item prices.**

Admin:

1. **Access:**
    - **Admin has exclusive login credentials.**
    - **Admin has full access to the system and its data.**
2. **User Management:**
    - **Admin can view all user details.**
    - **Admin can delete users.**
3. **Restaurant Management:**
    - **Admin can add new restaurants.**
    - **Admin can update restaurant details.**
    - **Admin can delete restaurants.**
    - **Admin can add Riders.**
4. **Order Oversight:**
    - **Admin can view all orders placed on the platform.**
    - **Admin can intervene in any order if necessary (though this should be limited to ensure restaurant autonomy).**

5. **Reports & Analytics (Optional but good for future scalability):**
    - **Admin can generate and view reports on user activity, top restaurants, order trends, etc.**

Entities:

USERS:

- **Represents the registered users on the food ordering platform.**

Attributes:

- **UserID: Unique identifier for each user. Auto-incremented.**
- **Role: Specifies the role of the user (e.g., user, admin). Default is 'user'.**
- **fullName: Full name of the user.**
- **email: Unique email address of the user.**
- **password: Encrypted password for the user.**
- **phone_number: Phone number of the user.**
- **address: Address where the user wishes to get the food delivered.**

RESTAURANTS:

- **Represents the restaurants on the platform.**

Attributes:

- **RestaurantID: Unique identifier for each restaurant. Auto-incremented.**
- **email: Unique email address for the restaurant.**
- **password: Encrypted password for the restaurant.**
- **RestaurantName: Name of the restaurant.**
- **address: Address of the restaurant.**
- **phone_number: Phone number of the restaurant.**
- **website: Website link for the restaurant.**

RESTAURANTITEMS:

- **Represents the menu items offered by restaurants.**

Attributes:

- **PRODUCTID: Unique identifier for each product/item. Auto-incremented.**
- **RestaurantID: Identifier linking the product to its respective restaurant.**
- **Name: Name of the product/item.**
- **Description: Description of the product/item.**
- **Category: Category of the product/item (e.g., starters, main course, etc.).**
- **Price: Price of the product/item.**

ORDERS:

- **Represents the orders placed by users.**

Attributes:

- **OrderID: Unique identifier for each order. Auto-incremented.**
- **UserID: Identifier linking the order to the user who placed it.**
- **RestaurantID: Identifier linking the order to the restaurant.**
- **OrderTimeDate: Timestamp when the order was placed.**
- **OrderStatus: Status of the order (e.g., pending, processing, delivered).**
- **GRANDTOTAL: Total amount for the order.**

ORDER_DETAILS:

- **Represents the detailed breakdown of an order in terms of the items/products.**

Attributes:

- **OrderID: Identifier linking to the main order.**
- **ProductID: Identifier linking to the product/item in the order.**
- **Quantity: Number of units of the product/item in the order.**
- **Subtotal: Total cost of the product/item considering its quantity.**

RIDERS:

- **Represents riders available in the system so orders can be assigned to the Rider.**

Attributes:

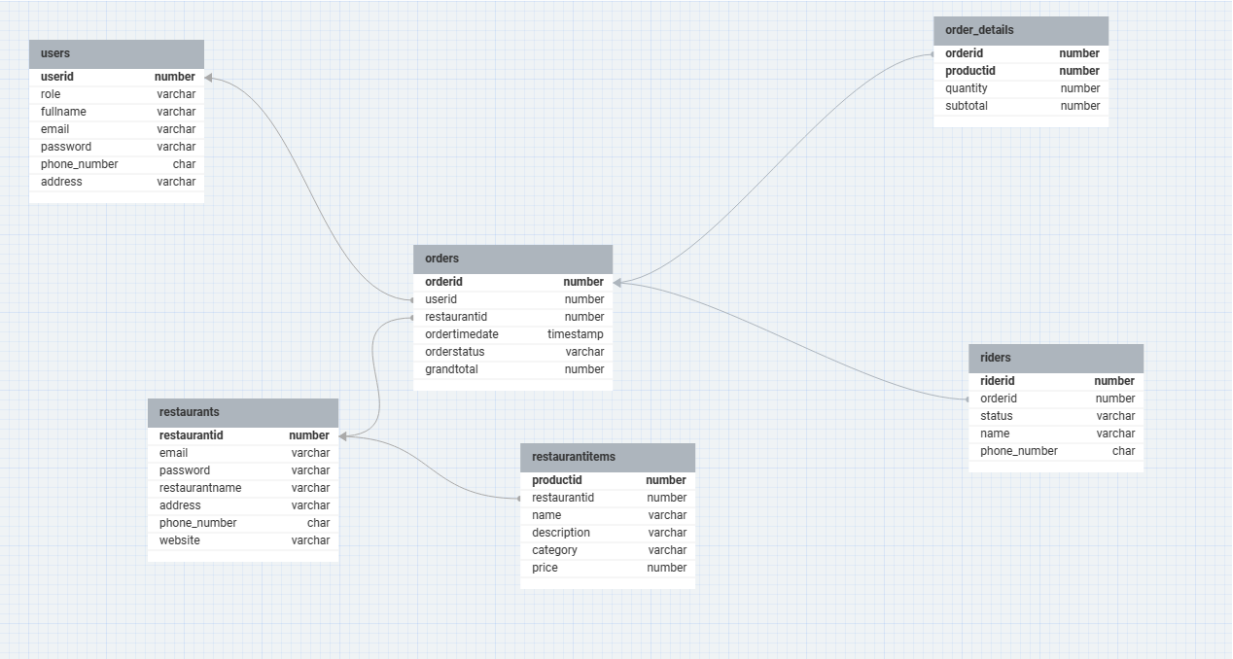
- **RiderID: The Riders unique Identifier.**

- **OrderID: Current Order assigned to rider; can be NULL.**
- **Status: Shows current status of rider e.g if the rider is available to take an order.**
- **Name: Rider name.**
- **Phone Number: Rider contact number.**

Entities:

1. **RESTAURANTITEMS to RESTAURANTS:**
    - **Type: Many-to-One**
    - **Multiplicity: A product/item (RESTAURANTITEMS) belongs to one restaurant (RESTAURANTS), but a restaurant can have many products/items.**
2. **ORDERS to USERS:**
    - **Type: Many-to-One**
    - **Multiplicity: An order (ORDERS) is placed by one user (USERS), but a user can place many orders.**
3. **ORDERS to RESTAURANTS:**
    - **Type: Many-to-One**
    - **Multiplicity: An order (ORDERS) is for one restaurant (RESTAURANTS), but a restaurant can receive many orders.**
4. **ORDER_DETAILS to ORDERS:**
    - **Type: Many-to-One**
    - **Multiplicity: A specific product/item detail (ORDER_DETAILS) belongs to one order (ORDERS), but an order can have details for many products/items.**
5. **ORDER_DETAILS to RESTAURANTITEMS:**
    - **Type: Many-to-One**
    - **Multiplicity: A specific product/item detail (ORDER_DETAILS) corresponds to one product/item (RESTAURANTITEMS), but a product/item can be in the details of many orders.**
6. **RIDERS to ORDERS:**
    - **Type: One-to-One**
    - **Multiplicity: A specific rider can only have one order assigned to him at a time**

ER Diagram:



DDL SNIPPETS:

DDL Script

Constraints Applied to Tables:

1. USERS table:
   - UserID: Primary Key
   - Role: Default value 'user'
   - Email: Unique
2. RESTAURANTS table:
   - RestaurantID: Primary Key
   - Email: Unique
3. RESTAURANTITEMS table:
   - PRODUCTID: Primary Key
   - RestaurantID: Foreign Key referencing RESTAURANTS
4. ORDERS table:
   - OrderID: Primary Key
   - UserID and RestaurantID: Foreign Keys
5. ORDER_DETAILS table:
   - Composite Primary Key (OrderID, ProductID)
   - Foreign Keys referencing ORDERS and RESTAURANTITEMS
6. RIDERS table:
   - RiderID: Primary Key
   - ORDERID: Foreign Key referencing ORDERS

Triggers, Stored Procedures, and Views:

1. Triggers:
   - users_bir: Before insert trigger on USERS table for auto-incrementing UserID.
   - restaurants_bir: Similar trigger for RESTAURANTS table.
   - restaurant_items_bir: Trigger for RESTAURANTITEMS table.
   - orders_bir: Trigger for ORDERS table.
   - riders_bir: Trigger for RIDERS table.
2. Stored Procedures:
   - UPDATEORDERANDRIDERSTATUS: Procedure to update order and rider statuses.

Technical Aspects:

We have used ReactJS as our front-end framework for this project. For the back-end we have used NodeJS with ExpressJS. As our database we have used Oracle SQL.

Work Contribution:

Database Schema Planning was done by both group members, and both worked equally to make our database schema which aligned with our plans for the project.

Khizar worked on the Back-end API's and architecture of the project.

Humayun worked on the front-end of the project.

Wireframes

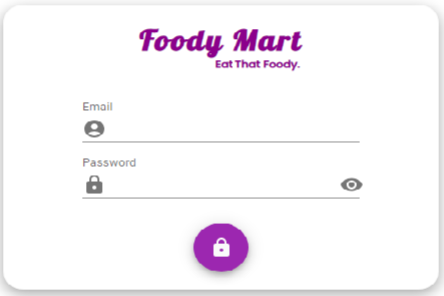


Figure 1 -Login Page

Figure 2 -Register Page

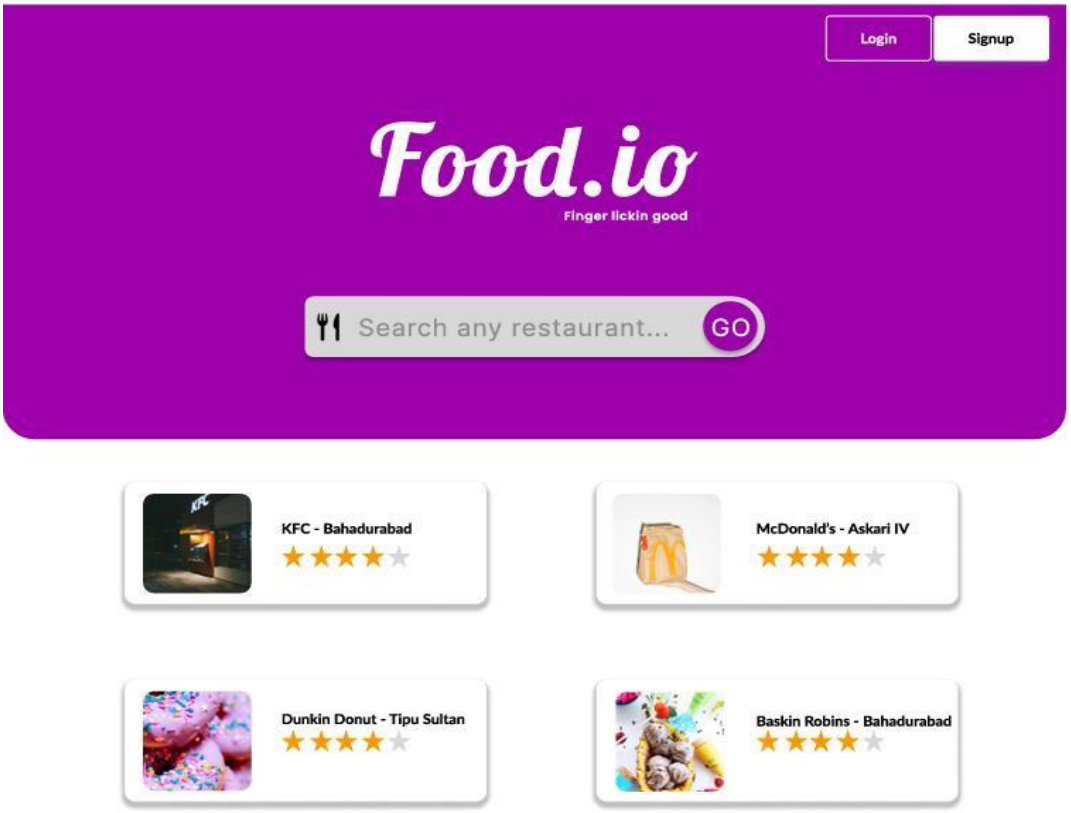


Figure 3- Home Page

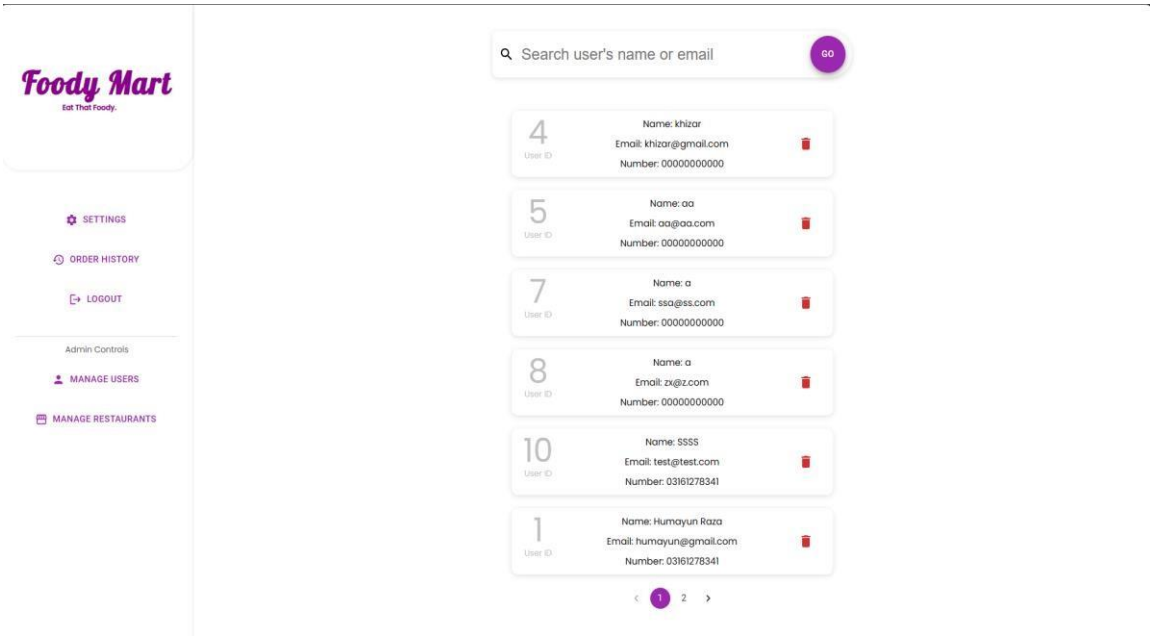


Figure 4 - User/Admin Dashboard