**LLD FOR FOOD DELIVERY APPLICATION**

**1.Introduction:**

Fun-food is a online platform that helps users to discover, explore, and order food from a wide range of restaurants, cafes, and eateries. It aims to provide the user-friendly interface for ordering and selling the food items. Customer can order food in a convenient way food from a variety of restaurants without the need to physically visit them. Merchant offers diverse culinary delights, ensuring quality and timely delivery for customer satisfaction.

* 1. **Purpose:**

The purpose of a Low-Level Design (LLD) document is to provide detailed technical specifications for implementing software modules or components. It is a component level process that follows a step-by-step refinement process. It outlines the interfaces, and dependencies required for development. It facilitates communication among team members, ensuring a shared understanding of the system's design and functionality.

* 1. **Scope:**

Low-Level Design describes the class diagrams with the methods and relations between classes and program specs. It describes the modules so that the programmer directly code the program from the document.

**2.Low Level Design:**

* LLD provides detailed technical specifications for individual components or modules within a software system.
* It translates the High-Level Design (HLD) into smaller, manageable units that can be implemented by developers.
* It specifies how different modules interact with each other and defines interfaces, inputs, and outputs for seamless communication.
* LLD serves as a blueprint or roadmap for developers, guiding them through the implementation phase with clear instructions and design decisions.

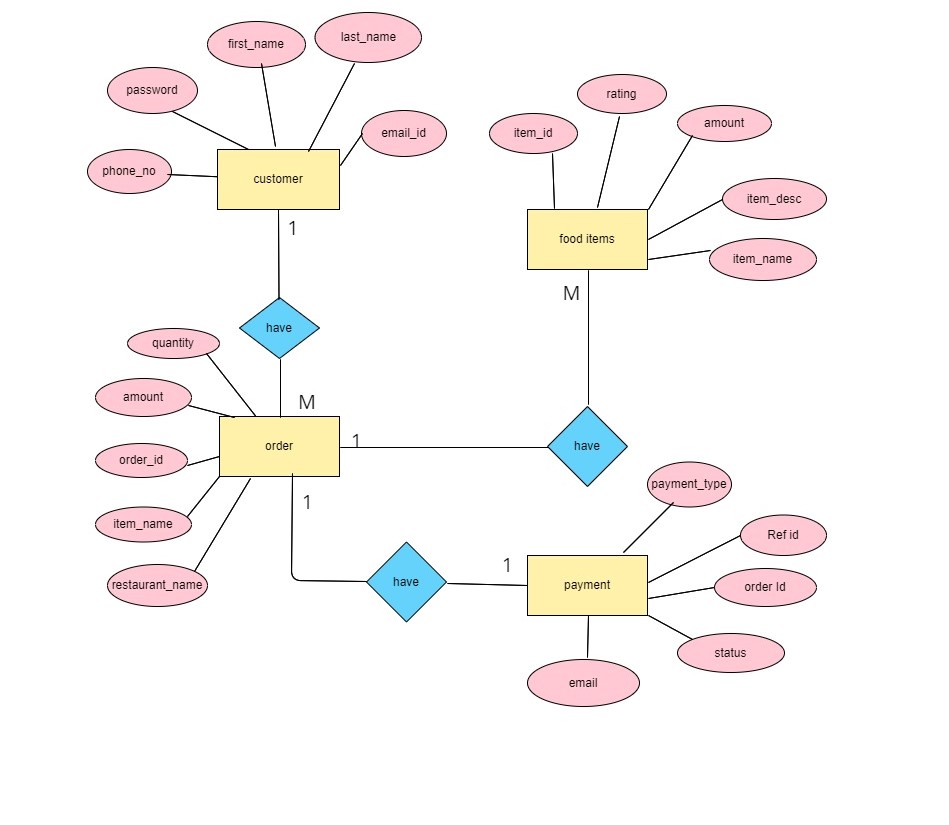
**2.1 Entity Relationship Diagram:**

Entity-Relationship (ER) modelling is a method used to visualize and describe the logical structure of a database.

**ERD for Customer model:**

It explains the structure of the application's classes and their relationships. It includes classes and essential entities such as users, items, orders, cart and payment. Based on these entities and attributes, we can now define the relationships between them:

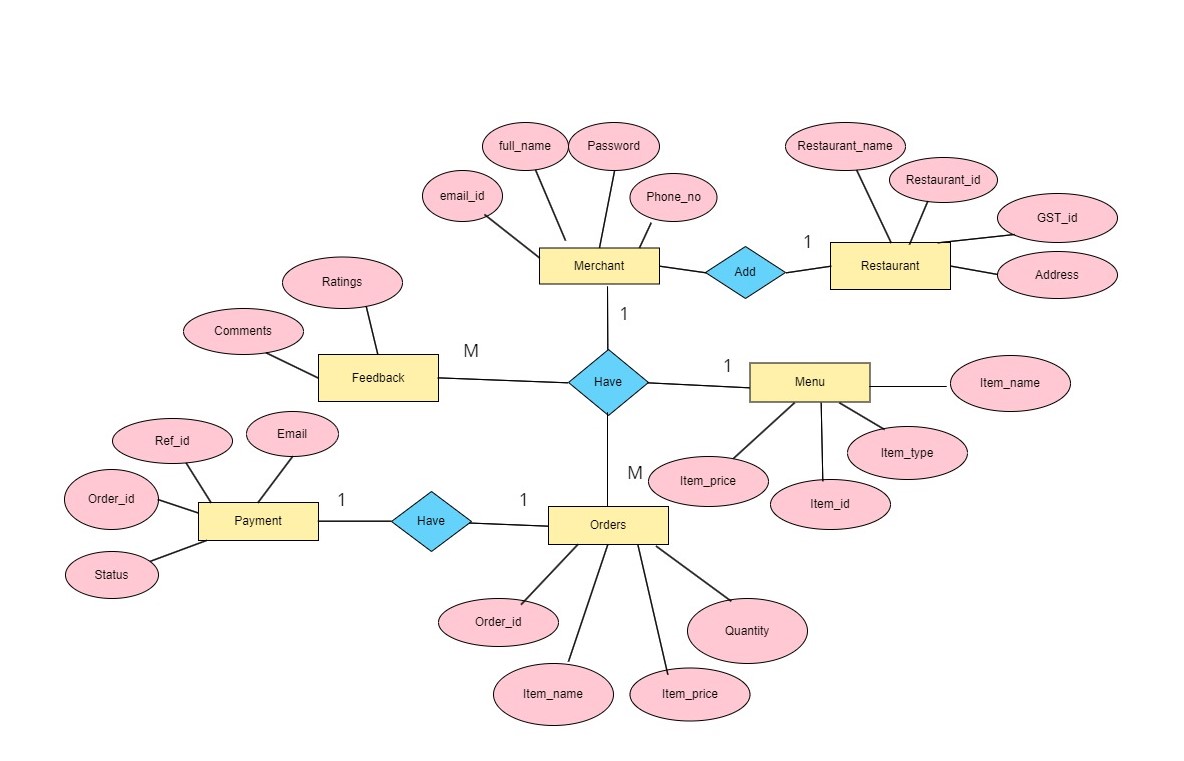
* User - Order: One-to-Many (One user can have multiple orders)
* Item- Order: Many-to-Many (Many items can be part of many orders)
* User - Payment: One-to-Many (One user can have multiple payments)
* Order - Payment: One-to-One (Each order has one payment)
* Cart- Item : Many-to-Many (Many items can be added to cart)
* User – Cart: One-to-Many (One User can add many items to cart)



**Fig : Customer model**

**ERD for Merchant model:**

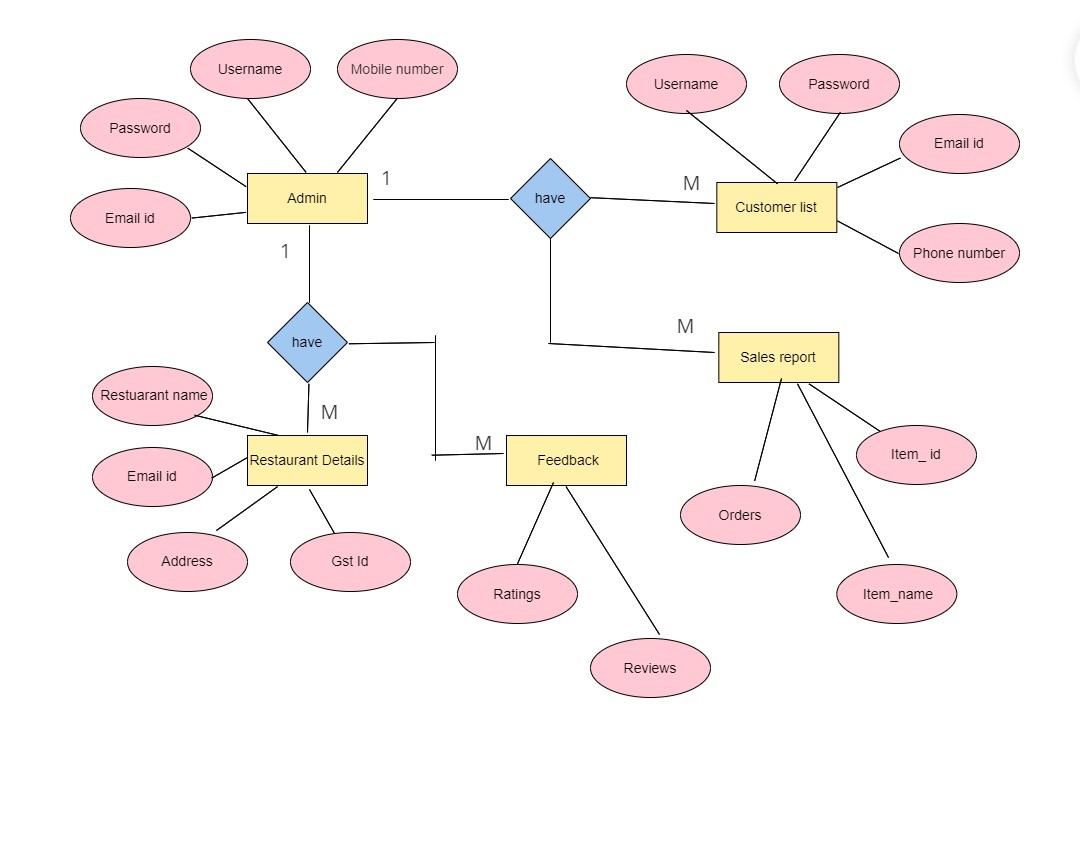
* The Merchant model contains entities such as merchant, item and order. Relationships defined between these entities are given as follows:
* Merchant – Item: One-to-Many (One merchant can have many items to sell)
* Item – Order: Many-to-Many (Items can be included in orders, and each order can contain multiple items)
* Merchant – Order: One-to-Many (Merchant can receive multiple orders)



**Fig : Merchant model**

**ERD for Admin model:**

Admin can have the privilege to manage merchants, users and food items. Merchant can add multiple food items. Users can place order food items.



**Fig : Admin model**

**3.2 Database Schema:**

**3.2.1 For User Model:**

1.USER DATABASE:

* userId INT PRIMARY KEY AUTO\_INCREMENT,
* username VARCHAR (50) NOT NULL,
* email VARCHAR (100) NOT NULL,
* password VARCHAR (100) NOT NULL,
* dob DATE

2.ITEM DATABASE:

* itemId INT PRIMARY KEY AUTO\_INCREMENT,
* item name VARCHAR(100) NOT NULL,
* amount DECIMAL(10, 2) NOT NULL,
* quantity INT NOT NULL,

3.ORDER DATABASE:

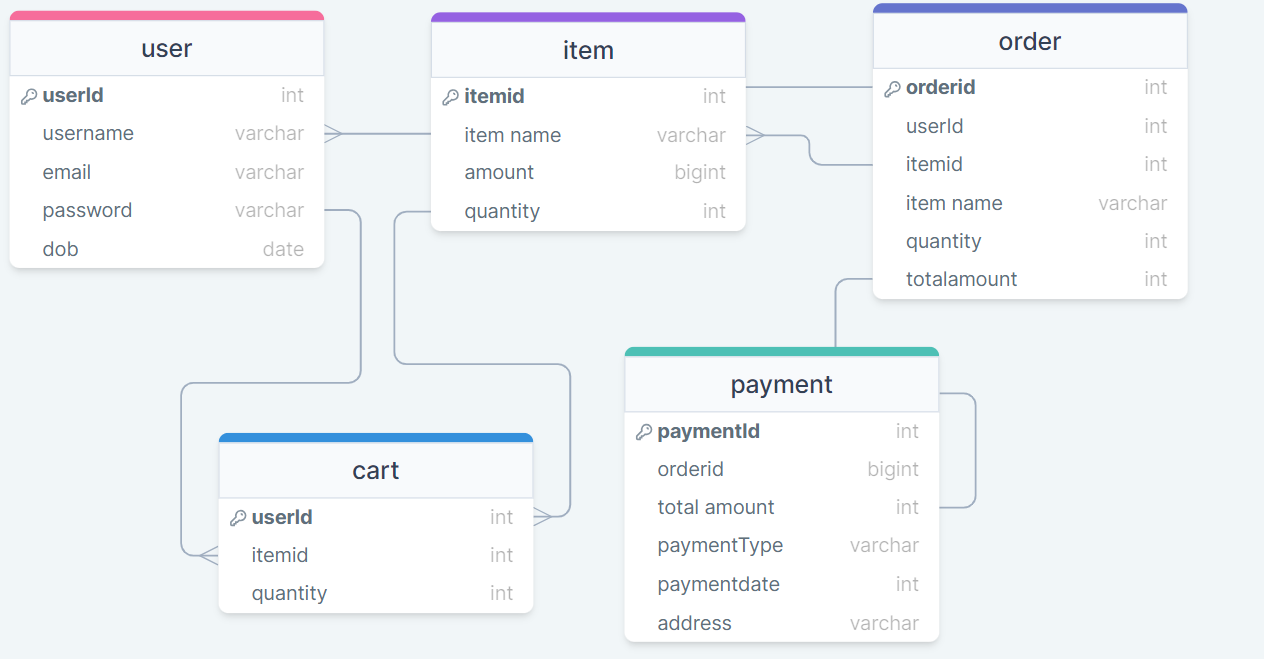
* + orderId INT PRIMARY KEY AUTO\_INCREMENT,
  + userId INT NOT NULL,
  + itemId INT NOT NULL,
  + quantity DECIMAL (10, 2) NOT NULL,
  + total Amount DECIMAL (10, 2) NOT NULL,
  + FOREIGN KEY (userId) REFERENCES User(userId)

4.CART DATABSE:

* + userId INT,
  + itemId INT,
  + quantity INT NOT NULL,
  + PRIMARY KEY (userId, itemId),
  + FOREIGN KEY (userId) REFERENCES User(userId),
  + FOREIGN KEY (itemId) REFERENCES item(itemId)

5. PAYMENT DTABASE:

* + paymentID INT PRIMARY KEY AUTO\_INCREMENT,
  + orderID INT NOT NULL,
  + total amount DECIMAL(10, 2) NOT NULL,
  + paymentType VARCHAR(50) NOT NULL,
  + paymentDate TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,
  + address VARCHAR (255)
  + FOREIGN KEY (orderID) REFERENCES `Order`(orderID)



**3.3.2 For Merchant model:**

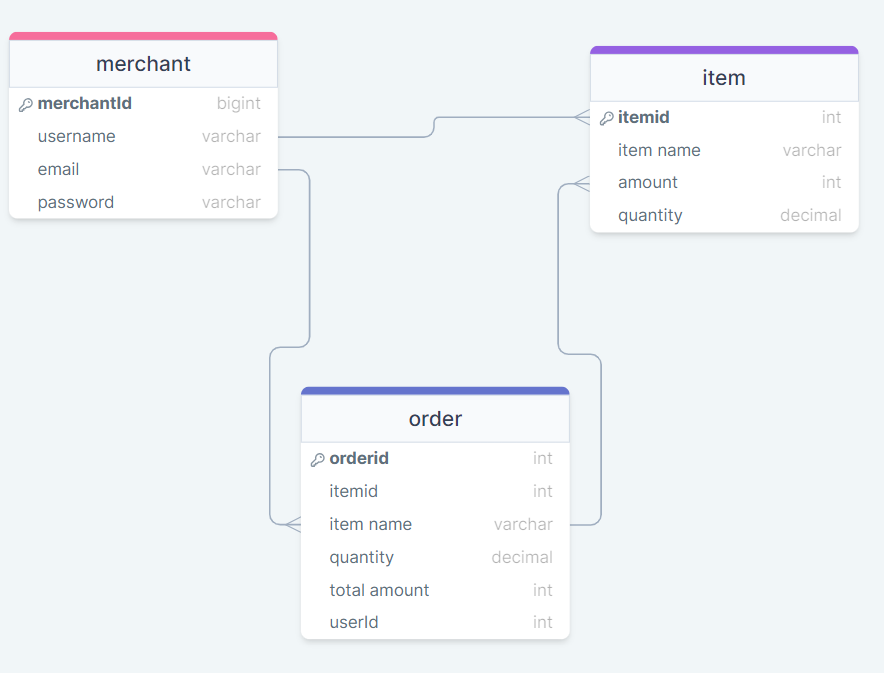
1.MERCHANT DATABASE:

* + merchantId INT PRIMARY KEY AUTO\_INCREMENT,
  + username VARCHAR(255) NOT NULL,
  + email VARCHAR(255) NOT NULL,
  + password VARCHAR(255) NOT NULL,

2.ITEM DATABASE:

* + itemId INT PRIMARY KEY AUTO\_INCREMENT,
  + item name VARCHAR(255) NOT NULL,
  + amount DECIMAL (10, 2) NOT NULL,
  + quantity INT NOT NULL,
  + FOREIGN KEY (merchantId) REFERENCES Merchant(merchantId)

3.ORDER DATABASE:

* + orderId INT PRIMARY KEY AUTO\_INCREMENT,
  + itemtId INT,
  + item name VARCHAR(255)
  + quantity INT NOT NULL,
  + total Amount DECIMAL (10, 2) NOT NULL,
  + userId IN

**4.Component Design:**

Frontend UI Components:

* User Interface: Developed using HTML, CSS, and JavaScript framework like Angular.
* Components: Modular components for user authentication, food item, cart, payment and checkout process.
* UI Design: Intuitive and user-friendly design with responsive layouts for various devices.

Backend API Components:

* Routing Layer: Handles incoming HTTP requests and routes them to the appropriate controllers.
* Controllers: Contains business logic for user authentication, product management, order processing, and payment integration.
* Middleware: Implements middleware functions for request parsing, authentication, error handling, and logging.
* Data Access Layer: Interacts with the database to perform CRUD operations on user data, item data, and order data.

**5. Security Consideration:**

* + Encryption: Utilizes encryption algorithms to secure sensitive data such as user passwords and payment information.
  + Authorization: Enforces role-based access control to restrict user access to authorized functionalities.
  + Input Validation: Validates user input to prevent injection attacks and other security vulnerabilities.
  + HTTPS: Ensures secure communication between the client and server using HTTPS protocol.

**6. Performance Optimization:**

* + Asynchronous Processing: Utilizes asynchronous processing techniques to offload time-consuming tasks and improve system responsiveness.
  + Database Optimization: Optimizes database queries and indexing strategies to minimize query execution time and improve overall database performance.

**7. Deployment Architecture:**

* + Cloud Deployment: Hosts the application on cloud platforms such as AWS Cloud for scalability and reliability.
  + Containerization: Uses containerization technologies like Docker for packaging and deploying the application components.
  + Load Balancing: Implements load balancing mechanisms to distribute incoming traffic across multiple servers and improve scalability and fault tolerance.

Low-Level Design (LLD) overall provides the detailed technical specifications necessary for implementing software modules or components. It delves into the nitty-gritty details of how individual components will function, including interfaces, and dependencies. By breaking down the system into smaller, manageable units, ensuring the system remains robust and adaptable over time.