Small-Scale e-commerce inventory system

FISAC

Web Technologies ICT 3131

Team Members:
Name: ANIRUDHA.L.SARALAYA
Reg. No.220953663
Roll No. 61

Name: H GANAPATHI KAMATH
Reg. No.220953667
Roll No.62

Name: S SUDHANVA KALKURA
Reg. No.220953016
Roll No. 7

Date

05-11-2024

Contents

GOOGLE,YOUTUBE,W3SCHOOLS

List of Figures

1 Introduction

Purpose:

The Small-Scale E-commerce Inventory System aims to help small online stores with simple inventory management automation. The database application intends to follow product, stock, and order details. This application easily manages every day-to-day inventory business to ensure real-time stock in the store.

Significance:

This project addresses common inventory challenges small businesses face, such as stock-outs, overstocking, and manual tracking errors. By automating inventory processes, the system helps reduce operational costs, improve order accuracy, and enhance customer satisfaction by ensuring timely order fulfillment.

Scope:
The scope of this project covers all fundamental aspects of inventory management, including:

- Product catalog management with details like product name, category, stock quantity, order quantity, and prices.
- Real-time stock level tracking.
- Order management, from order placement to fulfillment.

Specific goals and objectives of the database project.

Centralized Inventory Management:

To create a centralized database that consolidates product, stock, and seamless inventory tracking.

Automate Stock Tracking:

To implement automatic stock adjustments based on sales and order fulfillment, reducing the risk of human errors.

Real-Time Stock Updates:

To enable real-time inventory updates, providing users with accurate stock levels and alerting them of low stock to prevent stockouts.

User-Friendly Interface:

To design a user-friendly system that allows easy navigation and quick access to inventory data.

2 Design

<u>Database Design</u>

The database design follows a relational structure with tables representing different entities within the inventory system.

User Interface Design

While the database is the core of the system, the design includes a **user-friendly interface** for interaction. The interface will likely consist of the following modules:

- **Dashboard:** Provides an overview of stock levels, and recent orders.
- **Inventory Management:** Allows users to view, add, update, and categorize products, and track inventory levels in real-time.
- **Order Management:** Displays order information, enabling users to process, update, and fulfill orders.

Data Integrity and Security

The database design includes mechanisms for ensuring data accuracy and security:

- **Validation Constraints:** Enforces rules such as mandatory fields (e.g., product name) and unique fields (e.g., order ID, product ID).

ARCHITECTURE

The **architecture of the Small-Scale E-commerce Inventory System** is designed with modularity, scalability, and user accessibility in mind.

Client Interface

- **Purpose**: Acts as the interface between the user and the system, enabling interaction with inventory data.
- **Components**: The front-end application (web-based, desktop) is built using technologies like HTML, CSS, and JavaScript.
- **Functionality**: Provides a user-friendly interface for managing inventory, viewing order details, accessing reports, and interacting with product data.
- **Role**: Receives user input, sends requests to the application layer, and displays responses. It's responsible for data input validation before sending it to the backend.

Business Logic

- **Purpose**: Processes requests from the presentation layer and performs business logic before interacting with the data layer.
- **Components**: This layer contains the server-side application code written in languages like PHP.
- **Functionality**: Implements core business functions such as:
    - Handling CRUD (Create, Read, Update, Delete) operations on products and orders
    - Validating and processing orders (e.g., checking stock levels before confirming an order).
    - Managing stock level updates in real-time based on transactions.
- **Role**: Acts as an intermediary between the client interface and the database, ensuring secure and validated data flow.

Database

- **Purpose**: Stores, manages, and organizes all the data related to the inventory system.
- **Components**: A relational database management system (RDBMS) such as MySQL structured with tables that organize entities like Products, Orders, and Categories.
- **Functionality**:
    - Stores structured data in normalized tables to avoid redundancy and maintain data integrity.
    - Executes SQL queries received from the application layer for retrieving or updating data.
    - Manages stored procedures, triggers, and views for more efficient and automated data handling.
- **Role**: Provides data persistence, ensuring data is available and recoverable whenever needed. It also enforces referential integrity between related tables.

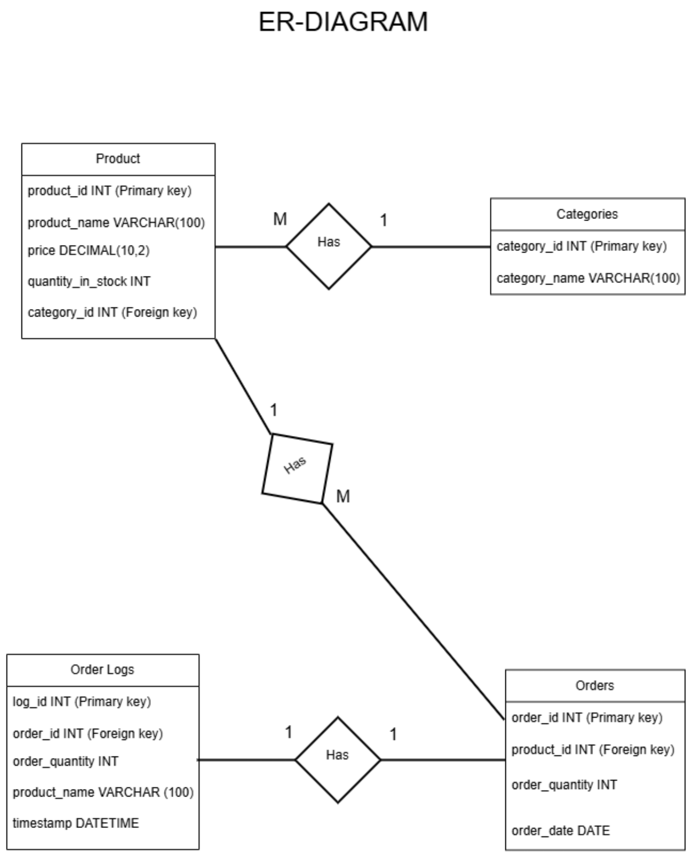
System Workflow

- **User Requests**: Users interact with the system via the presentation layer, accessing functions like adding new products, updating stock levels, or processing orders.
- **Business Logic Processing**: The application layer receives these requests, applies business logic (e.g., checking stock levels before confirming an order), and prepares the data for storage or retrieval.
- **Database Interaction**: The application layer interacts with the data layer to either retrieve data for display (e.g., current stock levels) or store updates (e.g., new orders or inventory changes).
- **Data Response**: The data layer responds with the requested data or confirms the completion of a data update, which is passed back through the application layer to the presentation layer.

Technology Stack

- **Frontend**: HTML, CSS, and JavaScript, for a responsive and interactive user experience.
- **Backend**: PHP for processing business logic and handling API requests.
- **Database**: MySQL for data storage with an organized relational schema.

Schema design, and Data modeling.



ER-DIAGRAM
Product
product_id INT (Primary key)
product_name VARCHAR(100)
price DECIMAL(10,2)
quantity_in_stock INT
category_id INT (Foreign key)
Has
M
1
Categories
category_id INT (Primary key)
category_name VARCHAR(100)
1
Has
M
Order Logs
log_id INT (Primary key)
order_id INT (Foreign key)
order_quantity INT
product_name VARCHAR (100)
timestamp DATETIME
1
Has
1
Orders
order_id INT (Primary key)
product_id INT (Foreign key)
order_quantity INT
order_date DATE

FIGURE 2

3 Implementation Details

Database Implementation

The database forms the system's core, responsible for storing and managing all inventory data. Key steps in implementing the database include:

- **Database Setup**:
  - Use a relational database management system (RDBMS) like **MySQL**.
  - Create tables such as **Products**, **Orders**, **Order Logs**, and **Categories** based on the data model.
  - Implement primary keys, foreign keys, and constraints to enforce relationships and data integrity between tables.
- **Schema Creation**:
  - Define each table's schema by specifying attributes, data types, and constraints (e.g., setting product_id as the primary key in the Products table).
  - Use SQL scripts to create the tables and constraints, ensuring the database schema matches the data model.

- **Stored Procedures & Triggers**:
    - **Stored Procedures/ Triggers**: Implement procedures for complex operations, such as generating order logs and getting history of sales date and time.

Frontend Implementation (GUI)

The front end is a user interface that allows users to interact with the system. Key sections of the GUI include:

- **Dashboard**:
    - Displays an overview of the inventory, recent orders, and low-stock alerts.
    - Provides quick access to frequently used features such as adding products, and viewing orders.
- **Inventory Management**:
    - Allows users to view, add, edit, or delete products in the inventory.
    - Displays product details, including name, price, stock quantity, and category.
- **Order Management**:
    - Allows users to create, view, and update orders, displaying details such as order ID, product name, order quantity, and order date.
    - Shows stock availability and alerts users if the requested quantity exceeds available stock.
- **Reporting and Analytics**:
    - Allows users to generate order history, helping in decision-making.

Implementation Workflow

1. **Database Setup and Configuration**:
    - Define the schema, implement tables, and set up constraints.
    - Populate tables with sample data for testing.
2. **Backend Development**:
    - Build the API, define routes, and set up business logic.
3. **Frontend Development**:
    - Develop UI components for each feature (e.g., Dashboard, Order Management).
    - Integrate the frontend with the backend API to enable data fetching and submission.
4. **Testing and Debugging**:
    - Perform functional testing for each feature to ensure they meet requirements.
    - Conduct integration testing to ensure smooth interaction between frontend, backend, and database.
5. **Deployment and Maintenance**:
    - Deploy the application on a server like PHP.

4 Testing and Validation

The testing process for the e-commerce inventory system comprises unit testing, integration testing, and acceptance testing.

Unit Testing:

- Each component underwent individual testing to guarantee accurate operation. Testing the insertion of sample data into each table was conducted to validate primary, foreign key, and unique constraints.
- The stock reduction logic in the PHP script was tested to verify that the system appropriately reduces stock quantity upon order placement. Error handling was validated by attempting to place orders that exceeded the available stock. The system exhibited a custom error message, as anticipated.

Integration Testing:

- The interactions between components were tested to verify combined functionality. Sample orders were placed. It was confirmed that the Orders table was updated correctly, and stock levels in the Products table were adjusted accordingly.
- The trigger on the Orders table was tested and accurately recorded every new order into the order_logs table, along with precise timestamps and order quantities. The stored procedure was evaluated through the execution of queries on product details, category information, and total orders, yielding the anticipated outcomes.

Acceptance Testing:

- The system underwent testing to verify its compliance with user requirements and its ability to manage real-world situations. Various order scenarios were tested, including orders that surpass the current stock levels, to verify that the system presented relevant error messages. The removal of the Categories table was assessed, and its influence on the Products table was observed. Referential integrity was upheld by imposing correct foreign key constraints.

Manual Testing:

- Manual testing was conducted to validate the system from the end-user's viewpoint. Users engaged with the application through manual interaction, navigating product listings, making orders, and verifying stock levels.
- Each feature was thoroughly tested by inputting various values, including edge cases, to verify that the system functioned as anticipated. Manual tests encompassed authenticating the layout and usability of the user interface, guaranteeing its user-friendliness and accessibility. Feedback was collected from testers to pinpoint areas for enhancement in both functionality and user experience.

5 Results and Discussion

The evaluation of the performance and functionality of the e-commerce inventory system proceeded as delineated:

System Functionality:

- The database tables were populated with sample data, and screenshots of the Products, Categories, and Orders tables were captured to illustrate their initial states. The system effectively decreased the stock quantity in the Products table upon the placement of an order. The order_logs table exhibited successful logging of orders, showcasing precise timestamps and quantities due to the trigger.

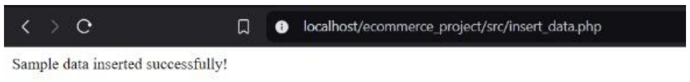
Error Handling:

- When an order is submitted with a quantity that exceeds the available stock, the system will show a customized error message stating that there is insufficient stock, showcasing proficient error handling. The system upheld referential integrity following the deletion of the Categories table, with the foreign key constraints in the Products table preventing the existence of orphaned records.

Stored Procedure Validation:

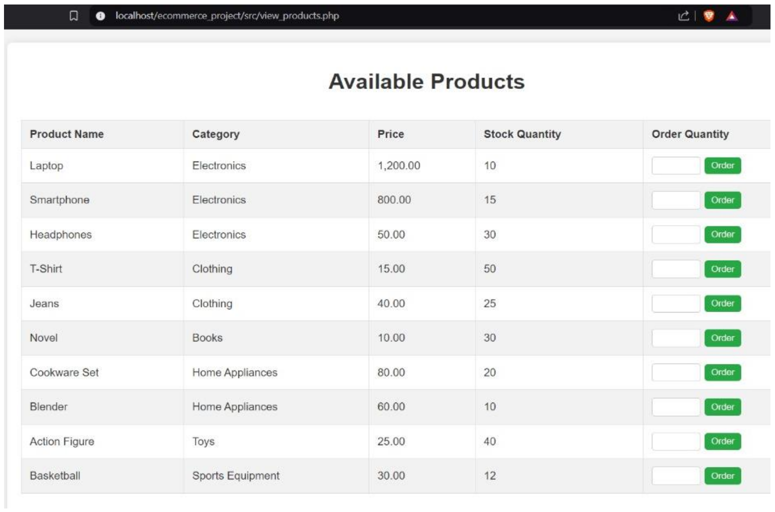
- The stored procedure successfully retrieved the product name, category, and the total number of orders as anticipated, thus verifying its functionality.

Here are some of the screenshots



```
Sample data inserted successfully!
```

Figure 5

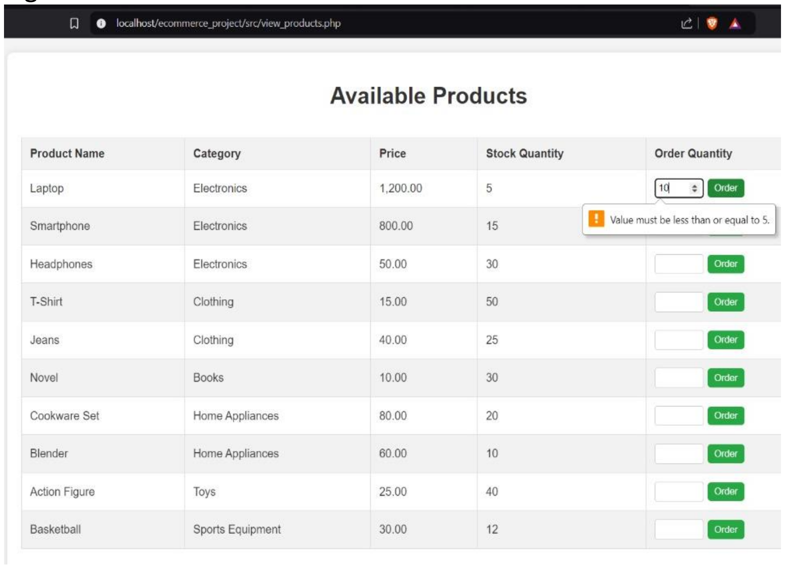


localhost/ecommerce_project/src/view_products.php

Available Products

Product Name	Category	Price	Stock Quantity	Order Quantity
Laptop	Electronics	1,200.00	10	Order
Smartphone	Electronics	800.00	15	Order
Headphones	Electronics	50.00	30	Order
T-Shirt	Clothing	15.00	50	Order
Jeans	Clothing	40.00	25	Order
Novel	Books	10.00	30	Order
Cookware Set	Home Appliances	80.00	20	Order
Blender	Home Appliances	60.00	10	Order
Action Figure	Toys	25.00	40	Order
Basketball	Sports Equipment	30.00	12	Order

Figure 5.1



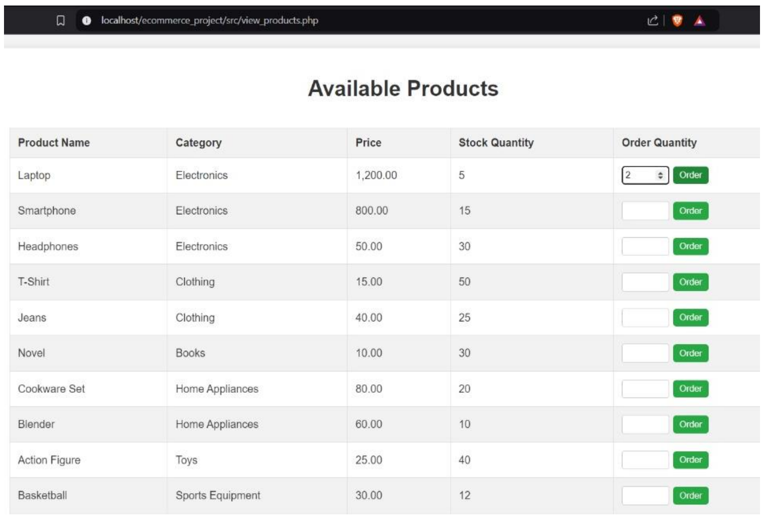
localhost/ecommerce_project/src/view_products.php

Available Products

Product Name	Category	Price	Stock Quantity	Order Quantity
Laptop	Electronics	1,200.00	5	10 Order
Smartphone	Electronics	800.00	15	
Headphones	Electronics	50.00	30	Order
T-Shirt	Clothing	15.00	50	Order
Jeans	Clothing	40.00	25	Order
Novel	Books	10.00	30	Order
Cookware Set	Home Appliances	80.00	20	Order
Blender	Home Appliances	60.00	10	Order
Action Figure	Toys	25.00	40	Order
Basketball	Sports Equipment	30.00	12	Order

Value must be less than or equal to 5.

Figure 5.2

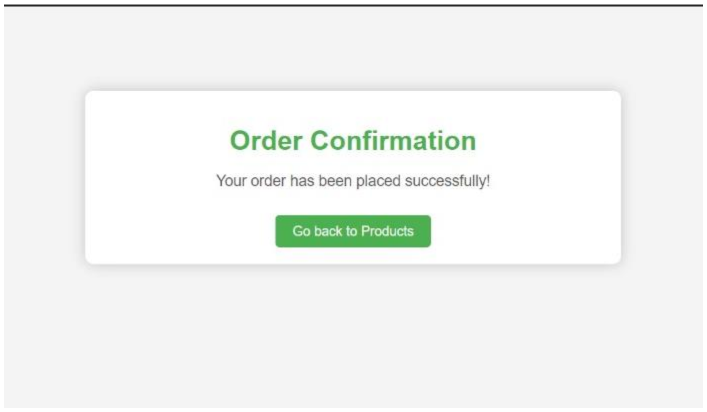


localhost/ecommerce_project/src/view_products.php

Available Products

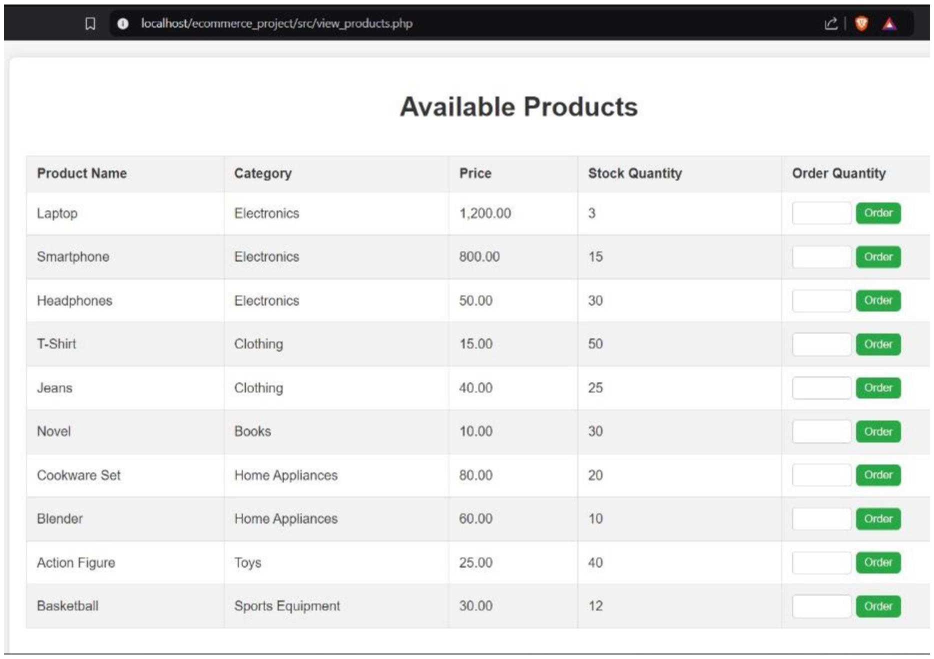
Product Name	Category	Price	Stock Quantity	Order Quantity
Laptop	Electronics	1,200.00	5	2 Order
Smartphone	Electronics	800.00	15	Order
Headphones	Electronics	50.00	30	Order
T-Shirt	Clothing	15.00	50	Order
Jeans	Clothing	40.00	25	Order
Novel	Books	10.00	30	Order
Cookware Set	Home Appliances	80.00	20	Order
Blender	Home Appliances	60.00	10	Order
Action Figure	Toys	25.00	40	Order
Basketball	Sports Equipment	30.00	12	Order

Figure 5.3


Order Confirmation

Your order has been placed successfully!

Figure 5.4



Available Products

Product Name	Category	Price	Stock Quantity	Order Quantity
Laptop	Electronics	1,200.00	3	Order
Smartphone	Electronics	800.00	15	Order
Headphones	Electronics	50.00	30	Order
T-Shirt	Clothing	15.00	50	Order
Jeans	Clothing	40.00	25	Order
Novel	Books	10.00	30	Order
Cookware Set	Home Appliances	80.00	20	Order
Blender	Home Appliances	60.00	10	Order
Action Figure	Toys	25.00	40	Order
Basketball	Sports Equipment	30.00	12	Order

Figure 5.5

6 Conclusion and Future Scope

- The e-commerce inventory system has successfully attained its primary goals, which encompass the management of inventory, automatic order logging, and provision of comprehensive product insights using a stored procedure.
- The custom error handling implemented for out-of-stock scenarios facilitated a seamless user experience, with referential integrity upheld across all database operations.

Future Scope:

User Authentication: Implementing user roles to distinguish between customers and administrators could enhance security and access control.

Order History Feature:

Incorporating user IDs into the Orders table could enhance the system's functionality by allowing customers to access their order history.

Automatic Restocking Alerts:

The introduction of alerts for low stock levels would assist administrators in efficiently managing stock.

Scalability Enhancements:

Expanding the system to support a larger catalog and additional features, such as user and payment information, would enhance its scalability and render it appropriate for real-world applications.

Team members

ANIRUDHA L SARALAYA	**H GANAPATHI KAMATH**	**S SUDHANVA KALKURA**
REG NO 220953663	**REG NO 220953667**	**REG NO 220953016**
ROLL NO 61	**ROLL NO 62**	**ROLL NO 7**
CCE C	**CCE A**	**CCE A**

EVERY TEAM MEMBER HAS CONTRIBUTED IN CODE AS WELL AS IN REPORTS.

References

- **YOUTUBE**
- **GOOGLE, W3C**

Appendices

SQL scripts

1.

```
-- Create the updated trigger to include product_name
DELIMITER //

CREATE TRIGGER after_order_insert
AFTER INSERT ON Orders
FOR EACH ROW
BEGIN
   DECLARE prod_name VARCHAR(255);

   -- Get the product name from the Products table based on the product_id
   SELECT product_name INTO prod_name FROM Products WHERE product_id = NEW.product_id;

   -- Insert into order_logs with product name
   INSERT INTO order_logs (order_id, order_quantity, product_name) VALUES (NEW.order_id, NEW.order_quantity,
prod_name);
END;
//

DELIMITER ;
```

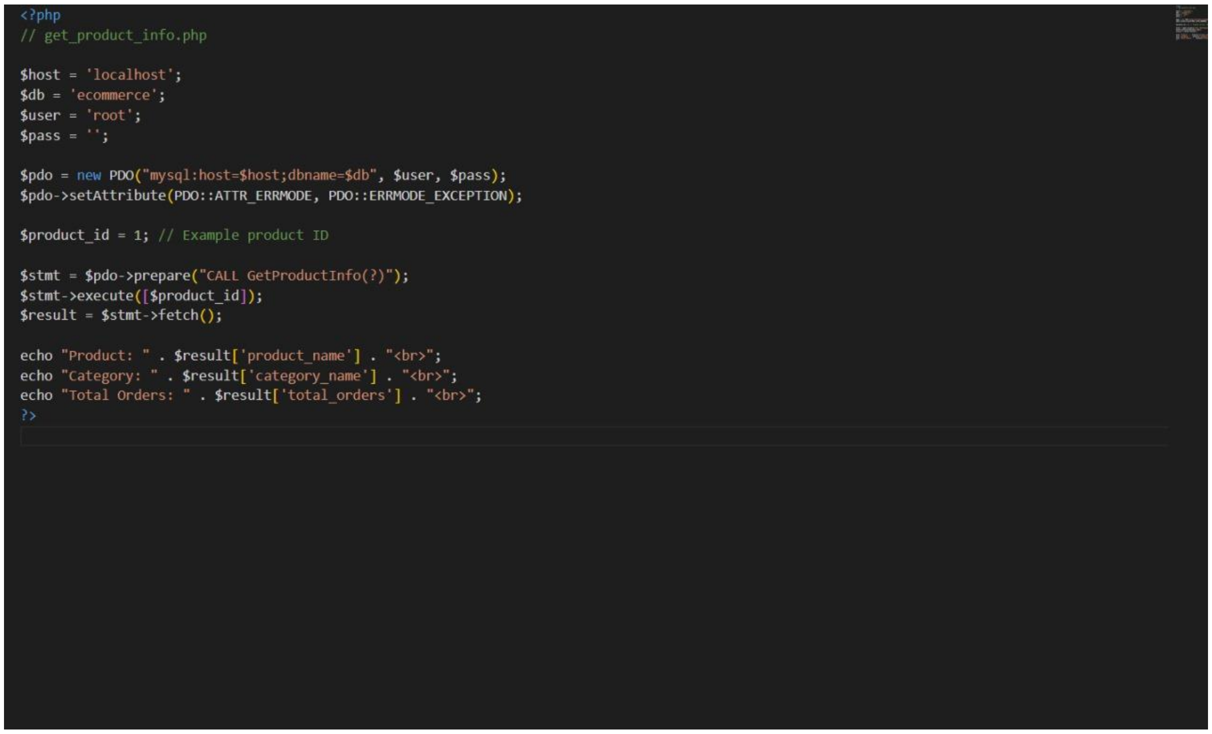
2.

```
DELIMITER //

CREATE TRIGGER after_order_insert
AFTER INSERT ON Orders
FOR EACH ROW
BEGIN
   INSERT INTO order_logs (order_id, order_quantity) VALUES (NEW.order_id, NEW.order_quantity);
END;

//

DELIMITER
```

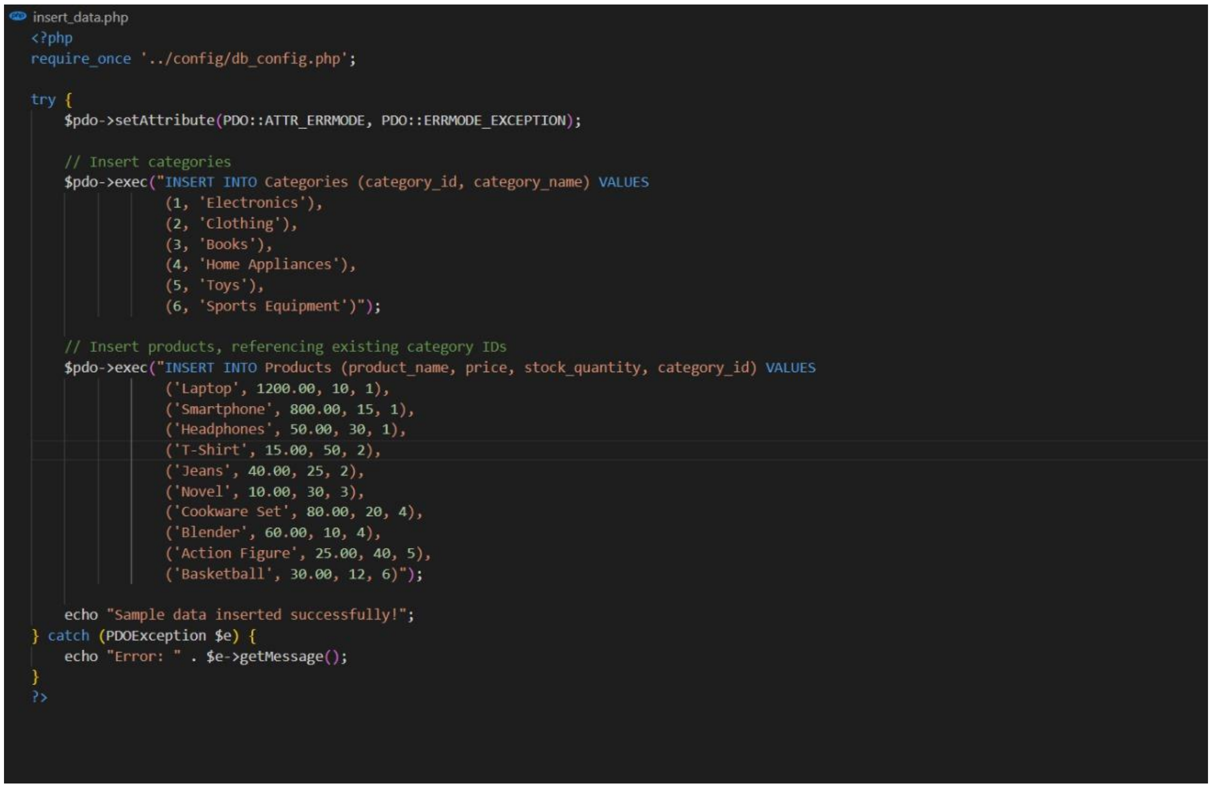
3.

```
PROCEDURES
BEGIN
   SELECT p.product_name, c.category_name, COUNT(o.order_id) AS total_orders
   FROM Products p
   JOIN Categories c ON p.category_id = c.category_id
   LEFT JOIN Orders o ON p.product_id = o.product_id
   WHERE p.product_id = prod_id
   GROUP BY p.product_id;
END;
```

PHP SCRIPTS AND DATABASE INTERACTION

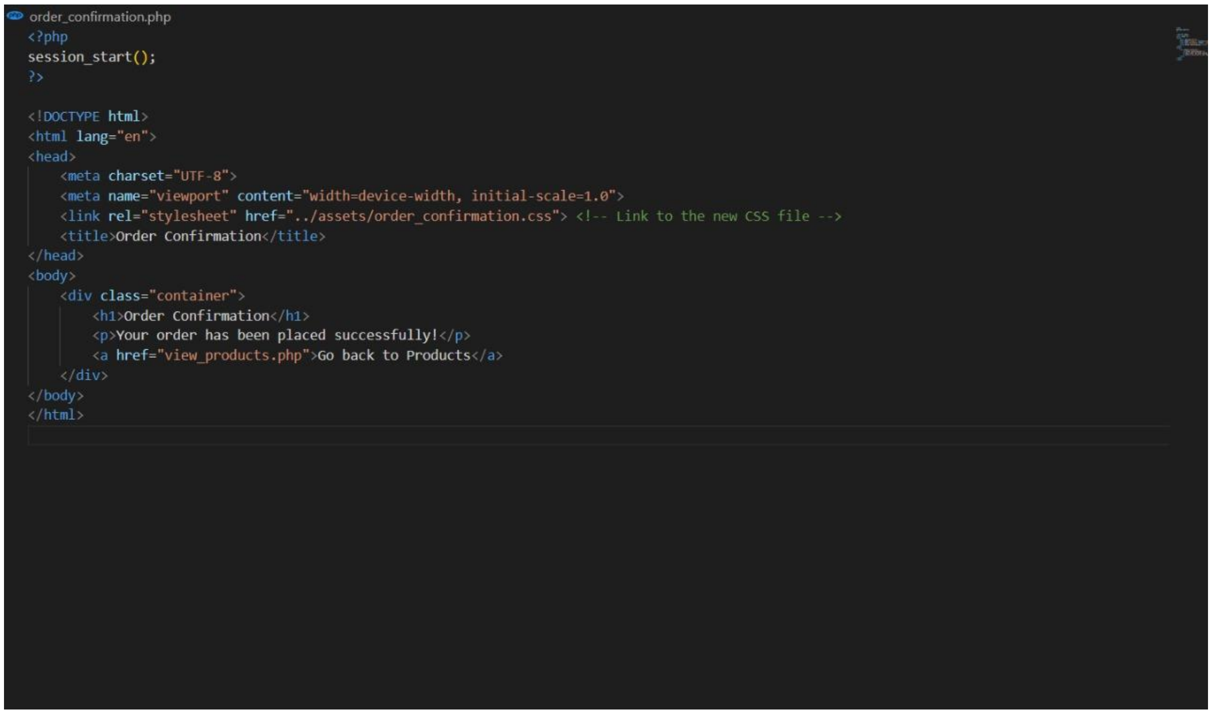


```php
<?php
// get_product_info.php

$host = 'localhost';
$db = 'ecommerce';
$user = 'root';
$pass = '';

$pdo = new PDO("mysql:host=$host;dbname=$db", $user, $pass);
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

$product_id = 1; // Example product ID

$stmt = $pdo->prepare("CALL GetProductInfo(?)");
$stmt->execute([$product_id]);
$result = $stmt->fetch();

echo "Product: " . $result['product_name'] . "<br>";
echo "Category: " . $result['category_name'] . "<br>";
echo "Total Orders: " . $result['total_orders'] . "<br>";
?>
```



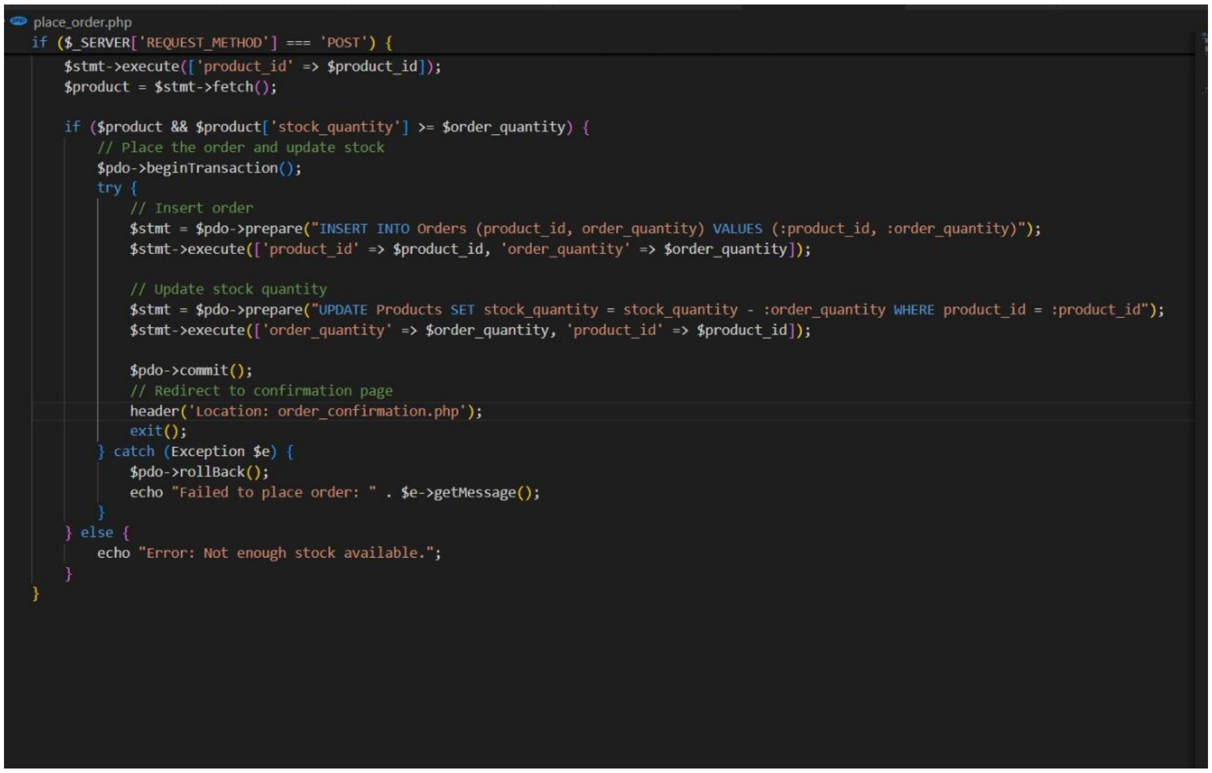
insert_data.php

```php
<?php
require_once '../config/db_config.php';

try {
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // Insert categories
    $pdo->exec("INSERT INTO Categories (category_id, category_name) VALUES
                (1, 'Electronics'),
                (2, 'Clothing'),
                (3, 'Books'),
                (4, 'Home Appliances'),
                (5, 'Toys'),
                (6, 'Sports Equipment')");

    // Insert products, referencing existing category IDs
    $pdo->exec("INSERT INTO Products (product_name, price, stock_quantity, category_id) VALUES
                ('Laptop', 1200.00, 10, 1),
                ('Smartphone', 800.00, 15, 1),
                ('Headphones', 50.00, 30, 1),
                ('T-Shirt', 15.00, 50, 2),
                ('Jeans', 40.00, 25, 2),
                ('Novel', 10.00, 30, 3),
                ('Cookware Set', 80.00, 20, 4),
                ('Blender', 60.00, 10, 4),
                ('Action Figure', 25.00, 40, 5),
                ('Basketball', 30.00, 12, 6)");

    echo "Sample data inserted successfully!";
} catch (PDOException $e) {
    echo "Error: " . $e->getMessage();
}
?>
```



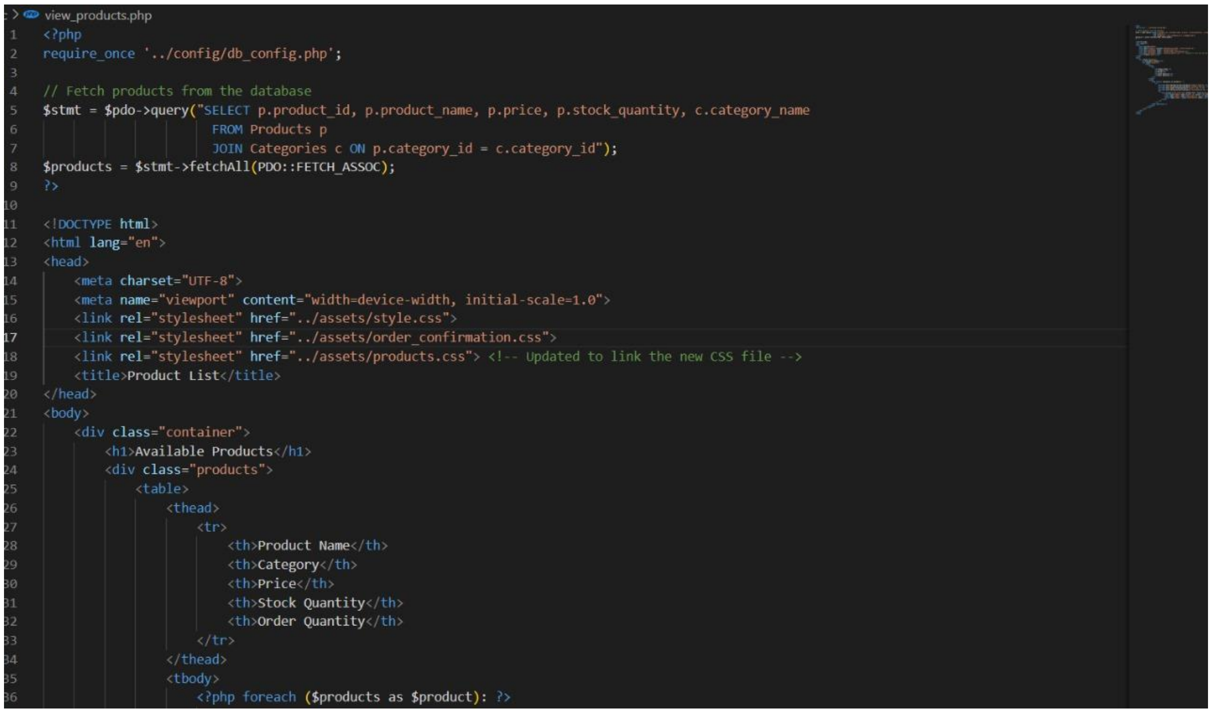
order_confirmation.php

```
<?php
session_start();
?>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="../assets/order_confirmation.css"> <!-- Link to the new CSS file -->
    <title>Order Confirmation</title>
</head>
<body>
    <div class="container">
        <h1>Order Confirmation</h1>
        <p>Your order has been placed successfully!</p>
        <a href="view_products.php">Go back to Products</a>
    </div>
</body>
</html>
```



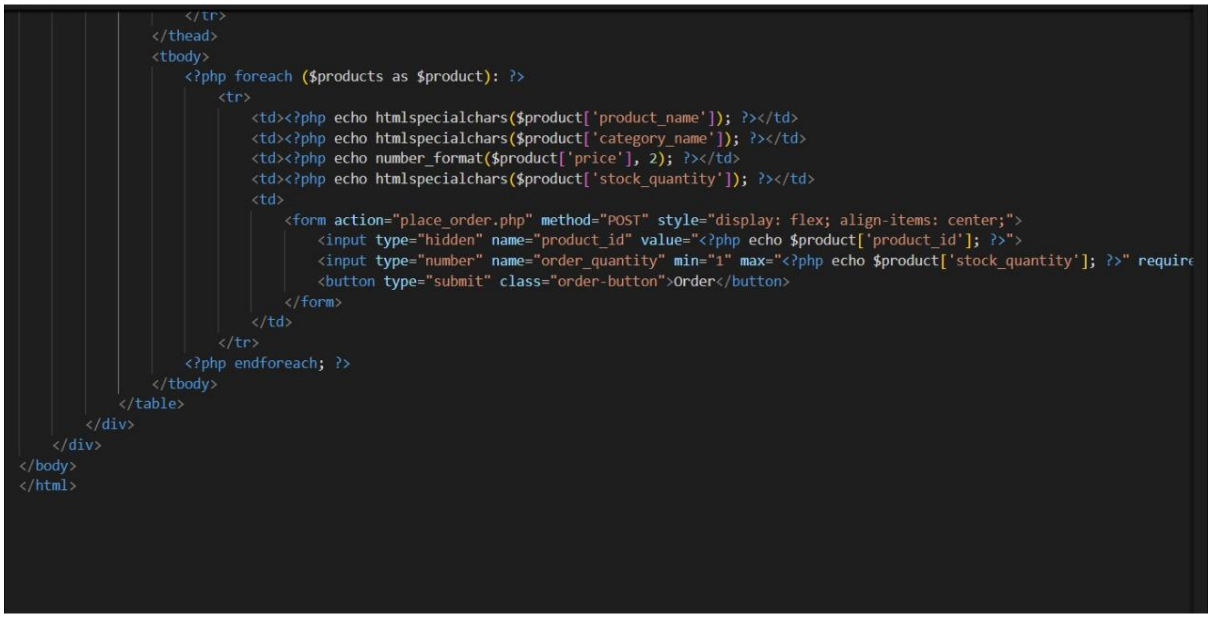
place_order.php

```
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $stmt->execute(['product_id' => $product_id]);
    $product = $stmt->fetch();

    if ($product && $product['stock_quantity'] >= $order_quantity) {
        // Place the order and update stock
        $pdo->beginTransaction();
        try {
            // Insert order
            $stmt = $pdo->prepare("INSERT INTO Orders (product_id, order_quantity) VALUES (:product_id, :order_quantity)");
            $stmt->execute(['product_id' => $product_id, 'order_quantity' => $order_quantity]);

            // Update stock quantity
            $stmt = $pdo->prepare("UPDATE Products SET stock_quantity = stock_quantity - :order_quantity WHERE product_id = :product_id");
            $stmt->execute(['order_quantity' => $order_quantity, 'product_id' => $product_id]);

            $pdo->commit();
            // Redirect to confirmation page
            header('Location: order_confirmation.php');
            exit();
        } catch (Exception $e) {
            $pdo->rollBack();
            echo "Failed to place order: " . $e->getMessage();
        }
    } else {
        echo "Error: Not enough stock available.";
    }
}
```

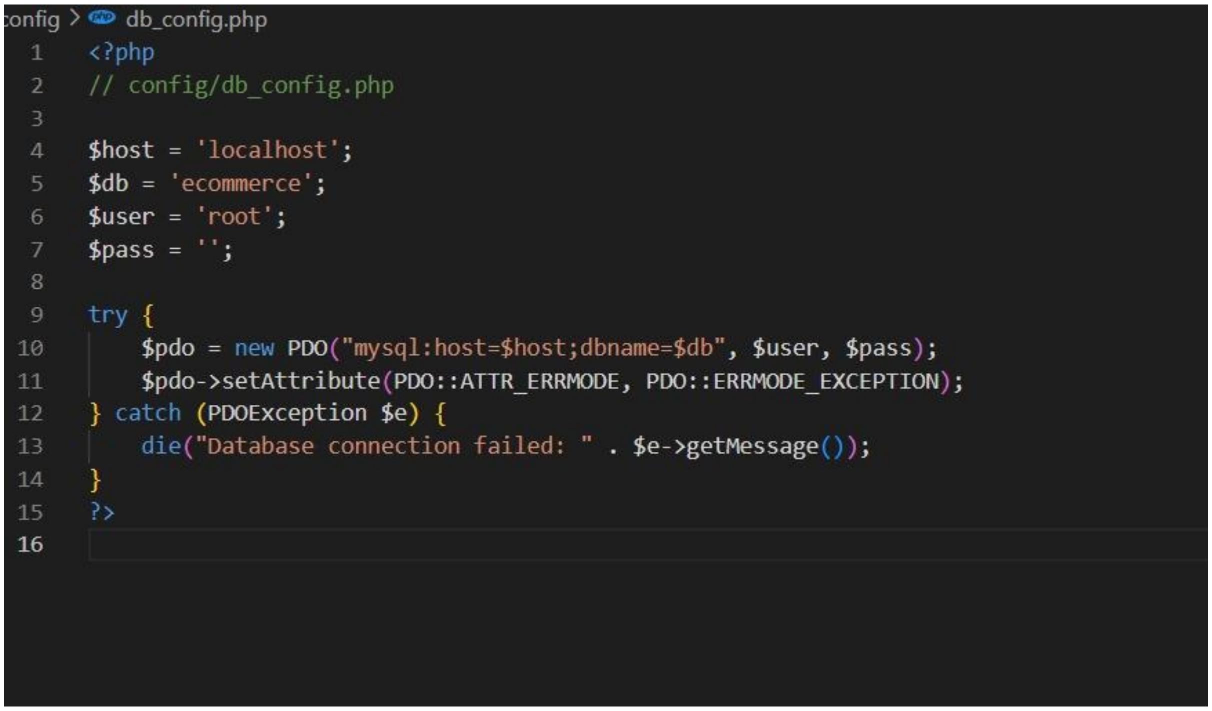


view_products.php

```php
<?php
require_once '../config/db_config.php';

// Fetch products from the database
$stmt = $pdo->query("SELECT p.product_id, p.product_name, p.price, p.stock_quantity, c.category_name
                     FROM Products p
                     JOIN Categories c ON p.category_id = c.category_id");
$products = $stmt->fetchAll(PDO::FETCH_ASSOC);
?>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="../assets/style.css">
    <link rel="stylesheet" href="../assets/order_confirmation.css">
    <link rel="stylesheet" href="../assets/products.css"> <!-- Updated to link the new CSS file -->
    <title>Product List</title>
</head>
<body>
    <div class="container">
        <h1>Available Products</h1>
        <div class="products">
            <table>
                <thead>
                    <tr>
                        <th>Product Name</th>
                        <th>Category</th>
                        <th>Price</th>
                        <th>Stock Quantity</th>
                        <th>Order Quantity</th>
                    </tr>
                </thead>
                <tbody>
                    <?php foreach ($products as $product): ?>
```



```php
                    </tr>
                </thead>
                <tbody>
                    <?php foreach ($products as $product): ?>
                        <tr>
                            <td><?php echo htmlspecialchars($product['product_name']); ?></td>
                            <td><?php echo htmlspecialchars($product['category_name']); ?></td>
                            <td><?php echo number_format($product['price'], 2); ?></td>
                            <td><?php echo htmlspecialchars($product['stock_quantity']); ?></td>
                            <td>
                                <form action="place_order.php" method="POST" style="display: flex; align-items: center;">
                                    <input type="hidden" name="product_id" value="<?php echo $product['product_id']; ?>">
                                    <input type="number" name="order_quantity" min="1" max="<?php echo $product['stock_quantity']; ?>" require
                                    <button type="submit" class="order-button">Order</button>
                                </form>
                            </td>
                        </tr>
                    <?php endforeach; ?>
                </tbody>
            </table>
        </div>
    </div>
</body>
</html>
```



```php
<?php
// config/db_config.php

$host = 'localhost';
$db = 'ecommerce';
$user = 'root';
$pass = '';

try {
    $pdo = new PDO("mysql:host=$host;dbname=$db", $user, $pass);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    die("Database connection failed: " . $e->getMessage());
}
?>

```