DBMS MINI PROJECT REPORT

ECOMMERCE MANAGEMENT SYSTEM

TEAM NO. – 02

TEAM MEMBERS

1. **DEESHA C - PES2UG23CS165**
2. **CHITRA MADARAKHANDI - PES2UG23CS157**

problem statement

In the digital era, online shopping has become an essential part of everyday life. However, many small and medium-scale businesses still struggle to maintain and manage their product data, customer information, and transaction records efficiently. Traditional manual systems or unstructured data storage lead to issues such as data redundancy, inconsistency, difficulty in product tracking, and poor user experience.

To address these challenges, there is a need for a Database Management System (DBMS)-based E-Commerce Management System that efficiently manages the entire process of online shopping — from user registration and login to product search, viewing, cart management, and order handling. The system should provide a secure and user-friendly platform that allows users to browse and purchase products, while enabling administrators to manage product inventory, user data, and transactions in a structured and reliable manner using database concepts such as normalization, relationships, and queries.

Additionally, the system must offer **strict admin-level access control**, ensuring that only authorized administrators can add, update, or delete product details. All administrative actions should be securely tracked to maintain data accuracy, transparency, and prevent unauthorized modifications. This ensures smooth management of the platform and enhances overall data reliability.

Abstract

The E-Commerce Management System is a database-driven application designed to simplify online shopping and streamline data handling. It enables users to register, log in, search for products, view details, and add items to the cart for purchase. On the administrative side, the system allows admins to securely manage product information, oversee customer records, update inventory, monitor transactions, and ensure smooth platform operation.

By applying key database management concepts such as normalization, relational schema design, and optimized queries, the project ensures structured data storage, fast retrieval, and efficient management for both users and administrators.

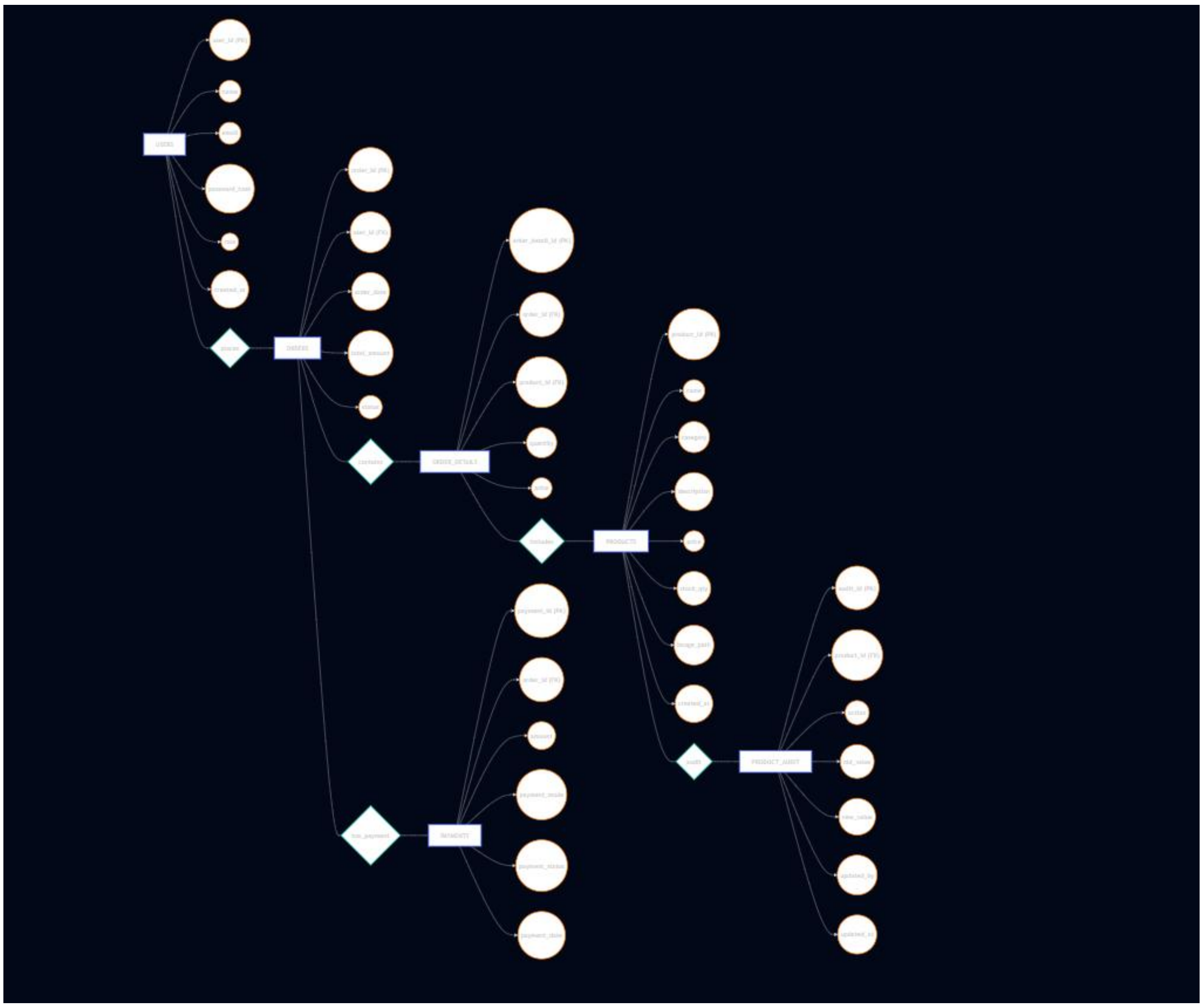
3. User Requirement Specification (Prepared for Review-1)

- **User Registration and Login:**
  Users should be able to create an account and log in using valid credentials.
- **Product Browsing and Search:**
  Users can view all available products and search for specific products by name or category.
- **Cart Management:**
  Users can add products to the cart, view the cart, update quantities, or remove items before purchasing.
- **Product Management (Admin):**
  Administrators can add, update, or delete product details such as name, price, category, and stock.
- **Admin Access Control and Audit Monitoring:**
  Only administrators should have permission to update or delete products.
  All admin actions (updates/deletions) should be automatically logged for transparency and security.
- **Order Management:**
  The system should store order details, including user information, product list, quantity, and total price.
- **Database Management:**
  All data related to users, products, and transactions should be stored and managed efficiently using DBMS concepts to ensure consistency and avoid redundancy.
- **User Interface:**
  The interface should be simple, interactive, and easy to navigate for both users and administrators.

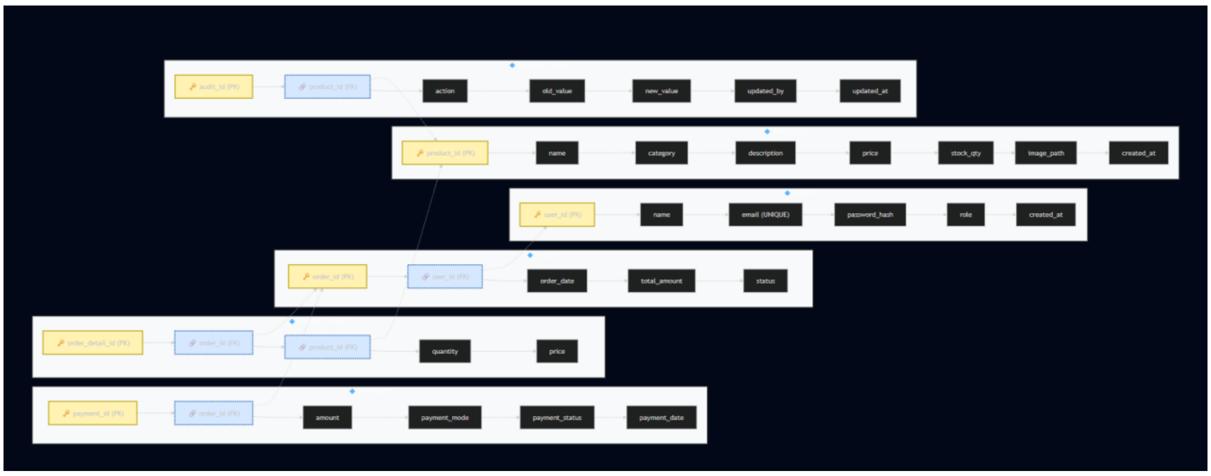
List of Softwares / Tools / Programming Languages Used

- **Frontend:** HTML,CSS AND JavaScropt
- **Backend:** Python
- **Database:** MySQL
- **Development Tools:** Visual Studio Code
- **Version Control:** Git and GitHub

ER Diagram



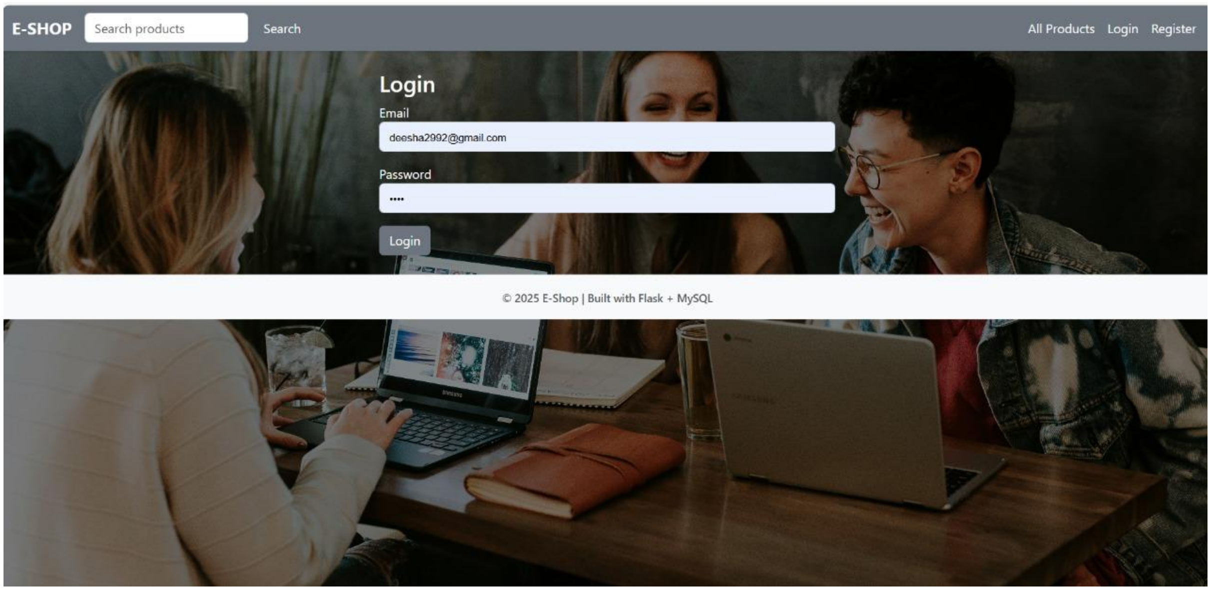
Relational Schema



DDL Commands

- **DDL:** DROP DATABASE, CREATE DATABASE, USE, CREATE TABLE, CREATE TRIGGER, CREATE PROCEDURE, CREATE FUNCTION.
- **DML**: INSERT INTO, UPDATE, DELETE, SELECT.
- **Session / safety:** SET SQL_SAFE_UPDATES.
- **Flow control:** use of DELIMITER for trigger/proc/function blocks.

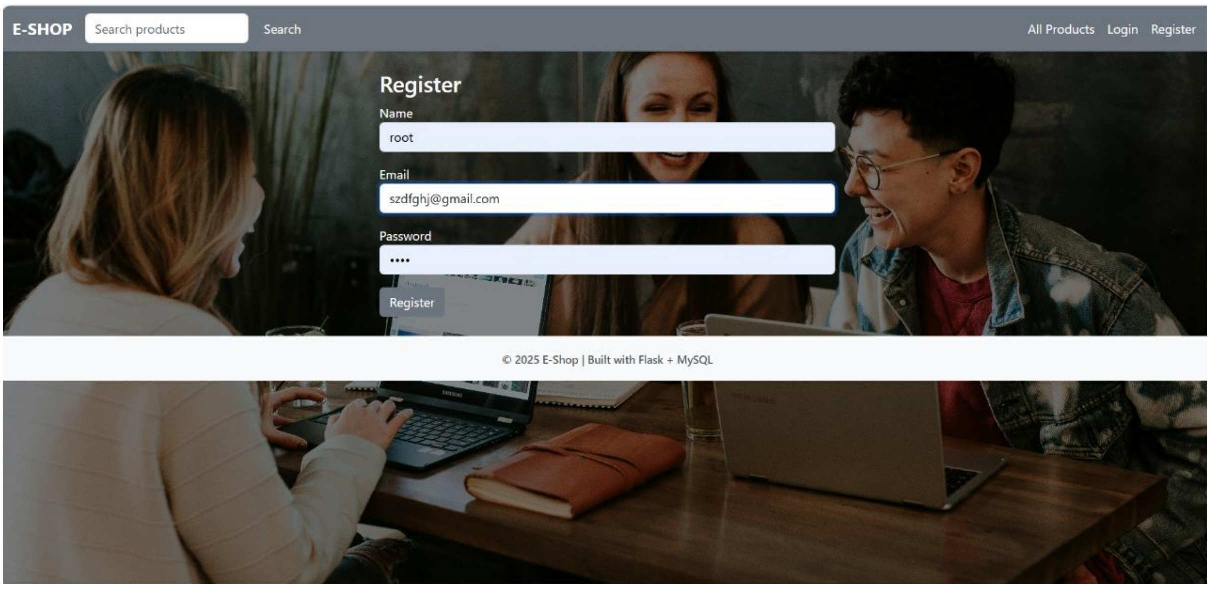
CRUD operation Screenshots



deesha2992@gmail.com

••••

© 2025 E-Shop | Built with Flask + MySQL


E-SHOP Search products Search All Products Login Register

Register

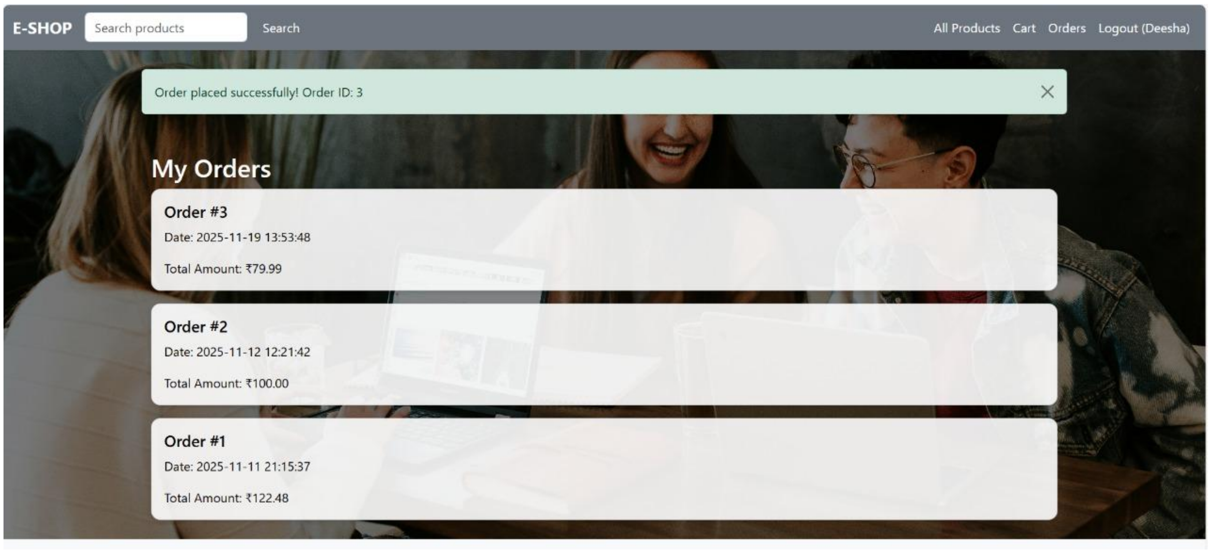
Name: root

Email: szdfghj@gmail.com

Password: ••••

Register

© 2025 E-Shop | Built with Flask + MySQL


E-SHOP Search products Search All Products Cart Orders Logout (Deesha)

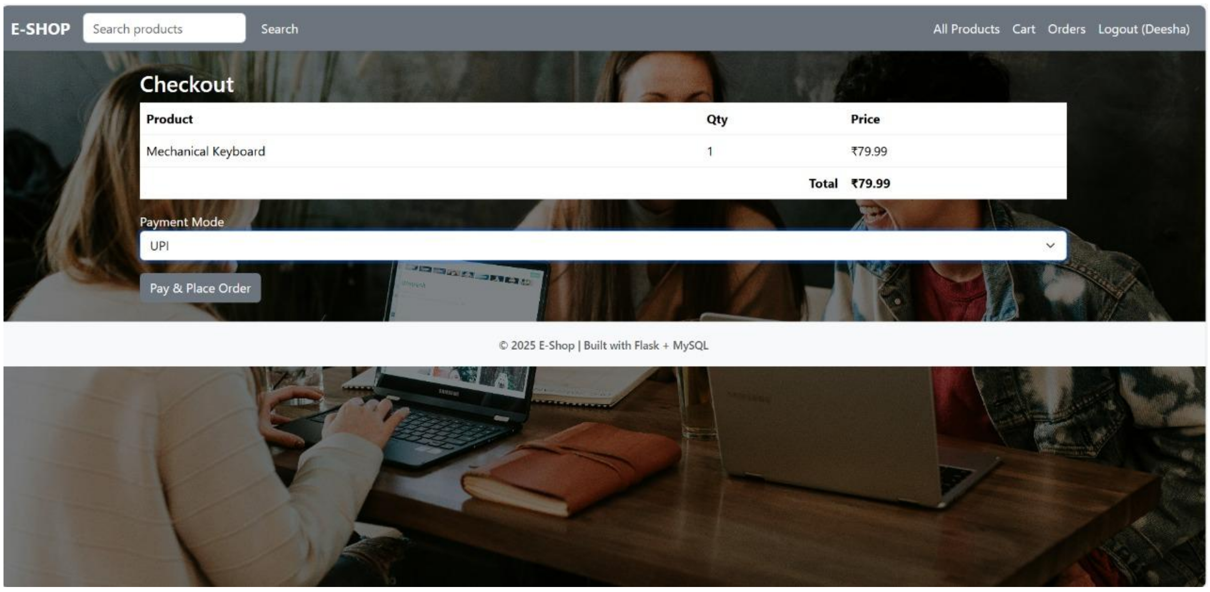
Order placed successfully! Order ID: 3

My Orders

Order #3
Date: 2025-11-19 13:53:48
Total Amount: ₹79.99

Order #2
Date: 2025-11-12 12:21:42
Total Amount: ₹100.00

Order #1
Date: 2025-11-11 21:15:37
Total Amount: ₹122.48


E-SHOP Search products Search All Products Cart Orders Logout (Deesha)

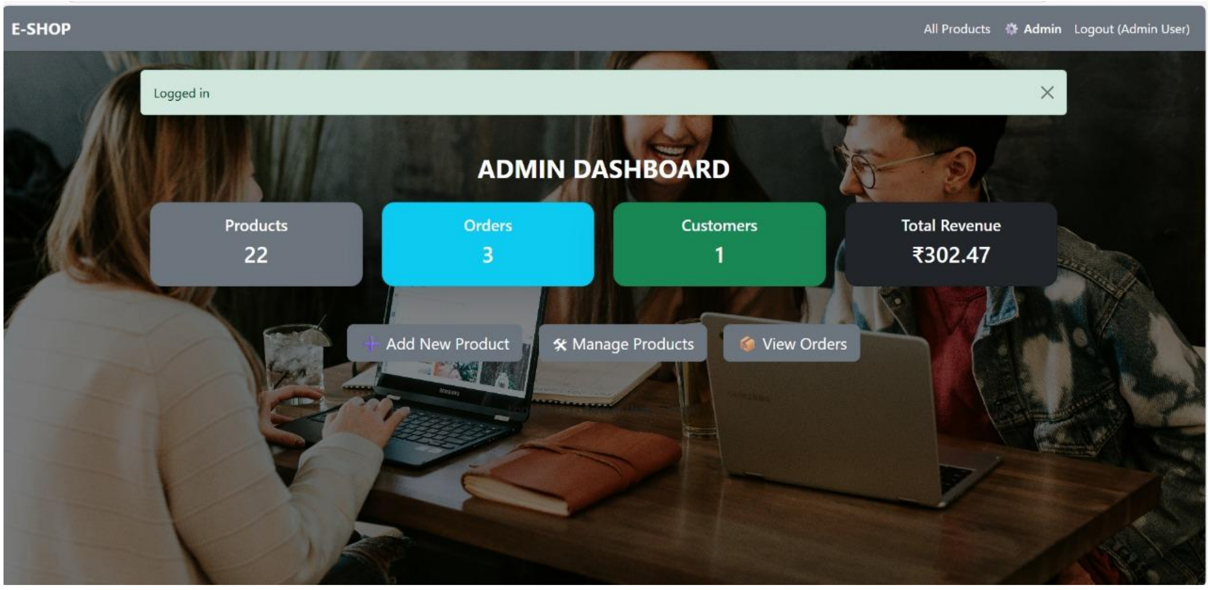
Checkout

Product	Qty	Price
Mechanical Keyboard	1	₹79.99
	Total	**₹79.99**

Payment Mode: UPI

Pay & Place Order

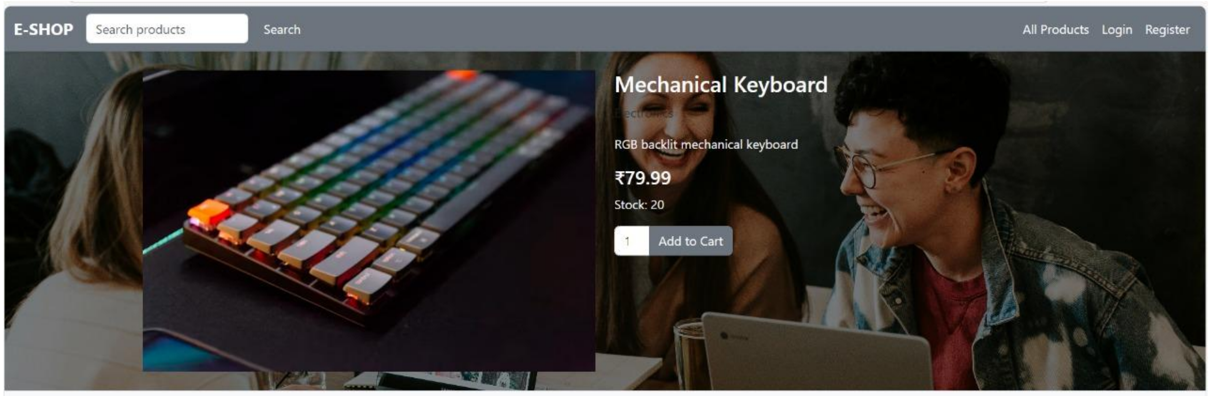
© 2025 E-Shop | Built with Flask + MySQL



Logged in

ADMIN DASHBOARD

Products	Orders	Customers	Total Revenue
22	3	1	₹302.47



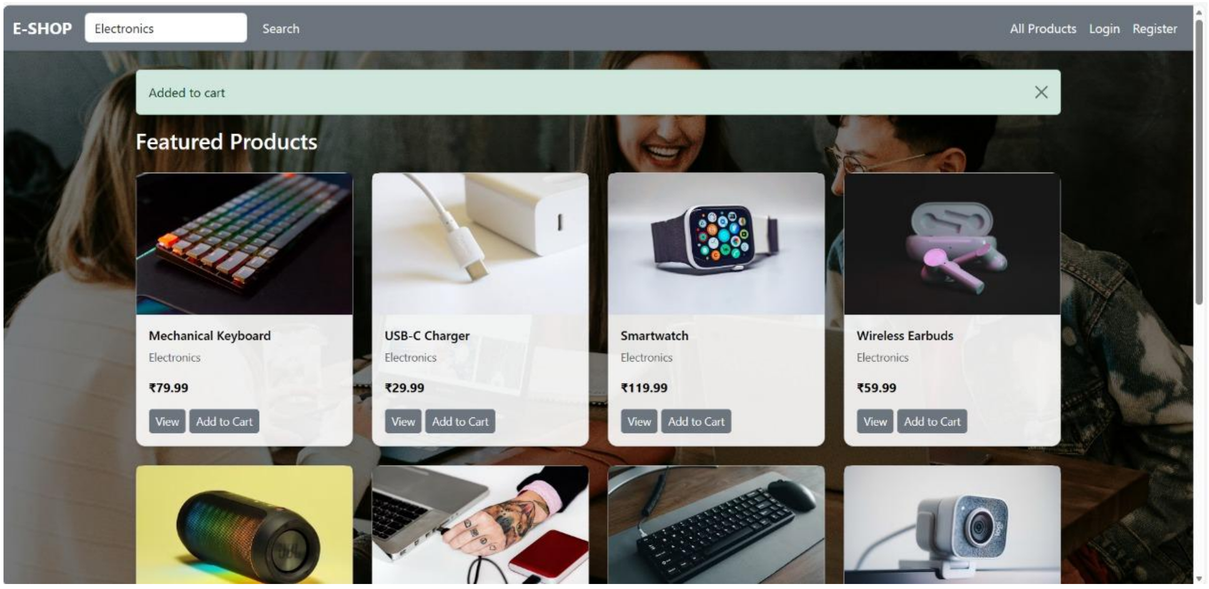
Mechanical Keyboard

Electronics

RGB backlit mechanical keyboard

₹79.99

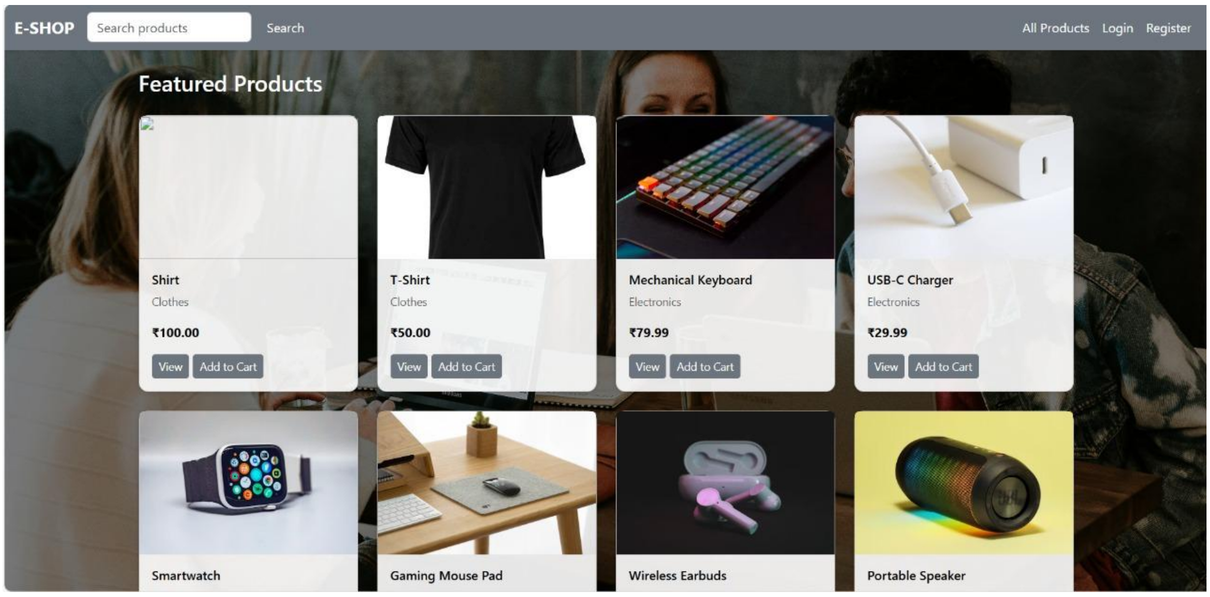
Stock: 20



Added to cart

Featured Products

Mechanical Keyboard	USB-C Charger	Smartwatch	Wireless Earbuds
Electronics	Electronics	Electronics	Electronics
₹79.99	₹29.99	₹119.99	₹59.99



Featured Products

Shirt
Clothes
₹100.00

T-Shirt
Clothes
₹50.00

Mechanical Keyboard
Electronics
₹79.99

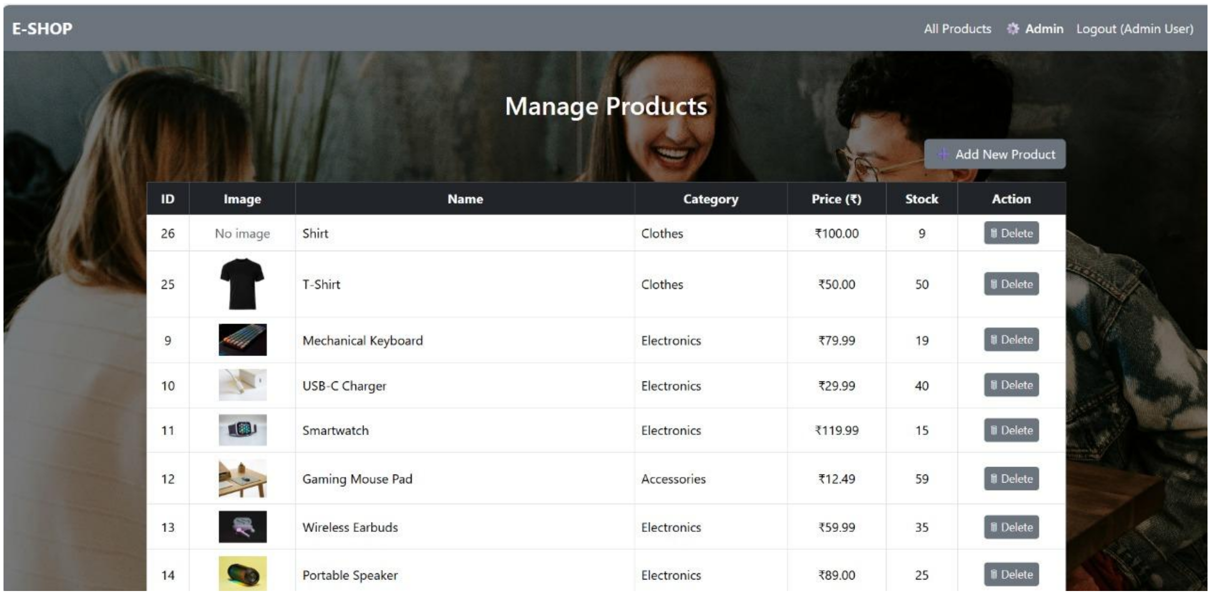
USB-C Charger
Electronics
₹29.99

Smartwatch

Gaming Mouse Pad

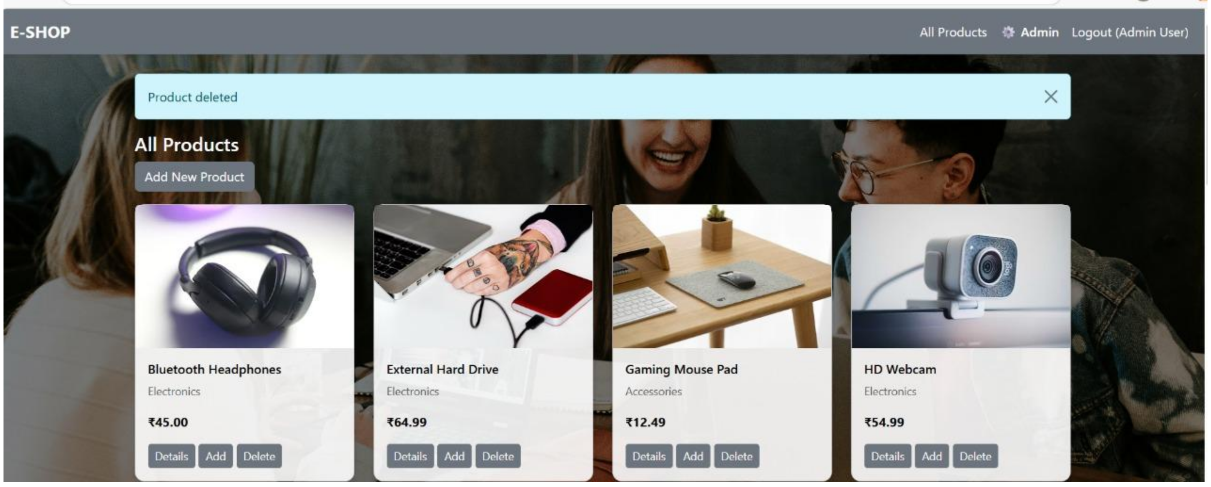
Wireless Earbuds

Portable Speaker



Manage Products

ID	Image	Name	Category	Price (₹)	Stock	Action
26	No image	Shirt	Clothes	₹100.00	9	Delete
25		T-Shirt	Clothes	₹50.00	50	Delete
9		Mechanical Keyboard	Electronics	₹79.99	19	Delete
10		USB-C Charger	Electronics	₹29.99	40	Delete
11		Smartwatch	Electronics	₹119.99	15	Delete
12		Gaming Mouse Pad	Accessories	₹12.49	59	Delete
13		Wireless Earbuds	Electronics	₹59.99	35	Delete
14		Portable Speaker	Electronics	₹89.00	25	Delete



Product deleted

All Products

Bluetooth Headphones
Electronics
₹45.00

External Hard Drive
Electronics
₹64.99

Gaming Mouse Pad
Accessories
₹12.49

HD Webcam
Electronics
₹54.99

List of functionalities/features of the application and its associated screenshots using front end

- **User Registration:**
  New users can register by entering their name, email, and password. The information is securely stored in the database.
- **User Login:**
  Registered users can log in using their email and password to access their personalized shopping dashboard.
- **Product Listing:**
  All available products are displayed with details such as name, category, description, price, and stock quantity.
- **Product Search and Filtering:**
  Users can search for products by name or category to quickly find the items they need.
- **Add to Cart:**
  Users can add selected products to the shopping cart for later checkout.
- **View Cart and Update Quantity:**
  Users can view all products in their cart, change quantities, or remove items before confirming the order.
- **Place Order:**
  Users can confirm their orders. Once confirmed, order details are stored in the database and stock quantity is updated automatically (via trigger).
- **Admin-only product control**

  Only the admin can update or delete products. Non-admin users are blocked by triggers.

- **Admin action logging**
  All admin updates and deletions are recorded in the audit table for tracking.

Triggers, Procedures/Functions, Nested query, Join, Aggregate queries

1. Triggers (Used in your code)

- Triggers automatically run when UPDATE or DELETE happens on the **products** table.

- They block non-admin users from modifying or deleting products (security control).

- They also log every admin update or delete into the **product_audit** table.

2. Procedures / Functions (Not used in your code, but explanation as per context)

- Allow reusable operations like calculating totals, updating stock, or processing an order.
- Help reduce repeated SQL logic by storing common tasks inside functions/procedures.
- Improve performance and consistency across the whole application.

3. Nested Queries (Not used directly, but meaning in your project)

- They help fetch data using one query inside another—for example, finding users who placed the most orders.

- Useful for filtering results with conditions dependent on another query (subquery).

- Can simplify complex logic, like calculating a product's sales or checking stock levels.

4. Joins (Used conceptually through foreign keys in your code)

- Joins combine data from related tables such as **orders**, **order_details**, and **users**.

- Used to fetch full order information (customer + product + payment details).

- Built-in relations using FOREIGN KEY support efficient JOIN operations.

5. Aggregate Queries (Not written in code but applicable to this DB)

- They perform calculations like SUM(total_amount), COUNT(orders), etc.

- Useful for admin reports: total sales, product stock summary, number of customers, etc.

- Work with GROUP BY to summarize data—for example, sales per product or per month.

Code snippets for invoking the Procedures/Functions/Trigger

As non-admin

```
UPDATE products SET price = 199.99 WHERE product_id = 1;

DELETE FROM products WHERE product_id = 2;
```

As admin

```
UPDATE products SET price = 179.99, name = 'New Product Name' WHERE product_id = 1;

DELETE FROM products WHERE product_id = 2;
```

Check audit log

```
SELECT * FROM product_audit ORDER BY updated_at DESC LIMIT 5;
```

2) Procedure Creation

```
DELIMITER $$

CREATE PROCEDURE sp_update_product_price(

  IN in_product_id INT,

  IN in_new_price DECIMAL(10,2)

)

BEGIN

  IF CURRENT_USER() NOT LIKE 'eshop_admin%' THEN

    SIGNAL SQLSTATE '45000'

    SET MESSAGE_TEXT = 'Only admin may call sp_update_product_price';

  END IF;

  UPDATE products

  SET price = in_new_price

  WHERE product_id = in_product_id;



  INSERT INTO product_audit(product_id, action, old_value, new_value, updated_by)

  VALUES (in_product_id, 'PROC_UPDATE_PRICE',

      CONCAT('price updated to ', in_new_price),

      NULL, CURRENT_USER());

END$$

DELIMITER ;
```

Call Procedure

```
CALL sp_update_product_price(1, 149.99);
```

3) Function Creation

```
DELIMITER $$
CREATE FUNCTION fn_get_stock(in_pid INT)
RETURNS INT
DETERMINISTIC
READS SQL DATA
BEGIN
  DECLARE v_stock INT DEFAULT 0;
  SELECT stock_qty INTO v_stock FROM products WHERE product_id = in_pid;
  RETURN IFNULL(v_stock, 0);
END$$
DELIMITER ;
```

Call Function

```
SELECT fn_get_stock(1);
```

4) Transactional Procedure (Create Order)

```
DELIMITER $$
CREATE PROCEDURE sp_create_order(
  IN in_user_id INT,
  IN in_product_id INT,
  IN in_qty INT
)
BEGIN
  DECLARE v_price DECIMAL(10,2);
  START TRANSACTION;
  SELECT price, stock_qty INTO v_price, @stock
  FROM products WHERE product_id = in_product_id FOR UPDATE;
  IF @stock < in_qty THEN
    ROLLBACK;
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Insufficient stock';
  END IF;
  INSERT INTO orders(user_id, total_amount)
  VALUES (in_user_id, v_price * in_qty);
  SET @new_order_id = LAST_INSERT_ID();
  INSERT INTO order_details(order_id, product_id, quantity, price)
  VALUES (@new_order_id, in_product_id, in_qty, v_price);
  UPDATE products
  SET stock_qty = stock_qty - in_qty
  WHERE product_id = in_product_id;
  COMMIT;
END$$
DELIMITER ;
```

Call Procedure

```
CALL sp_create_order(3, 5, 2);
```

SQL queries(Create, Insert, Triggers, Procedures/Functions, Nested query, Join, Aggregate queries) used in the project in the form of .sql file

```
DROP DATABASE IF EXISTS ecommerce_db;
CREATE DATABASE ecommerce_db;
USE ecommerce_db;
CREATE TABLE users (
  user_id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(120) NOT NULL,
  email VARCHAR(150) NOT NULL UNIQUE,
  password_hash VARCHAR(255) NOT NULL,
  role ENUM('admin','customer') DEFAULT 'customer',
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
CREATE TABLE products (
  product_id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(200) NOT NULL,
  category VARCHAR(100),
  description TEXT,
  price DECIMAL(10,2) NOT NULL,
  stock_qty INT DEFAULT 0,
  image_path VARCHAR(255),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
CREATE TABLE orders (
  order_id INT AUTO_INCREMENT PRIMARY KEY,
  user_id INT NOT NULL,
  order_date DATETIME DEFAULT CURRENT_TIMESTAMP,
  total_amount DECIMAL(10,2) DEFAULT 0.00,
  status ENUM('Pending','Confirmed','Shipped','Delivered','Cancelled') DEFAULT 'Pending',
  FOREIGN KEY (user_id) REFERENCES users(user_id)
);
CREATE TABLE order_details (
  order_detail_id INT AUTO_INCREMENT PRIMARY KEY,
  order_id INT NOT NULL,
  product_id INT NOT NULL,
  quantity INT NOT NULL,
  price DECIMAL(10,2) NOT NULL,
  FOREIGN KEY (order_id) REFERENCES orders(order_id) ON DELETE CASCADE,
  FOREIGN KEY (product_id) REFERENCES products(product_id)
);
```

```
CREATE TABLE payments (
  payment_id INT AUTO_INCREMENT PRIMARY KEY,
  order_id INT NOT NULL,
  amount DECIMAL(10,2) NOT NULL,
  payment_mode VARCHAR(60),
  payment_status ENUM('Pending','Success','Failed') DEFAULT 'Pending',
  payment_date DATETIME DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (order_id) REFERENCES orders(order_id) ON DELETE CASCADE
);
CREATE TABLE product_audit (
  audit_id INT AUTO_INCREMENT PRIMARY KEY,
  product_id INT,
  action VARCHAR(20),
  old_value TEXT,
  new_value TEXT,
  updated_by VARCHAR(100),
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
CREATE USER 'eshop_admin'@'%' IDENTIFIED BY 'Admin123!';
GRANT ALL PRIVILEGES ON ecommerce_db.* TO 'eshop_admin'@'%';
CREATE USER 'eshop_user'@'%' IDENTIFIED BY 'User123!';
GRANT SELECT, INSERT ON ecommerce_db.* TO 'eshop_user'@'%';
FLUSH PRIVILEGES;
DELIMITER $$
CREATE TRIGGER trg_block_update
BEFORE UPDATE ON products
FOR EACH ROW
BEGIN
    IF CURRENT_USER() NOT LIKE 'eshop_admin%' THEN
        SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Only admin can UPDATE products';
  END IF;
END$$
CREATE TRIGGER trg_block_delete
BEFORE DELETE ON products
FOR EACH ROW
BEGIN
  IF CURRENT_USER() NOT LIKE 'eshop_admin%' THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Only admin can DELETE products';
  END IF;
END$$
CREATE TRIGGER trg_log_update
AFTER UPDATE ON products
FOR EACH ROW
BEGIN
  INSERT INTO product_audit(product_id, action, old_value, new_value, updated_by)
  VALUES (
    OLD.product_id,
    'UPDATE',
    CONCAT('Old name=', OLD.name, ', Old price=', OLD.price),
    CONCAT('New name=', NEW.name, ', New price=', NEW.price),
    CURRENT_USER()
  );
END$$
CREATE TRIGGER trg_log_delete
AFTER DELETE ON products
FOR EACH ROW
BEGIN
  INSERT INTO product_audit(product_id, action, old_value, new_value, updated_by)
  VALUES (
    OLD.product_id,
    'DELETE',
    CONCAT('Deleted product: ', OLD.name, ', price=', OLD.price),
    NULL,
    CURRENT_USER()
  );
END$$
DELIMITER ;
INSERT INTO users(name, email, password_hash, role) VALUES
('Admin User','admin@example.com','hash_admin','admin'),
('Alice','alice@example.com','hash_alice','customer'),
('Bob','bob@example.com','hash_bob','customer');
INSERT INTO products(name, category, description, price, stock_qty, image_path)
VALUES
('Wireless Mouse','Electronics','Ergonomic wireless
mouse',499.00,50,'/images/mouse.jpg'),
('Mechanical Keyboard','Electronics','RGB mechanical
keyboard',2999.00,20,'/images/keyboard.jpg'),
('Water Bottle','Home','Stainless steel bottle',799.00,100,'/images/bottle.jpg');
INSERT INTO orders(user_id, total_amount, status) VALUES
(2, 998.00, 'Confirmed');
INSERT INTO order_details(order_id, product_id, quantity, price) VALUES
(1, 1, 2, 499.00);
INSERT INTO payments(order_id, amount, payment_mode, payment_status) VALUES
(1, 998.00, 'Credit Card', 'Success');
DELIMITER $$
CREATE PROCEDURE sp_update_product_price(
  IN in_product_id INT,
  IN in_new_price DECIMAL(10,2)
)
```

```
BEGIN
  IF CURRENT_USER() NOT LIKE 'eshop_admin%' THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Only admin may call sp_update_product_price';
  END IF;
  UPDATE products
  SET price = in_new_price
  WHERE product_id = in_product_id;

  INSERT INTO product_audit(product_id, action, old_value, new_value, updated_by)
  VALUES (in_product_id, 'PROC_UPDATE_PRICE', CONCAT('price updated to ',
in_new_price), NULL, CURRENT_USER());
END$$
CREATE FUNCTION fn_get_stock(in_pid INT)
RETURNS INT
DETERMINISTIC
READS SQL DATA
BEGIN
  DECLARE v_stock INT DEFAULT 0;
  SELECT stock_qty INTO v_stock FROM products WHERE product_id = in_pid;
  RETURN IFNULL(v_stock, 0);
END$$
CREATE PROCEDURE sp_create_order(
  IN in_user_id INT,
  IN in_product_id INT,
  IN in_qty INT
)
BEGIN
  DECLARE v_price DECIMAL(10,2);
  START TRANSACTION;
  SELECT price, stock_qty INTO v_price, @stock
  FROM products WHERE product_id = in_product_id FOR UPDATE;
  IF @stock < in_qty THEN
    ROLLBACK;
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Insufficient stock';
  END IF;
  INSERT INTO orders(user_id, total_amount) VALUES (in_user_id, v_price * in_qty);
  SET @new_order_id = LAST_INSERT_ID();
  INSERT INTO order_details(order_id, product_id, quantity, price)
  VALUES (@new_order_id, in_product_id, in_qty, v_price);
  UPDATE products SET stock_qty = stock_qty - in_qty WHERE product_id =
in_product_id;
  COMMIT;
END$$
DELIMITER ;
SELECT * FROM products WHERE price > (SELECT AVG(price) FROM products);
SELECT u.name, o.order_id, od.product_id, od.quantity, od.price
FROM users u
JOIN orders o ON u.user_id = o.user_id
JOIN order_details od ON o.order_id = od.order_id;
SELECT product_id, SUM(quantity) AS total_sold, SUM(quantity*price) AS total_revenue
FROM order_details
GROUP BY product_id
ORDER BY total_revenue DESC;
SELECT u.user_id, u.name, COUNT(o.order_id) AS orders_count, SUM(o.total_amount)
AS total_spent
FROM users u
LEFT JOIN orders o ON u.user_id = o.user_id
GROUP BY u.user_id, u.name
HAVING orders_count > 0
```

```
ORDER BY total_spent DESC;
```

Github repo link

https://github.com/chitramadarakhandi/PES2UG23CS157_165_DBMS_MINIPROJECT_ECOMMERCE-MANAGEMENT-SYSTEM.git