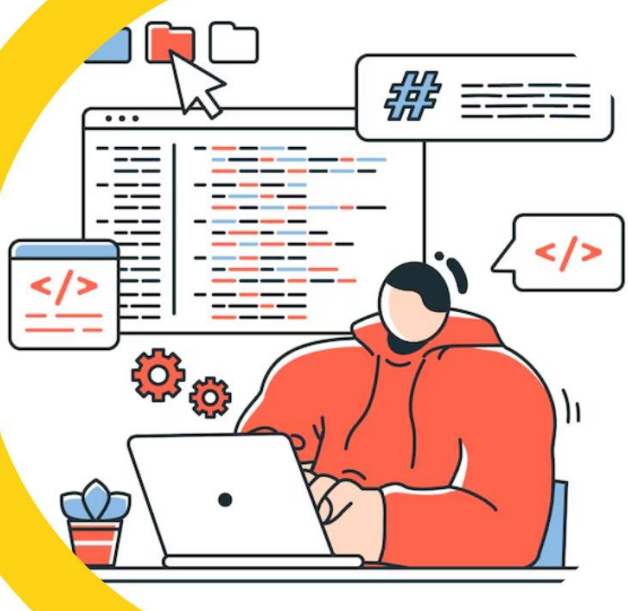
DataBase



E Commerce Store

Laiba zulfiqar
Zainab Fatima
Ali Akbar

Submitted to:

Mam Hina Rashid

Team Members

Ali Akbar – BSE 233016

Laiba Zulfiqar – BSE233022

Zainab Fatima – BSE233008

Git hub URL = https://github.com/ali-akbar-019/db-project.git

Contents

Requirements Gathering 4
1. Requirements Gathering for E-Commerce Store Database 4
2. Functional Requirements: 4
A. User Management: 4
B. Product Management: 4
C. Cart Management: 5
D. Order Management: 5
E. Wishlist: 5
F. Admin Panel: 5
3. Non-Functional Requirements: 6
A. Performance: 6
B. Scalability: 6
C. Availability: 6
D. Security: 6
E. Usability: 7
Database Implementation 8
Database Schema 8

Database Development Life Cycle Phases:

Requirements Gathering

1. Requirements Gathering for E-Commerce Store Database

The process of gathering requirements for the database involved collaborating with users (customers), administrators, and other stakeholders. This was done through interviews, surveys, and reviewing system documentation. The goal is to understand the data needs, workflows, and expectations of the database system for the e-commerce platform.

2. Functional Requirements:

Functional requirements define the specific behavior of the system, detailing what actions the system should perform in response to user or admin interactions. These requirements are derived from the core use cases that the database must support for the e-commerce platform.

A. User Management:

- **User Registration & Authentication:**
  - Users must be able to sign up using their email address or third-party authentication (e.g., Google or Facebook).
  - Upon successful registration, users should be able to log in to their accounts securely.
  - Password recovery functionality will be provided in case users forget their credentials.
- **Profile Management:**
  - Users should have the ability to view and update their personal profile information, such as name, email, phone number, and shipping address.
  - Users can set preferences like notification settings and marketing opt-ins.

B. Product Management:

- **Product Listings:**
  - Admins will add, update, and remove products in the database.
  - Each product will have details such as name, description, price, category, stock availability, and brand.
  - Products may include additional attributes like size, color, and material depending on the category.
- **Product Availability & Stock Control:**
  - Admins will be notified when stock levels of products are low.
  - Products will be marked as in-stock or out-of-stock based on inventory levels.

C. Cart Management:

- **Add/Remove Items to/from Cart:**
  - Users can add products to their shopping cart, specifying quantities.
  - Users can remove items or update quantities in the cart as needed.
- **Cart Persistence:**
  - Items in the cart should persist even if the user logs out or closes the browser.

D. Order Management:

- **Order Placement:**
  - Users can review their cart, enter shipping information, and proceed to checkout.
  - Once the order is placed, an order ID will be generated, and the order status will be set to "Placed."
- **Order Status Updates:**
  - Admins can update the status of an order (e.g., "Shipped," "Delivered," "Cancelled").
- **Order History:**
  - Users can view their past orders, including details like order date, products purchased, status, and payment status.

E. Wishlist:

- **Add/Remove Products to/from Wishlist:**
  - Users can mark products as favorites and add them to their wishlist for future reference.
  - Users can remove products from their wishlist at any time.

F. Admin Panel:

- **User and Order Management:**
  - Admins should be able to manage users by editing their profiles, viewing purchase history, and handling support requests.
  - Admins can view all orders and update their statuses (e.g., "Shipped," "Delivered").
- **Sales and Inventory Reports:**
  - Admins can generate reports for sales, product stock levels, and customer activity.

3. Non-Functional Requirements:

Non-functional requirements describe **how** the database system should operate, focusing on performance, security, scalability, and user experience. These requirements are essential for ensuring that the system performs efficiently, securely, and reliably.

A. Performance:

- **Fast Query Response Times:**
  - The system should support fast queries to ensure a smooth user experience. For instance, product search should return results within 1-2 seconds.
- **High-Throughput:**
  - The database must handle a high volume of transactions, such as hundreds or thousands of users browsing the site, placing orders, and updating inventory.

B. Scalability:

- **Horizontal Scalability:**
  - The database should be able to scale horizontally, meaning that as traffic grows, more database instances can be added to distribute the load.
- **Vertical Scaling:**
  - The system should allow upgrading the hardware (e.g., CPU, memory) for handling more data or users.

C. Availability:

- **High Availability:**
  - The database should have a high level of availability, ensuring the system is up and running 99.99% of the time. This is essential for an e-commerce platform that operates 24/7.
- **Fault Tolerance and Disaster Recovery:**
  - In the event of server failure or data corruption, the system should be capable of restoring data from backups without significant downtime.

D. Security:

- **Data Encryption:**
  - All sensitive data, including user credentials, payment information, and order details, should be encrypted both at rest and in transit.
- **Access Control:**
  - Role-based access control (RBAC) should be implemented to restrict who can perform certain actions (e.g., only admins can manage products and orders).
- **Secure Payment Processing:**
  - Integration with secure payment gateways (e.g., Stripe, PayPal) for handling payments and ensuring compliance with PCI-DSS standards.

E. Usability:

- **Intuitive User Interface (UI):**
  - The system should provide an easy-to-use interface for customers to browse products, manage their cart, and complete purchases.
- **Admin Dashboard:**
  - The admin panel should provide an intuitive dashboard for managing orders, customers, products, and reports.

Implementation

Database Implementation

According to the Entity-Relationship Diagram (ERD) and Relational Data Model (RDM) designed in the previous assignments, our database schema is structured to handle users, products, orders, cart items, favorites, and their relationships. Below is the schema implemented using **Prisma** and **MySQL** as the database management system.

Database Schema

Our database schema includes the following models:

- **User**: Stores user information like name, email, address, and phone number.
- **Product**: Contains product details such as name, brand, price, and stock.
- **Image**: Handles images associated with products.
- **CartItem**: Tracks products added to a user's cart and their order status.
- **Order**: Represents a user's completed purchase, along with delivery details.
- **Fav**: Tracks the user's favorite products.

Below is the **Prisma schema** used to define the database:

```
// This is your Prisma schema file,
// learn more about it in the docs: https://pris.ly/d/prisma-schema

// Looking for ways to speed up your queries, or scale easily with your serverless
or edge functions?
// Try Prisma Accelerate: https://pris.ly/cli/accelerate-init

generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "mysql"
  url      = env("DATABASE_URL")
}

model User {
  id             Int        @id @default(autoincrement())
  name           String?
  email          String     @unique
  auth0Id        String     @unique
  addressLine1   String?
  city           String?
  country        String?
```

```
  phone          String?    @db.VarChar(15)
  createdAt      DateTime   @default(now())
  isProfileSetup Boolean    @default(false)
  Cart           CartItem[]
  Order          Order[]
  Fav            Fav[]
}

model Product {
  id            Int      @id @default(autoincrement())
  name          String   @db.VarChar(100)
  brand         String   @db.VarChar(50)
  category      String?
  gender        Gender
  description   String?
  price         Float
  discountPrice Float?
  stock         Int      @default(0)
  isAvailable   Boolean  @default(true)
  rating        Float    @default(0)
  createdAt     DateTime @default(now())
  updatedAt     DateTime @updatedAt

  images Image[]
  Cart   CartItem[]
  Fav    Fav[]
}

model Image {
  id        Int     @id @default(autoincrement())
  url       String  @db.Text
  altText   String?
   product    Product @relation(fields: [productId], references: [id], onDelete:
Cascade)
  productId Int
}

model CartItem {
  id        Int      @id @default(autoincrement())
  user      User     @relation(fields: [userId], references: [id])
  product   Product  @relation(fields: [productId], references: [id])
  userId    Int
  productId Int
  quantity  Int
  isOrdered Boolean  @default(false)
```

```
  Order     Order?   @relation(fields: [orderId], references: [id])
  orderId   Int?
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
}

model Order {
  id              Int         @id @default(autoincrement())
  userId          Int
  user            User        @relation(fields: [userId], references: [id])
  deliveryDetails Json // Delivery details stored as JSON
  cartItems       CartItem[]
  totalAmount     Float       @default(0.0)
  status          OrderStatus @default(PLACED)
  createdAt       DateTime    @default(now())
  updatedAt       DateTime    @updatedAt
}

model Fav {
  id        Int     @id @default(autoincrement())
  userId    Int
  productId Int
  user      User    @relation(fields: [userId], references: [id])
  product   Product @relation(fields: [productId], references: [id])
}

enum Gender {
  MALE
  FEMALE
  BOTH
}

enum OrderStatus {
  PLACED
  PAID
  IN_PROGRESS
  OUT_FOR_DELIVERY
  DELIVERED
  CANCELLED
}
```

The **prisma** code translates to these queries :

SQL Queries

1. **Create** User **Table**

```
MariaDB [example]> CREATE TABLE User (
    ->     id INT AUTO_INCREMENT PRIMARY KEY,
    ->     name VARCHAR(255),
    ->     email VARCHAR(255) UNIQUE NOT NULL,
    ->     auth0Id VARCHAR(255) UNIQUE NOT NULL,
    ->     addressLine1 VARCHAR(255),
    ->     city VARCHAR(255),
    ->     country VARCHAR(255),
    ->     phone VARCHAR(15),
    ->     createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    ->     isProfileSetup BOOLEAN DEFAULT FALSE
    -> );
Query OK, 0 rows affected (0.070 sec)
```

2. **Create** Product **Table**

```
MariaDB [example]> CREATE TABLE Product (
    ->     id INT AUTO_INCREMENT PRIMARY KEY,
    ->     name VARCHAR(100) NOT NULL,
    ->     brand VARCHAR(50) NOT NULL,
    ->     category VARCHAR(255),
    ->     gender ENUM('MALE', 'FEMALE', 'BOTH') NOT NULL,
    ->     description TEXT,
    ->     price FLOAT NOT NULL,
    ->     discountPrice FLOAT,
    ->     stock INT DEFAULT 0,
    ->     isAvailable BOOLEAN DEFAULT TRUE,
    ->     rating FLOAT DEFAULT 0,
    ->     createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    ->     updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
    -> );
Query OK, 0 rows affected (0.033 sec)
```

3. **Create** Image **Table**

```
MariaDB [example]> CREATE TABLE Image (
    ->     id INT AUTO_INCREMENT PRIMARY KEY,
    ->     url TEXT NOT NULL,
    ->     altText VARCHAR(255),
    ->     productId INT NOT NULL,
    ->     FOREIGN KEY (productId) REFERENCES Product(id) ON DELETE CASCADE
    -> );
Query OK, 0 rows affected (0.043 sec)
```

4. **Create** Order **Table**

```
MariaDB [example]> CREATE TABLE `Order` (
    ->     id INT AUTO_INCREMENT PRIMARY KEY,
    ->     userId INT NOT NULL,
    ->     deliveryDetails JSON NOT NULL,
    ->     totalAmount FLOAT DEFAULT 0.0,
    ->     status ENUM('PLACED', 'PAID', 'IN_PROGRESS', 'OUT_FOR_DELIVERY', 'DELIVERED', 'CANCELLED') DEFAULT 'PLACED',
    ->     createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    ->     updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    ->     FOREIGN KEY (userId) REFERENCES User(id)
    -> );
Query OK, 0 rows affected (0.046 sec)
```

5. **Create** CartItem **Table**

```
MariaDB [example]> CREATE TABLE CartItem (
    ->     id INT AUTO_INCREMENT PRIMARY KEY,
    ->     userId INT NOT NULL,
    ->     productId INT NOT NULL,
    ->     quantity INT NOT NULL,
    ->     isOrdered BOOLEAN DEFAULT FALSE,
    ->     orderId INT,
    ->     createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
    ->     updatedAt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    ->     FOREIGN KEY (userId) REFERENCES User(id),
    ->     FOREIGN KEY (productId) REFERENCES Product(id),
    ->     FOREIGN KEY (orderId) REFERENCES `Order`(id)
    -> );
Query OK, 0 rows affected (0.033 sec)
```

6. **Create** Fav **Table**

```
MariaDB [example]> CREATE TABLE Fav (
    ->     id INT AUTO_INCREMENT PRIMARY KEY,
    ->     userId INT NOT NULL,
    ->     productId INT NOT NULL,
    ->     FOREIGN KEY (userId) REFERENCES User(id),
    ->     FOREIGN KEY (productId) REFERENCES Product(id)
    -> );
Query OK, 0 rows affected (0.025 sec)
```

Implementation Using MySQL

We used **Prisma** as the ORM (Object Relational Mapper) and **MySQL** as the database. Here is how we implemented the database:

- **Step 1: Setting Up the Environment**

Installed Prisma:

npm install prisma --save-dev

Initialized Prisma:

npx prisma init

This created the `schema.prisma` file and a `.env` file for the database connection string.

Updated the `.env` file with the MySQL connection string:

DATABASE_URL = “mysql://root:@localhost:3306/shope_ease”

- **Step 2: Pushing the Schema to MySQL**

To create the database tables in MySQL:

npx prisma db push

- **Step 3: Generating the Prisma Client**

To interact with the database using Prisma:

npx prisma generate