North South University
Department of Electrical & Computer Engineering

Project Final Report

Course Code: CSE 327

Course Title: Software Engineering

Course Faculty: Mr. AKM Iqtidar Newaz [IqN]

Project Name:

Digital Tech Hub (Hardware and Electronics Selling Website system.)

Date of Submission: 16/12/2025

Section: 06

Group Number: 04

Submitted To: Mr. AKM Iqtidar Newaz [IqN]

Submitted By

Member 1: Md Rakibul Hasan **ID:** 2212346042

Member 2: Md. Rokib Hasan Oli **ID:** 2211950642

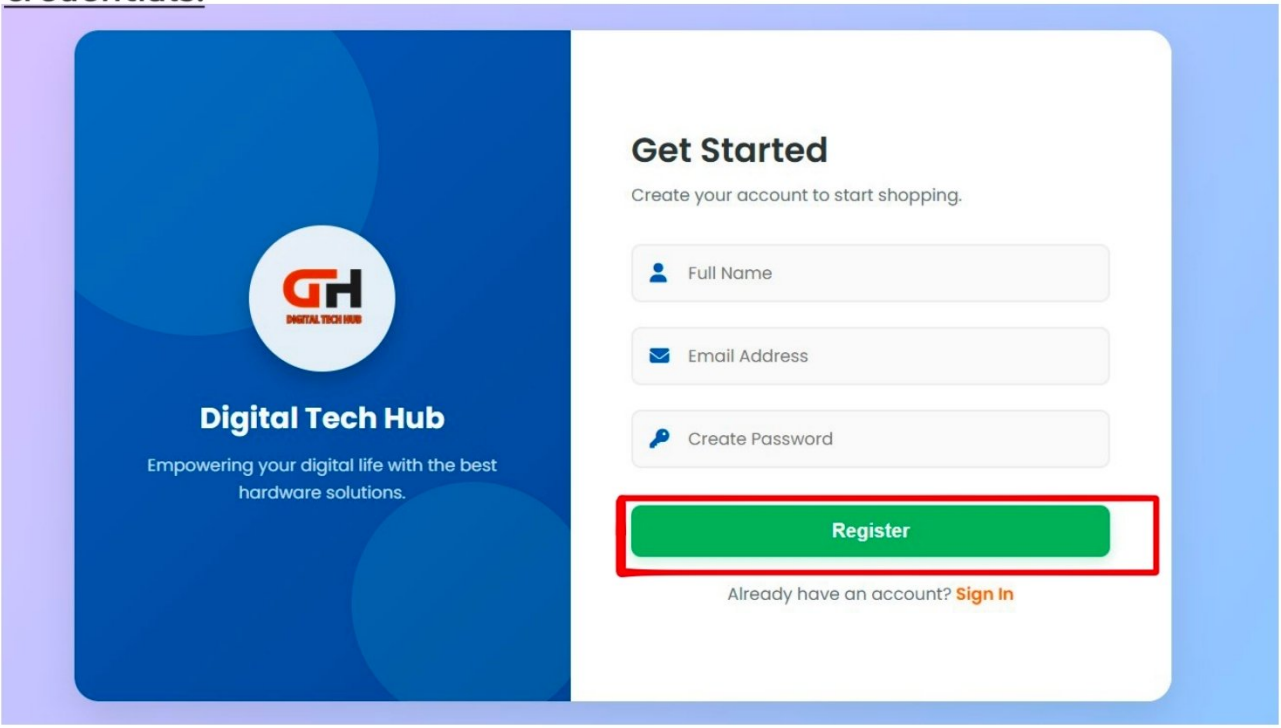
Member 3: Tirtho Mojumdar **ID:** 2312536048

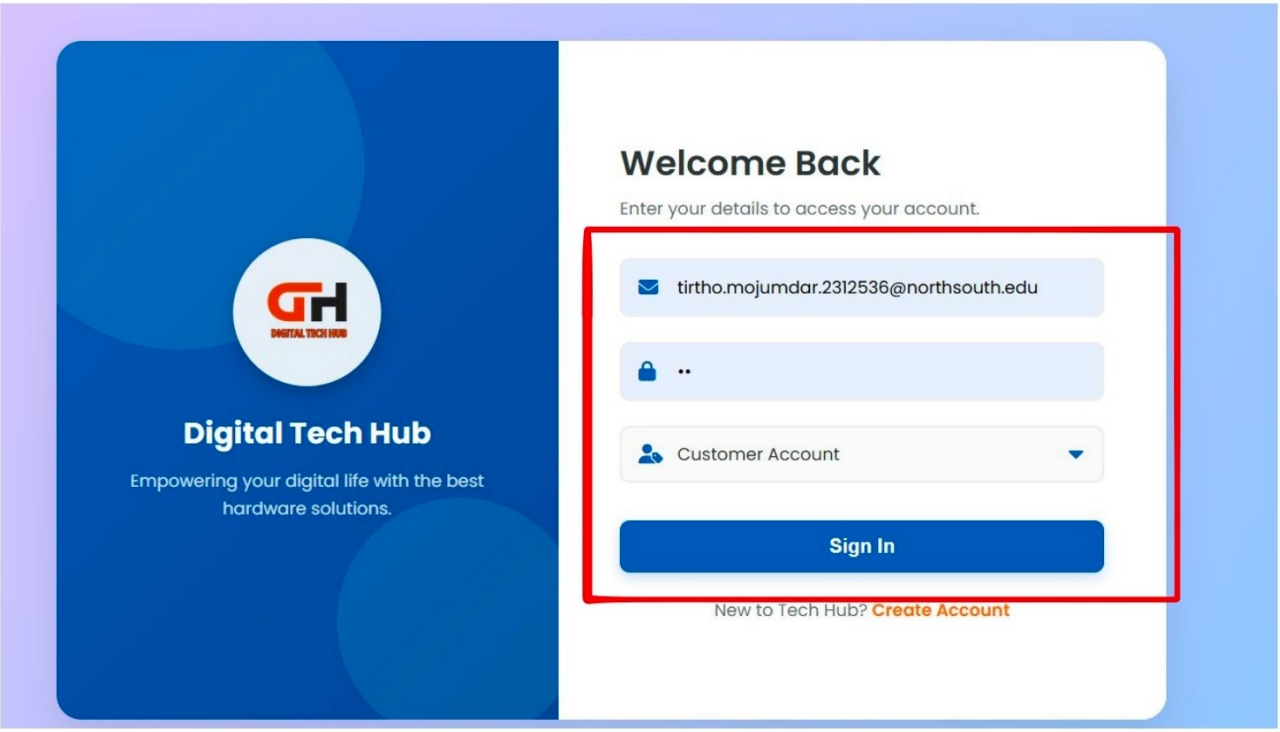
Member 4: Sabbir Ahmod **ID:** 2311501042

Git Repository: https://github.com/Rokib-Hasan-Oli/Digital_Tech_Hub_--CSE327_Sec-6/tree/main

TABLE OF CONTENTS

FRU.0: The system shall allow users to register and log in using credentials.



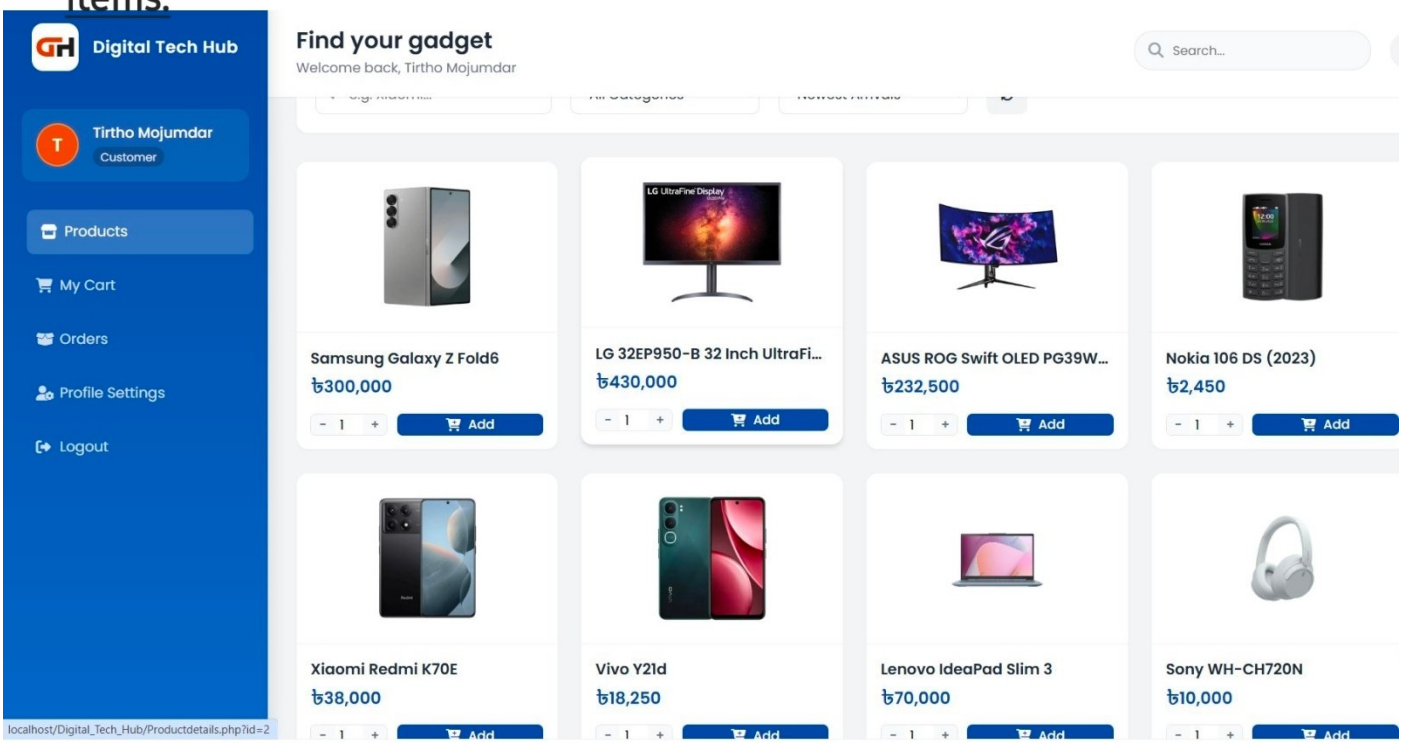


```php
// ========================================
// 1. HANDLE REGISTER (FIXED)
// ========================================
if (isset($_POST['register'])) {
    $name = $db->real_escape_string($_POST['name']);
    $email = $db->real_escape_string($_POST['email']);
    $pass = password_hash($_POST['password'], PASSWORD_DEFAULT);

    // --- FIX START: Check if email already exists ---
    $check_query = "SELECT * FROM users WHERE email = '$email'";
    $check_result = $db->query($check_query);

    if ($check_result->num_rows > 0) {
        // Email found, show red error message
        $msg = "This account already exists on your device";
        $msg_type = "error";
    } else {
        // Email is new, proceed with registration
        $sql = "INSERT INTO users (full_name, email, password) VALUES ('$name', '$email', '$pass')";
        if ($db->query($sql)) {
            $msg = "Registration Successful! Please Login.";
            $msg_type = "success";
        } else {
            $msg = "Error: " . $db->error;
            $msg_type = "error";
        }
    }
    // --- FIX END ---
}
```

```php
if (isset($_POST['login'])) {
    $email = $db->real_escape_string($_POST['email']);
    $pass = $_POST['password'];
    $role = $_POST['role'];

    if ($role == 'admin') {
        $res = $db->query("SELECT * FROM admins WHERE email='$email'");
        $row = $res->fetch_assoc();

        // Check password (supports both plain text sample data & hashed real data)
        if ($row && ($pass === $row['password'] || password_verify($pass, $row['password']))) {
            $_SESSION['role'] = 'admin';
            $_SESSION['user_id'] = $row['admin_id'];
            header("Location: admin_dashboard.php");
            exit();
        } else {
            $msg = "Invalid Admin Credentials";
            $msg_type = "error";
        }
    } else {
        $res = $db->query("SELECT * FROM users WHERE email='$email'");
        $row = $res->fetch_assoc();

        if ($row && password_verify($pass, $row['password'])) {
            if ($row['status'] == 'Blocked') {
                $msg = "Access Denied: Your account is blocked.";
                $msg_type = "error";
            } else {
                $_SESSION['role'] = 'user';
                $_SESSION['user_id'] = $row['user_id'];
                $_SESSION['name'] = $row['full_name'];
                $_SESSION['photo'] = $row['profile_image'];
                header("Location: user_dashboard.php");
                exit();
            }
```

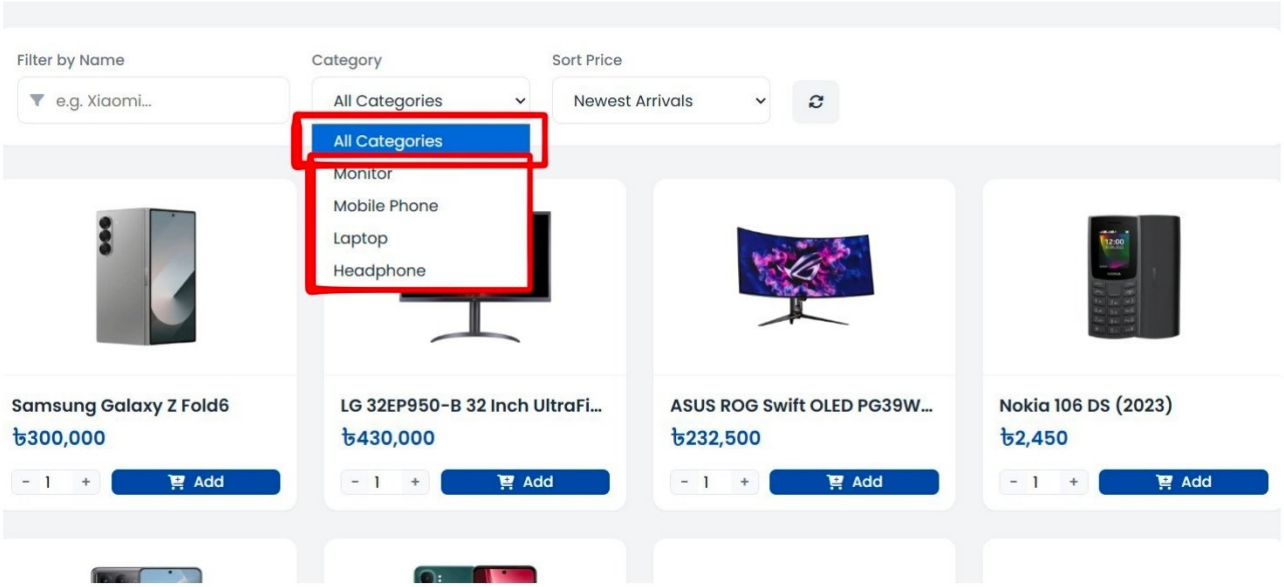
```
81              }
82          } else {
83              $msg = "Invalid Email or Password";
84              $msg_type = "error";
85          }
86      }
87  }
88  ?>
89
```

FRU.1: The system shall allow users to view all Hardware and Electronics items.



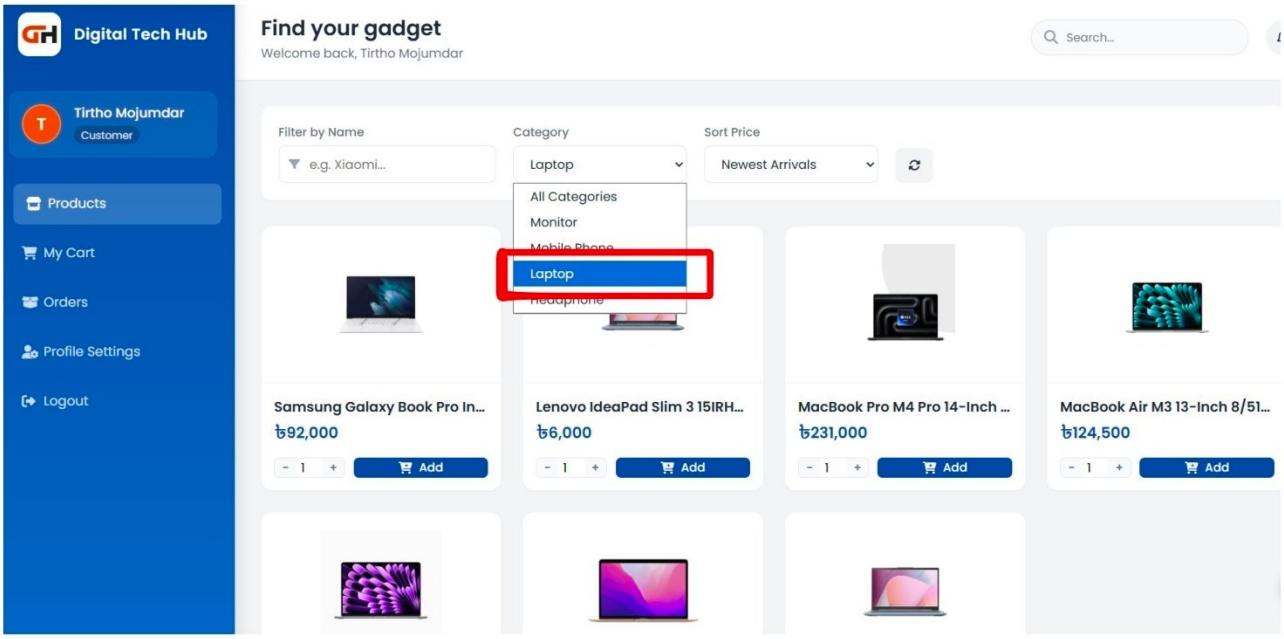
```php
if (!isset($_GET['view']) || $_GET['view'] == 'products') {
    $cat_query = "SELECT * FROM categories";
    $categories = $db->query($cat_query);

    // Filter System
    $conditions = [];
    $order_sql = "";

```

FRU.2: The system shall allow users to view all product categories



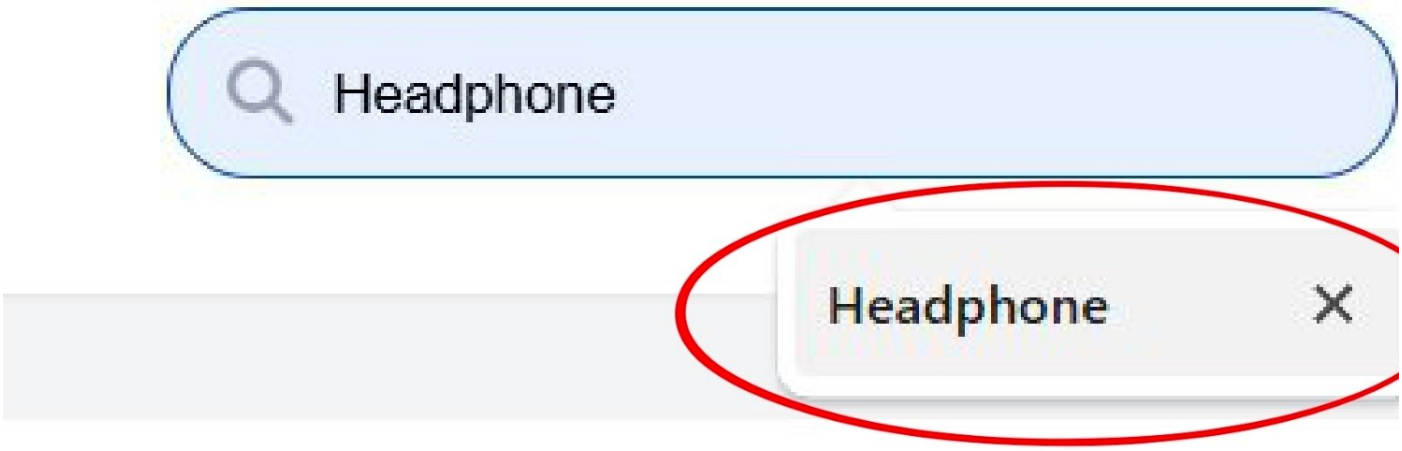
```php
$cat_query = "SELECT * FROM categories";
$categories = $db->query($cat_query);
```

FRU.3: The system shall allow users to view products filtered by category



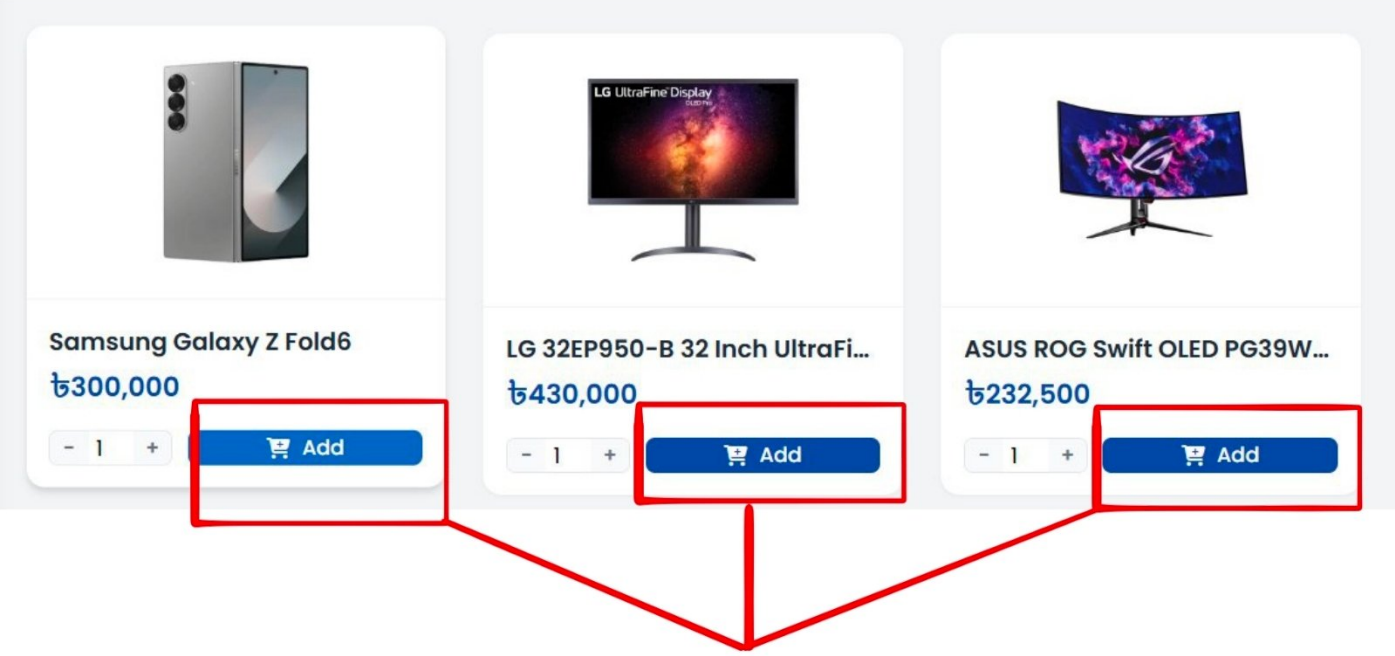
```php
44
45  if (!empty($_GET['category'])) {
46      $cat_id = (int)$_GET['category'];
47      $conditions[] = "category_id = '$cat_id'";
48  }
```

FRU.4: The system shall support a search function to find products



```php
// Filter System
$conditions = [];
$order_sql = "";

if (!empty($_GET['search'])) {
    $s = $db->real_escape_string($_GET['search']);
    $conditions[] = "name LIKE '%$s%'";
}
```

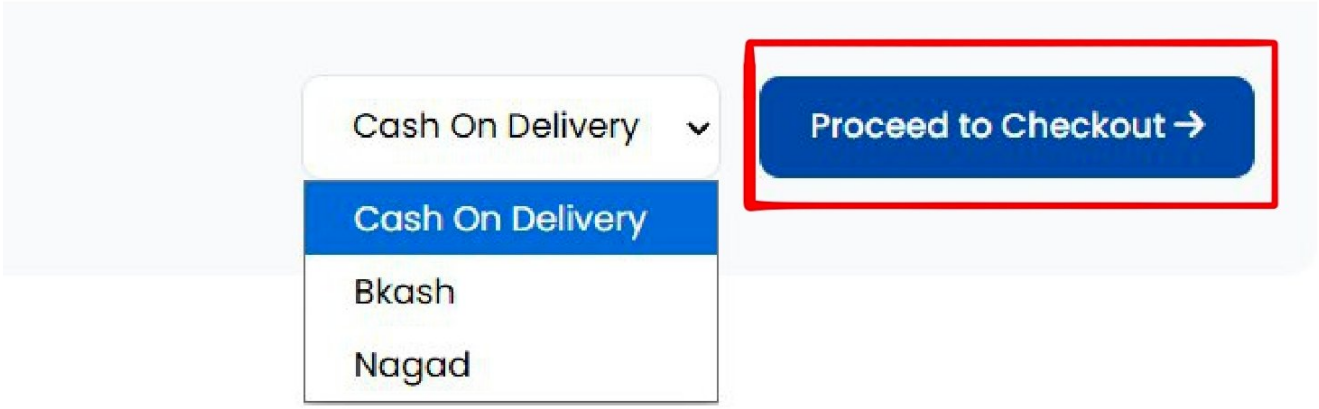
FRU.5: The system shall allow users to add items to a shopping cart and view the cart contents.



```php
if (isset($_POST['add_cart'])) {
    if (!isset($_SESSION['cart'])) {
        $_SESSION['cart'] = [];
    }
    $_SESSION['cart'][] = [
        'id'    => $_POST['pid'],
        'name'  => $_POST['pname'],
        'price' => $_POST['pprice'],
        'qty'   => $_POST['qty']
    ];
    echo "<script>alert('Added to Cart');</script>";
}
```

```php
if (isset($_POST['checkout']) && isset($_SESSION['cart'])) {

    $facade = new OrderFacade();

    $uid    = $_SESSION['user_id'];
    $amount = $_POST['pay_amount'];
    $method = $_POST['method'];
    $items  = $_SESSION['cart'];

    // Place order
    $orderId = $facade->placeOrder($uid, $amount, $method, $items);
    unset($_SESSION['cart']);

    // ✅ FIX: update order & payment status after successful payment
    if (strtolower($method) !== 'cod') {
        $db->query("
            UPDATE orders
            SET payment_status = 'Paid',
                order_status = 'Processing'
            WHERE order_id = $orderId
        ");
    }

    $products = $db->query("
        SELECT p.name, oi.quantity, oi.unit_price
        FROM order_items oi
        JOIN products p ON p.product_id = oi.product_id
        WHERE oi.order_id = $orderId
    ");
```

FRU.6: The system shall facilitate a checkout process.

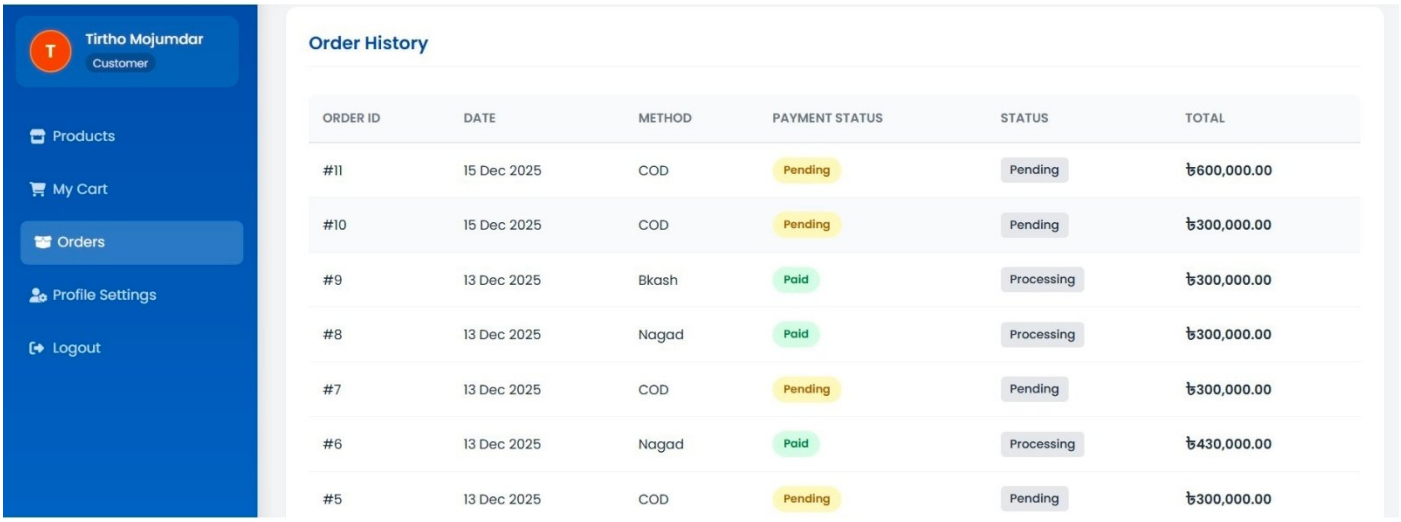



```php
if (isset($_POST['checkout']) && isset($_SESSION['cart'])) {

    $facade = new OrderFacade();

    $uid    = $_SESSION['user_id'];
    $amount = $_POST['pay_amount'];
    $method = $_POST['method'];
    $items  = $_SESSION['cart'];

    // Place order
    $orderId = $facade->placeOrder($uid, $amount, $method, $items);
    unset($_SESSION['cart']);

    // ✅ FIX: update order & payment status after successful payment
    if (strtolower($method) !== 'cod') {
        $db->query("
            UPDATE orders
            SET payment_status = 'Paid',
                order_status = 'Processing'
            WHERE order_id = $orderId
        ");
    }

    $products = $db->query("
        SELECT p.name, oi.quantity, oi.unit_price
        FROM order_items oi
        JOIN products p ON p.product_id = oi.product_id
        WHERE oi.order_id = $orderId
    ");

    $badge = strtolower($method);
?>
```

FRU.7: The system shall offer online payment methods (e.g., Bkash, Nagad).

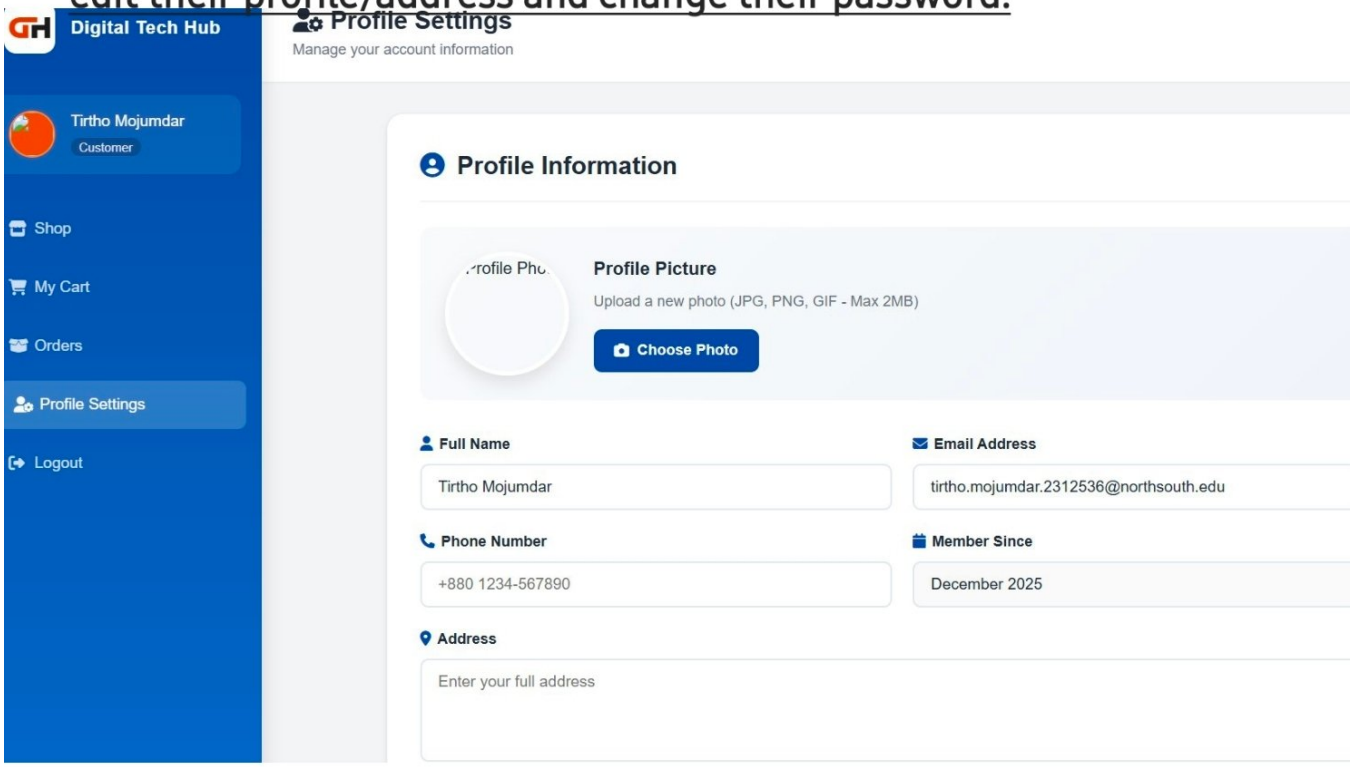
```php
<?php
require_once 'Database.php';
interface IPayment {
    public function pay($amount);
}

class BkashPayment implements IPayment {
    public function pay($amount) { return "Paid $amount Tk via Bkash (Trans ID: BK-" . rand(1000,9999) . ")"; }
}

class NagadPayment implements IPayment {
    public function pay($amount) { return "Paid $amount Tk via Nagad (Trans ID: NG-" . rand(1000,9999) . ")"; }
}

class CODPayment implements IPayment {
    public function pay($amount) { return "Amount $amount Tk to be paid on Delivery."; }
}

class PaymentFactory {
    public static function create($method) {
        if ($method == 'Bkash') return new BkashPayment();
        if ($method == 'Nagad') return new NagadPayment();
        return new CODPayment(); // Default
    }
}
?>
```

FRU.8: The system shall allow users to view their order history and the current status of each order.




```
144                 <a href="?view=orders" class="<?php echo (isset($_GET['view']) && $_GET['view'] == 'orders') ? 'active' : ''; ?>">
145                     <i class="fa fa-box-open"></i> Orders
146                 </a>
147             </li>
```

FRU.9: The system shall provide Profile Management allowing users to edit their profile/address and change their password.

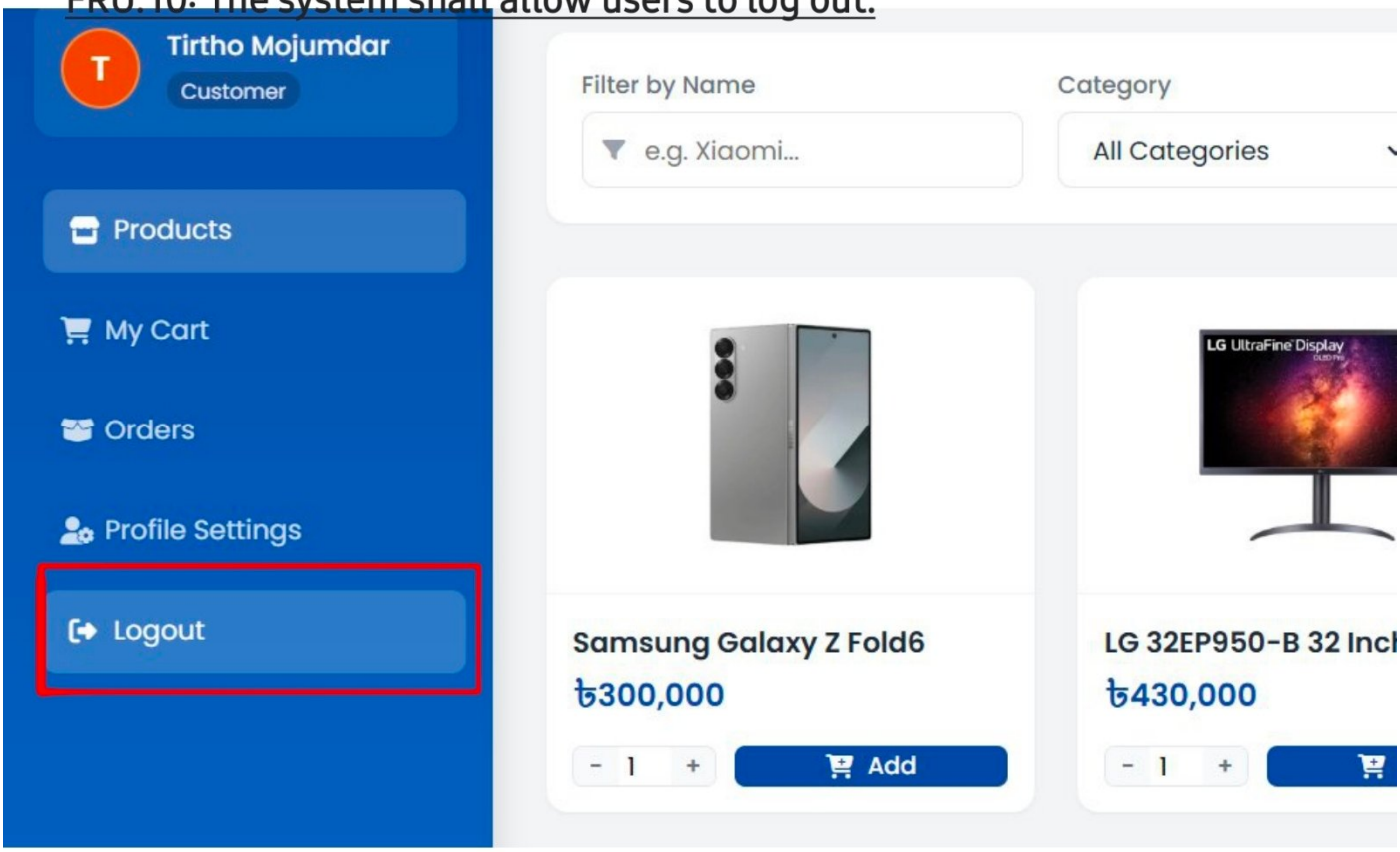



```php
<?php
session_start();
require_once 'Database.php';

// Auth Check
if (!isset($_SESSION['role']) || $_SESSION['role'] !== 'user') {
    header("Location: login_register.php");
    exit();
}

$db = Database::getInstance()->getConnection();
$logo = "https://ln.run/WlP6-";
$user_id = $_SESSION['user_id'];
$success_msg = "";
$error_msg = "";

// Fetch current user data
$user_sql = "SELECT * FROM users WHERE user_id = $user_id";
$user_result = $db->query($user_sql);
$user_data = $user_result->fetch_assoc();

// Handle Profile Update
if (isset($_POST['update_profile'])) {
    $full_name = $db->real_escape_string($_POST['full_name']);
    $email = $db->real_escape_string($_POST['email']);
    $phone = $db->real_escape_string($_POST['phone'] ?? '');
    $address = $db->real_escape_string($_POST['address'] ?? '');

    // Handle profile photo upload
    $profile_image = $user_data['profile_image'] ?? 'default-avatar.png';

    if (isset($_FILES['profile_image']) && $_FILES['profile_image']['error'] == 0) {
        $upload_dir = "uploads/profiles/";
        if (!file_exists($upload_dir)) {
            mkdir($upload_dir, 0777, true);
        }

```

```php
        $file_ext = strtolower(pathinfo($_FILES['profile_image']['name'], PATHINFO_EXTENSION));
        $allowed_ext = ['jpg', 'jpeg', 'png', 'gif'];

        if (in_array($file_ext, $allowed_ext)) {
            $new_filename = "user_" . $user_id . "_" . time() . "." . $file_ext;
            $target_file = $upload_dir . $new_filename;

            if (move_uploaded_file($_FILES['profile_image']['tmp_name'], $target_file)) {
                $profile_image = $new_filename;
            }
        }
    }

    // Update database
    $update_sql = "UPDATE users SET
                   full_name = '$full_name',
                   email = '$email',
                   phone = '$phone',
                   address = '$address',
                   profile_image = '$profile_image'
                   WHERE user_id = $user_id";

    if ($db->query($update_sql)) {
        $_SESSION['name'] = $full_name;
        $_SESSION['email'] = $email;
        $success_msg = "Profile updated successfully!";

        // Refresh user data
        $user_result = $db->query($user_sql);
        $user_data = $user_result->fetch_assoc();
    } else {
        $error_msg = "Failed to update profile.";
    }
}
```

```php
// Handle Password Change
if (isset($_POST['change_password'])) {
    $current_password = $_POST['current_password'];
    $new_password = $_POST['new_password'];
    $confirm_password = $_POST['confirm_password'];

    // FIX 1: Use password_verify() to check the hash, not direct comparison
    if (password_verify($current_password, $user_data['password'])) {

        if ($new_password == $confirm_password) {
            if (strlen($new_password) >= 6) {

                // FIX 2: Hash the NEW password before saving it to the database
                // If you don't do this, you won't be able to login next time!
                $hashed_password = password_hash($new_password, PASSWORD_DEFAULT);

                $update_pass = "UPDATE users SET password = '$hashed_password' WHERE user_id = $user_id";

                if ($db->query($update_pass)) {
                    $success_msg = "Password changed successfully!";

                    // Optional: Update the local user_data so consecutive changes don't fail immediately
                    $user_data['password'] = $hashed_password;
                } else {
                    $error_msg = "Failed to change password.";
                }
            } else {
                $error_msg = "Password must be at least 6 characters.";
            }
        } else {
            $error_msg = "New passwords do not match.";
        }
    } else {
        $error_msg = "Current password is incorrect.";
    }
}
?>
```

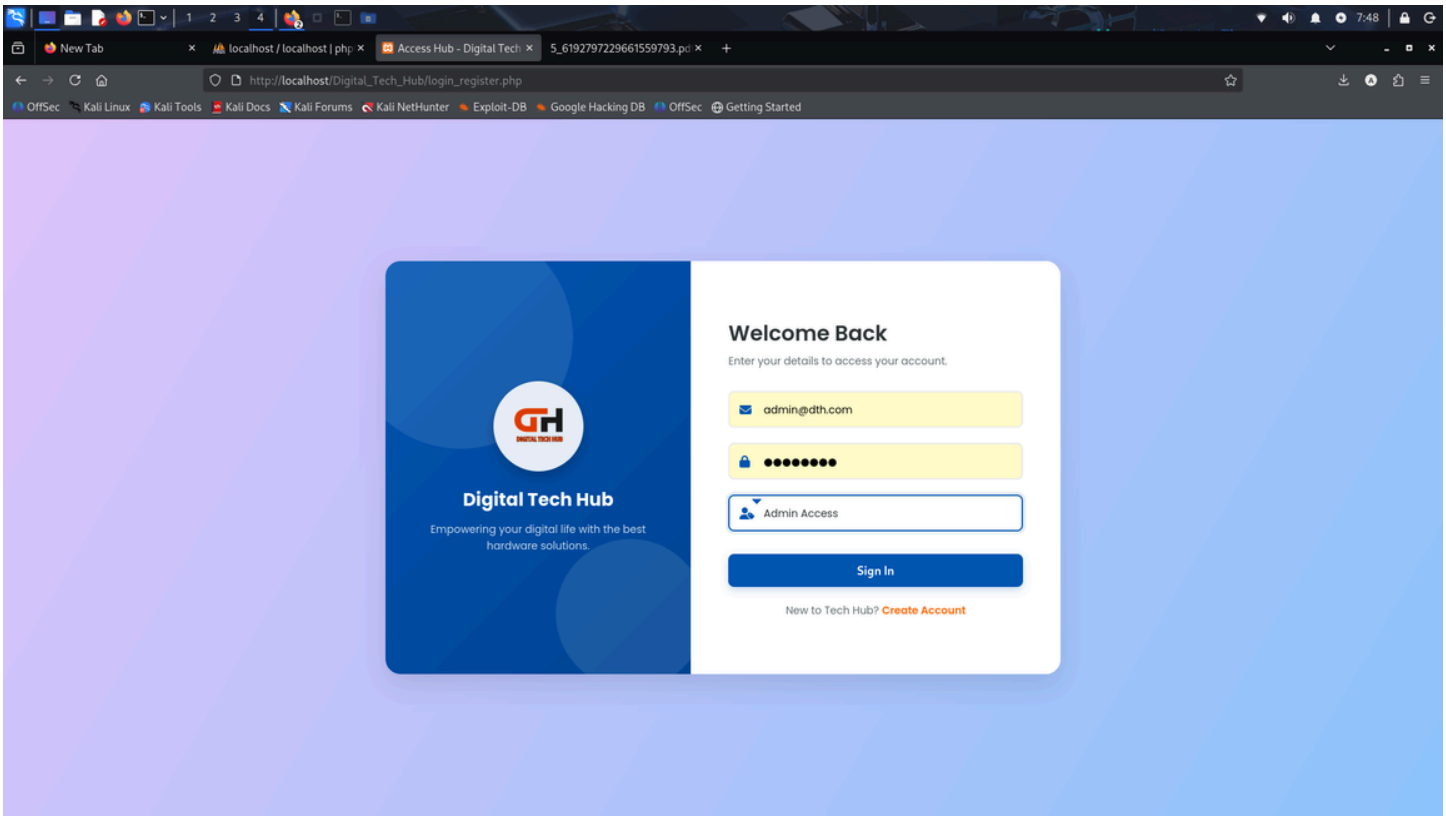
FRU 10: The system shall allow users to log out.

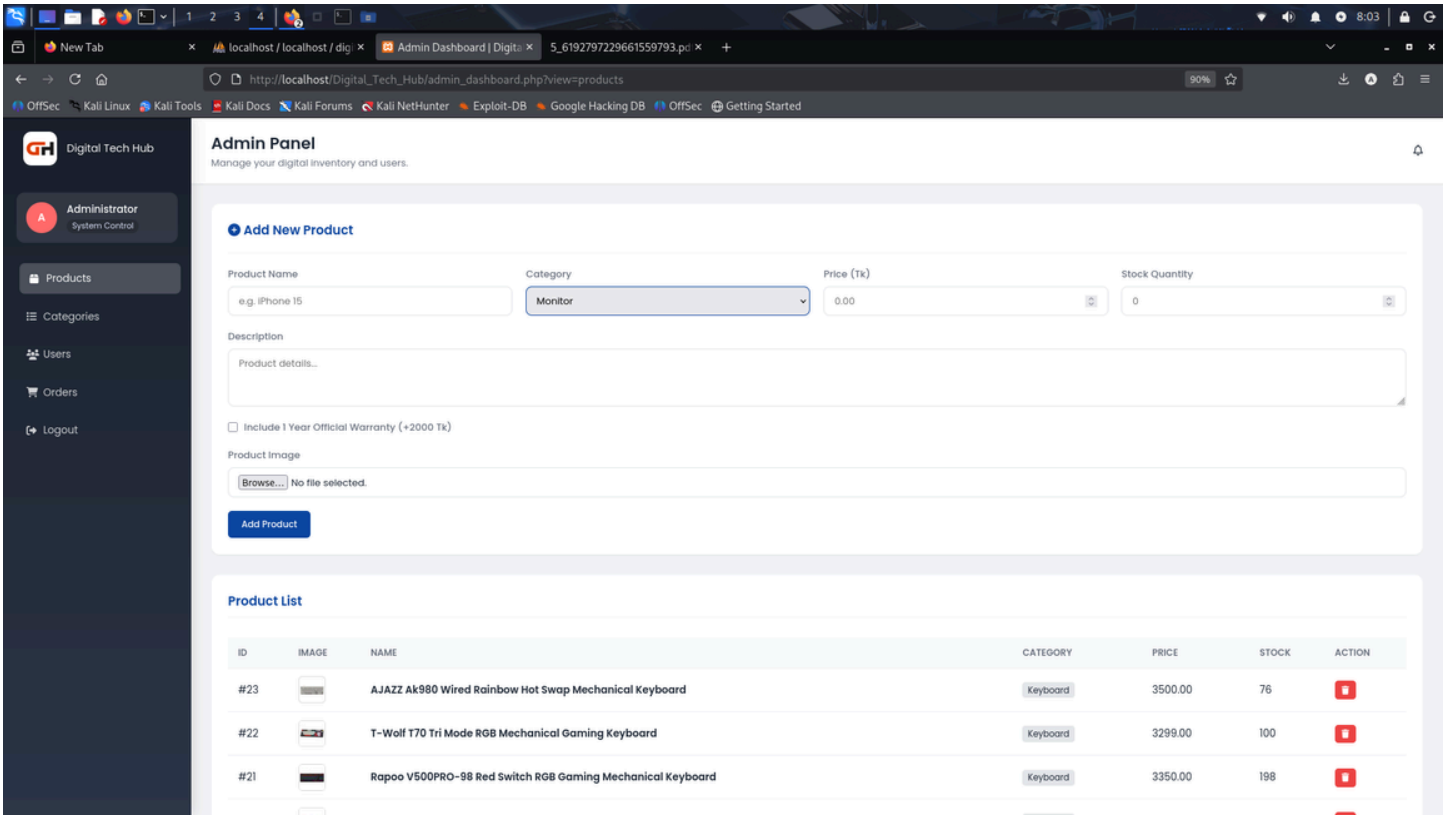


```php
<?php
session_start();
session_destroy();
header("Location: login_register.php");
?>
```

Admin Functionalities (Owner-Facing)

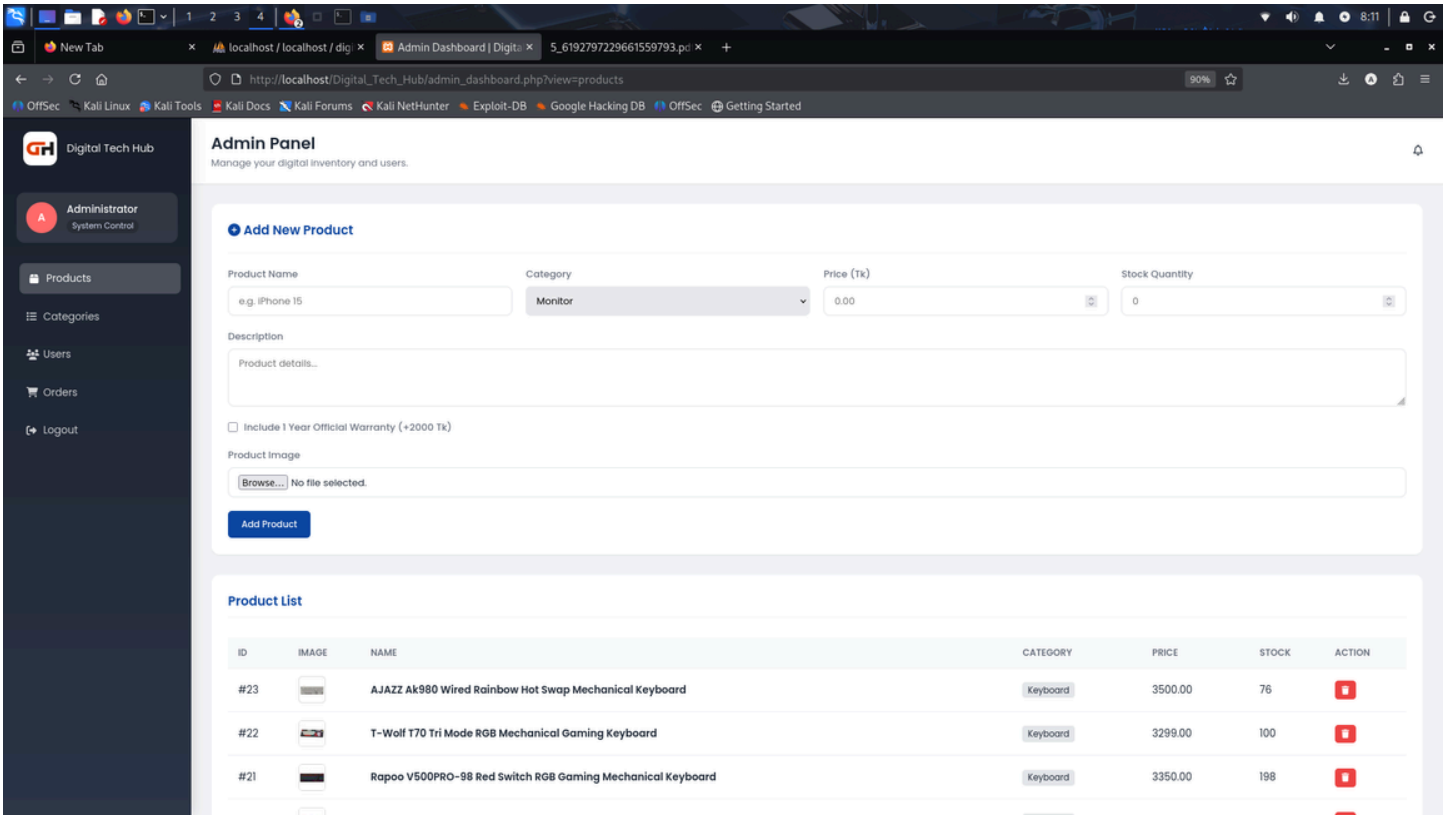
- FRA.0: The system shall allow the Admin to log in





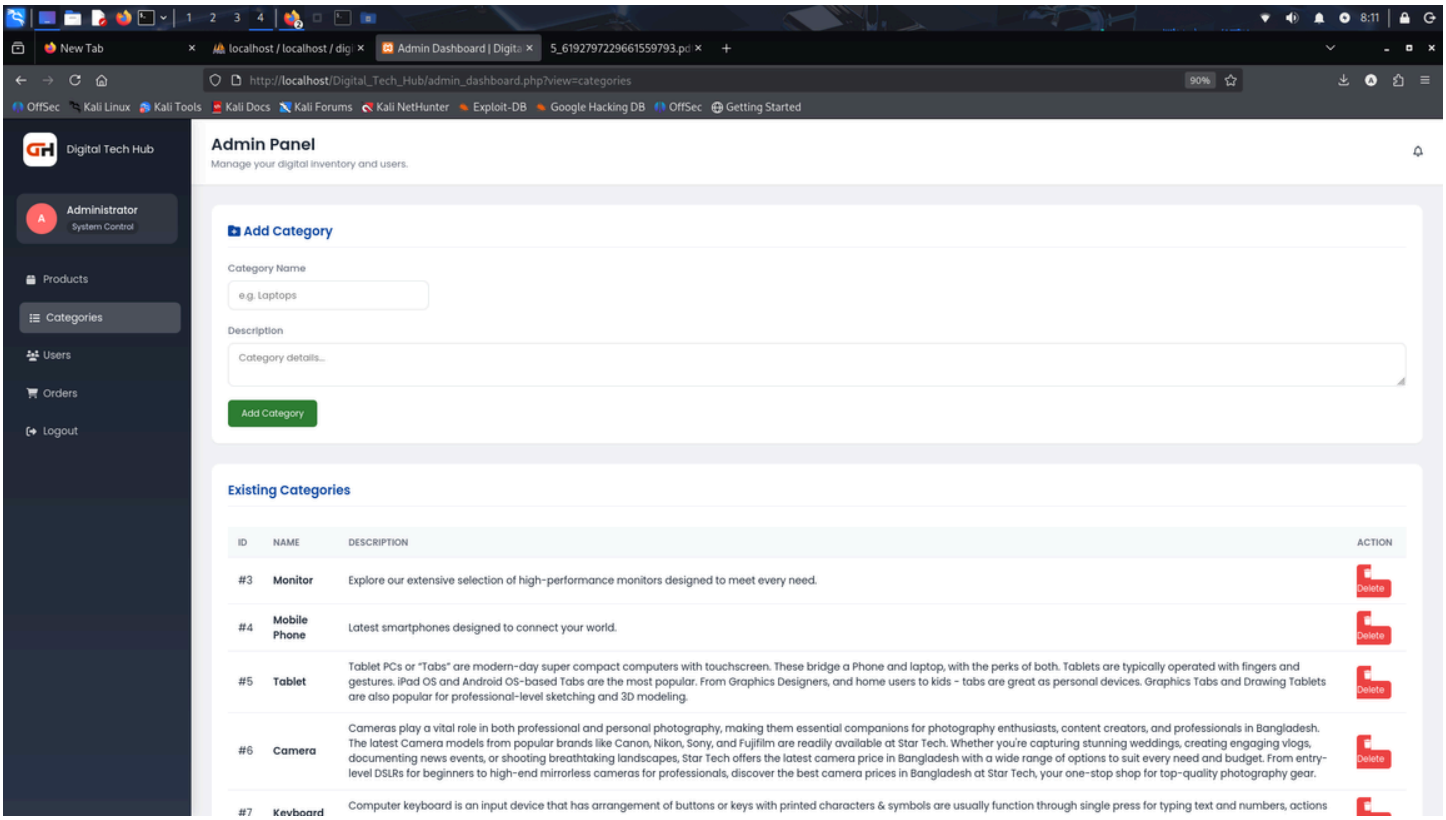

```php
<?php
session_start();
require_once 'AdminProxy.php';

// 1. PROXY PATTERN: Security Check
$proxy = new AdminProxy();
$proxy->render();
```

- FRA.1: The system shall allow the Admin to Add, Edit, and Delete Hardware and Electronics categories.



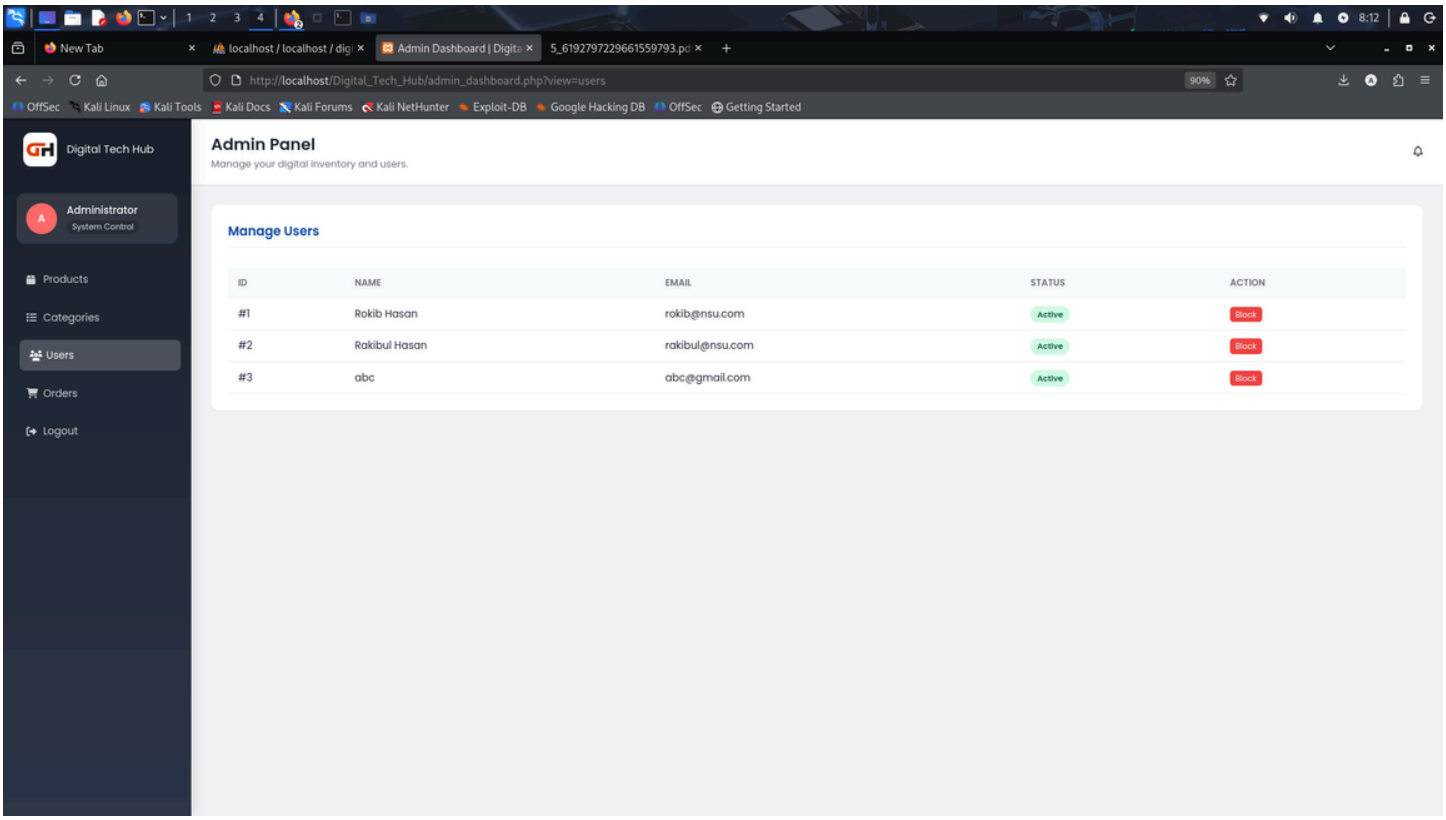

```php
if (isset($_POST['add_cat'])) {
    $name = $db->real_escape_string($_POST['cat_name']);
    $desc = $db->real_escape_string($_POST['cat_desc']);
    if ($db->query("INSERT INTO categories (name, description) VALUES ('$name', '$desc')")) {
        $msg = "Category Added Successfully.";
    } else {
        $msg = "Error: " . $db->error;
    }
}
if (isset($_GET['delete_cat'])) {
    $id = intval($_GET['delete_cat']);
    $db->query("DELETE FROM categories WHERE category_id=$id");
    $msg = "Category Deleted.";
}
```

- FRA.2: The system shall allow the Admin to Add, Edit, and Delete Hardware and Electronics items (inventory management).



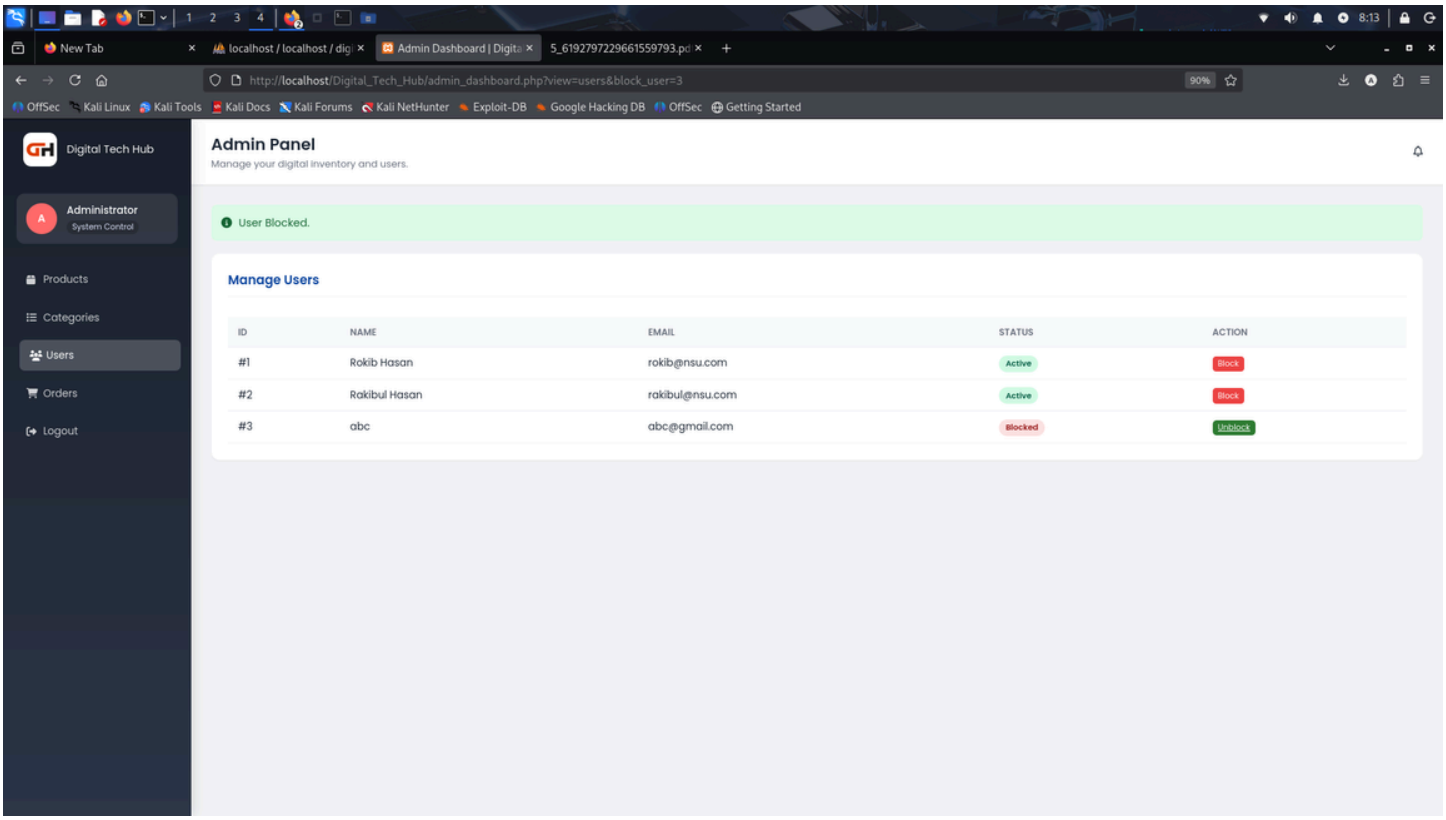

```php
if (isset($_POST['add_product'])) {
    $name = $db->real_escape_string($_POST['name']);
    $desc = $db->real_escape_string($_POST['description']);
    $price = $_POST['price'];
    $stock = $_POST['stock'];
    $cat_id = $_POST['category_id'];
    $has_warranty = isset($_POST['has_warranty']) ? 1 : 0;
    // Image Upload Handling
    $image = "default_product.png";
    if (isset($_FILES['product_image']['name']) && $_FILES['product_image']['name'] != "") {
        $target_dir = "uploads/";
        if (!file_exists($target_dir)) { mkdir($target_dir, 0777, true); }

        $image = basename($_FILES["product_image"]["name"]);
        $target_file = $target_dir . $image;
        move_uploaded_file($_FILES["product_image"]["tmp_name"], $target_file);
    }

    $sql = "INSERT INTO products (category_id, name, description, price, stock_quantity, product_image, has_warranty)
            VALUES ('$cat_id', '$name', '$desc', '$price', '$stock', '$image', '$has_warranty')";

    if ($db->query($sql)) {
        $msg = "Product Added Successfully.";
    } else {
        $msg = "Error: " . $db->error;
    }
}
if (isset($_GET['delete_prod'])) {
    $id = intval($_GET['delete_prod']);
    $db->query("DELETE FROM products WHERE product_id=$id");
    $msg = "Product Deleted.";
}
```

- FRA.3: The system shall allow the Admin to view a list of all registered users.



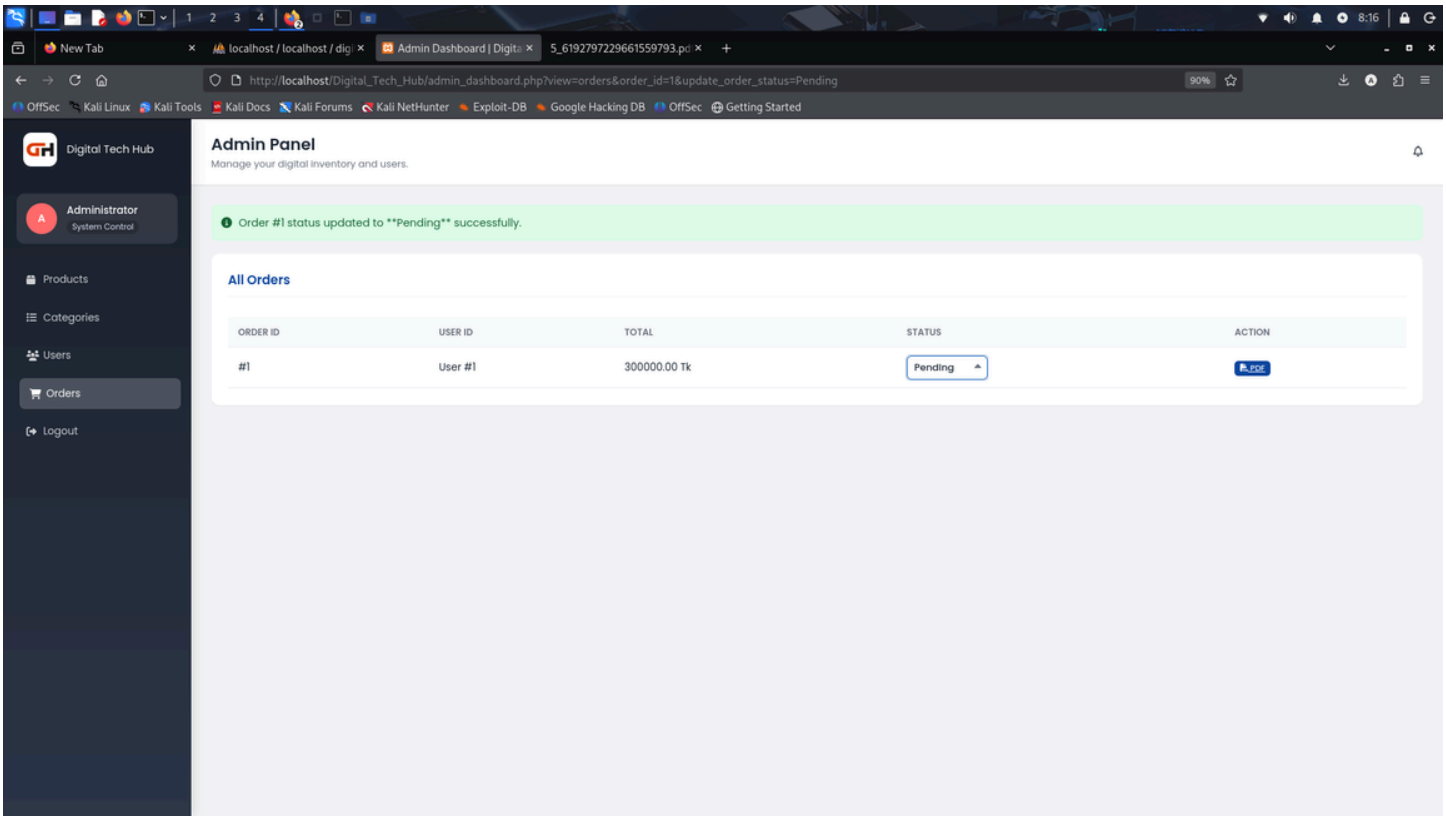

```php
<?php if (isset($_GET['view']) && $_GET['view'] == 'users') { ?>
    <div class="panel-section">
        <h3>Manage Users</h3>
        <div class="table-responsive">
            <table class="data-table">
                <thead>
                    <tr><th>ID</th><th>Name</th><th>Email</th><th>Status</th><th>Action</th></tr>
                </thead>
                <tbody>
                    <?php
                    $res = $db->query("SELECT * FROM users");
                    while ($row = $res->fetch_assoc()) {
                        $statusClass = ($row['status'] == 'Blocked') ? 'status-blocked' : 'status-active';
                        echo "<tr>
                            <td>#{$row['user_id']}</td>
                            <td>{$row['full_name']}</td>
                            <td>{$row['email']}</td>
                            <td><span class='status-badge $statusClass'>{$row['status']}</span></td>
                            <td>";
                            // UPDATED LOGIC: Display Block or Unblock button
                            if($row['status'] == 'Blocked') {
                                echo "<a href='?view=users&unblock_user={$row['user_id']}' class='btn-success btn-sm'>Unblock</a>";
                            } else {
                                echo "<a href='?view=users&block_user={$row['user_id']}' class='btn-danger btn-sm'>Block</a>";
                            }
                        echo "</td></tr>";
                    }
                    ?>
                </tbody>
            </table>
        </div>
    </div>
<?php } ?>
```

- FRA.4: The system shall provide the ability for the Admin to block specific users.



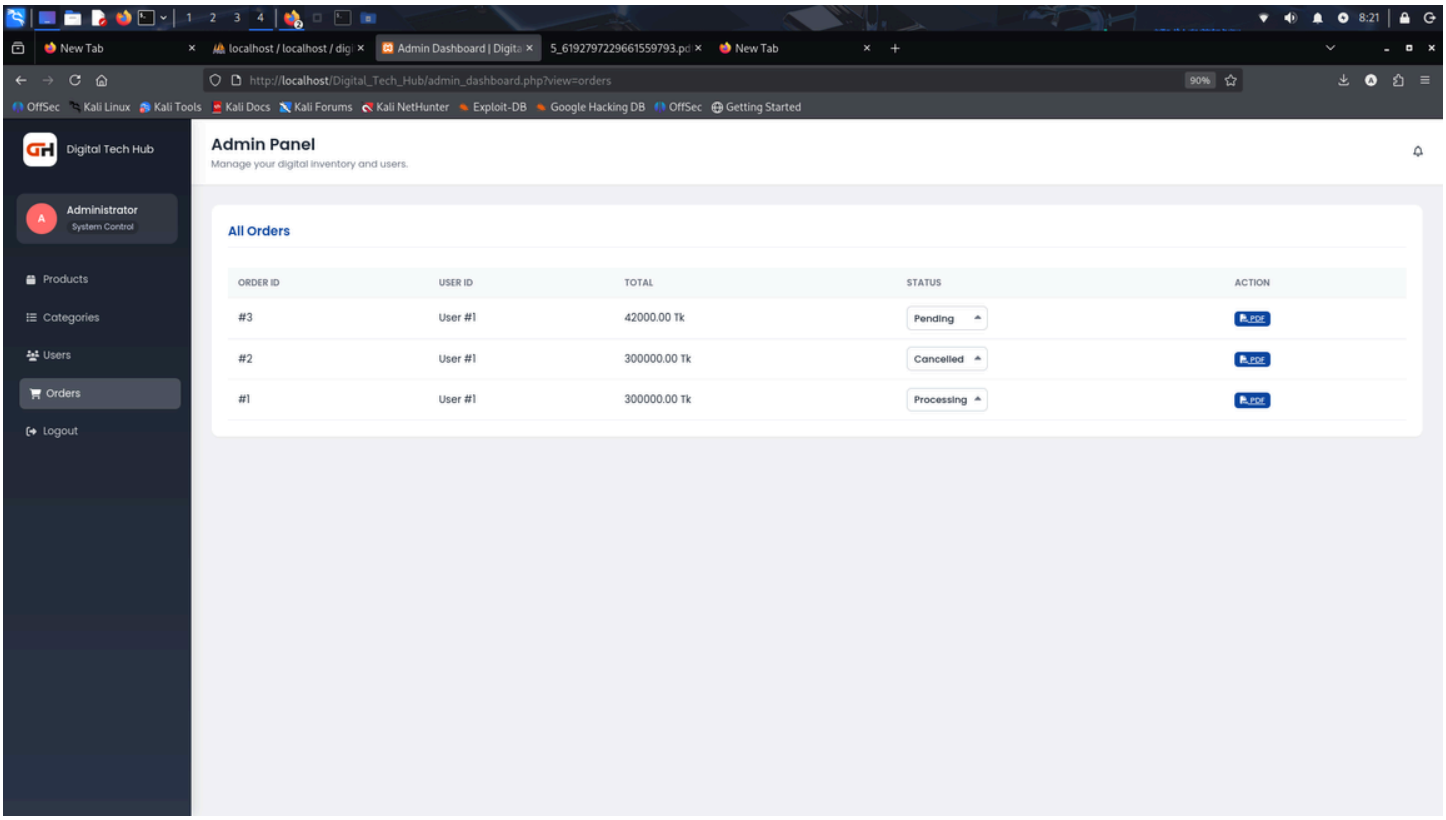

```php
if (isset($_GET['block_user'])) {
    $uid = intval($_GET['block_user']);
    $db->query("UPDATE users SET status='Blocked' WHERE user_id=$uid");
    $msg = "User Blocked.";
}

// 4. Unblock User (GET) - ADDED FOR UNBLOCK FUNCTIONALITY
if (isset($_GET['unblock_user'])) {
    $uid = intval($_GET['unblock_user']);
    $db->query("UPDATE users SET status='Active' WHERE user_id=$uid");
    $msg = "User Unblocked and set to Active.";
}
```

- FRA.5: The system shall allow the Admin to view all submitted orders.



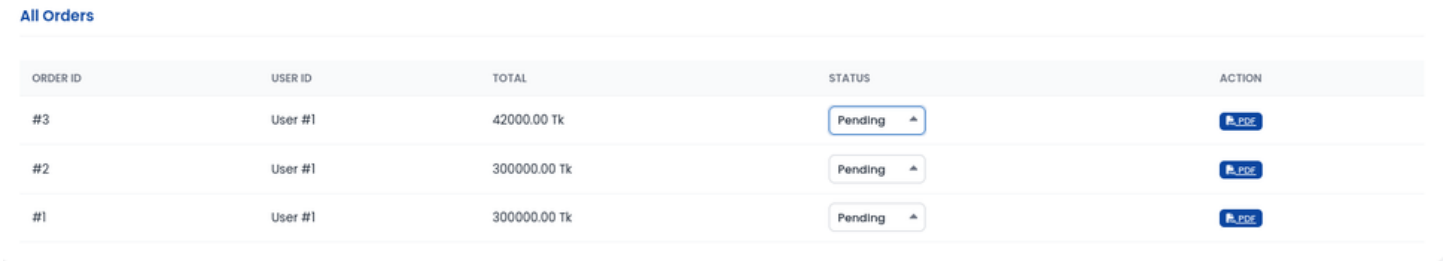

```php
<?php if (isset($_GET['view']) && $_GET['view'] == 'orders') {
    // NEW: Define the possible order statuses
    $order_statuses = ["Pending", "Processing", "Shipped", "Delivered", "Cancelled"];
?>
    <div class="panel-section">
        <h3>All Orders</h3>
        <div class="table-responsive">
            <table class="data-table">
                <thead>
                    <tr><th>Order ID</th><th>User ID</th><th>Total</th><th>Status</th><th>Action</th></tr>
                </thead>
                <tbody>
                    <?php
                    $res = $db->query("SELECT * FROM orders ORDER BY order_id DESC");
                    while ($row = $res->fetch_assoc()) {
                        echo "<tr>
                            <td>#{$row['order_id']}</td>
                            <td>User #{$row['user_id']}</td>
                            <td>{$row['total_amount']} Tk</td>
                            <td>
                                <select
                                    class='status-select'
                                    onchange='window.location.href = \"?view=orders&order_id={$row['order_id']}&update_order_status=\" + this.value'
                                >
                                ";
                                // NEW: Loop through statuses to create the dropdown
                                foreach ($order_statuses as $status) {
                                    $selected = ($row['order_status'] == $status) ? 'selected' : '';
                                    echo "<option value='$status' $selected>$status</option>";
                                }
                                echo "</select>
                            </td>
                            <td>
                                <a href='generate_pdf.php?order_id={$row['order_id']}' target='_blank' class='btn-primary btn-sm'><i class='fa fa-file-pdf'></i> PDF</a>
                            </td>
                        </tr>";
                    }
                    ?>
                </tbody>
            </table>
        </div>
    </div>
<?php } ?>
```

- FRA.6: The system shall allow the Admin to update the status of an order.



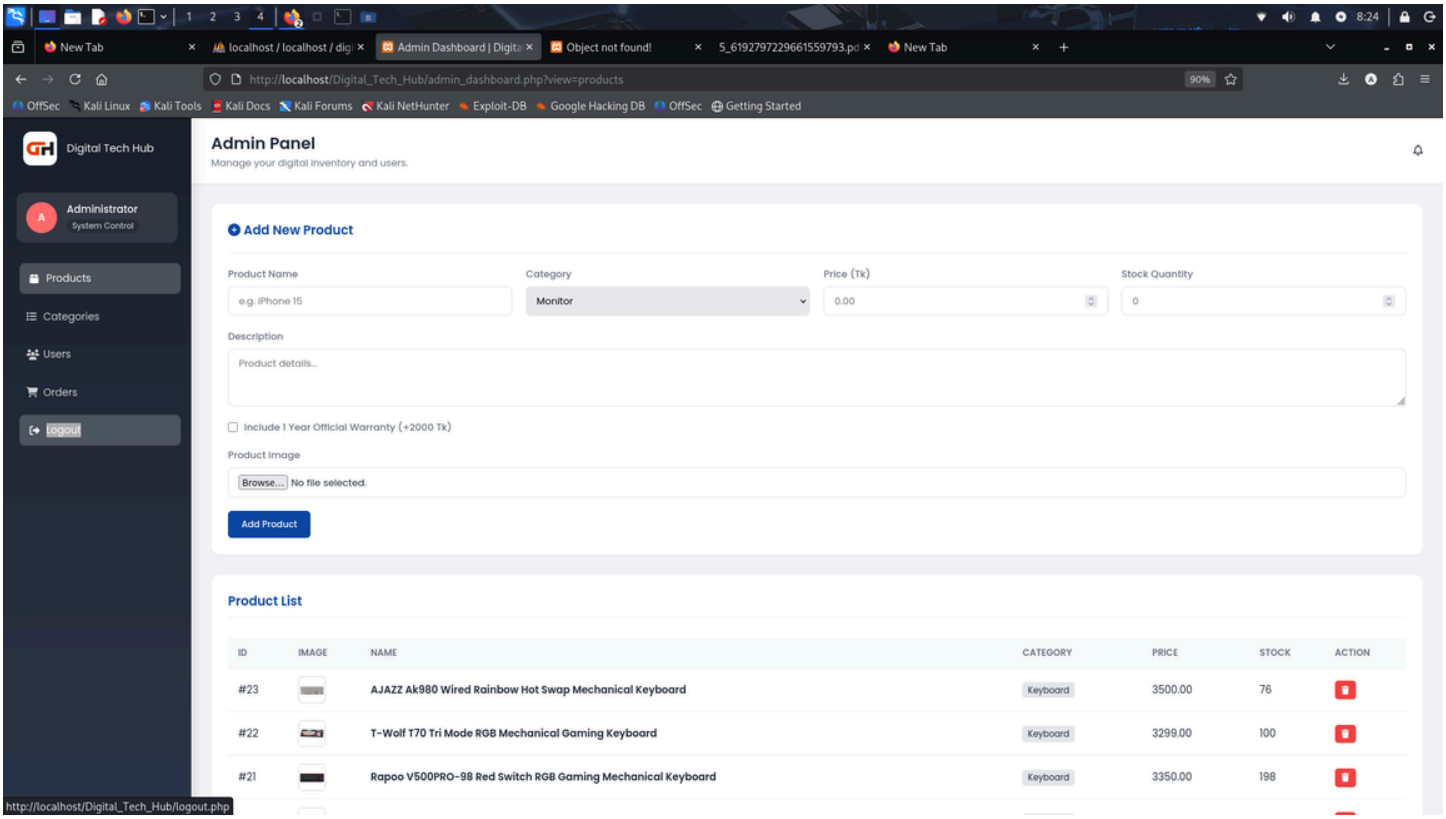

```php
if (isset($_GET['update_order_status']) && isset($_GET['order_id'])) {
    $order_id = intval($_GET['order_id']);
    $new_status = $db->real_escape_string($_GET['update_order_status']);

    // Check if the order_id is valid
    $check = $db->query("SELECT 1 FROM orders WHERE order_id=$order_id");

    if ($check->num_rows > 0) {
        $db->query("UPDATE orders SET order_status='$new_status' WHERE order_id=$order_id");
        $msg = "Order #$order_id status updated to **$new_status** successfully.";
    } else {
        $msg = "Error: Order #$order_id not found.";
    }
}
```

- FRA.7: The system shall enable the Admin to generate customer bills in PDF format.




```php
</td>
<td>
    <a href='generate_pdf.php?order_id={$row['order_id']}' target='_blank' class='btn-primary btn-sm'><i class='fa fa-file-pdf'></i> PDF</a>
</td>
```

- FRA.8: The system shall allow the Admin to log out.




```
<li><a href="?view=products" class="<?php echo (!isset($_GET['view']) || $_GET['view'] == 'products') ? 'active' : ''; ?>"><i class="fa fa-box"></i> Products</a></li>
<li><a href="?view=categories" class="<?php echo (isset($_GET['view']) && $_GET['view'] == 'categories') ? 'active' : ''; ?>"><i class="fa fa-list"></i> Categories</a></li>
<li><a href="?view=users" class="<?php echo (isset($_GET['view']) && $_GET['view'] == 'users') ? 'active' : ''; ?>"><i class="fa fa-users"></i> Users</a></li>
<li><a href="?view=orders" class="<?php echo (isset($_GET['view']) && $_GET['view'] == 'orders') ? 'active' : ''; ?>"><i class="fa fa-shopping-cart"></i> Orders</a></li>
<li class="logout-item"><a href="logout.php"><i class="fa fa-sign-out-alt"></i> Logout</a></li>
```

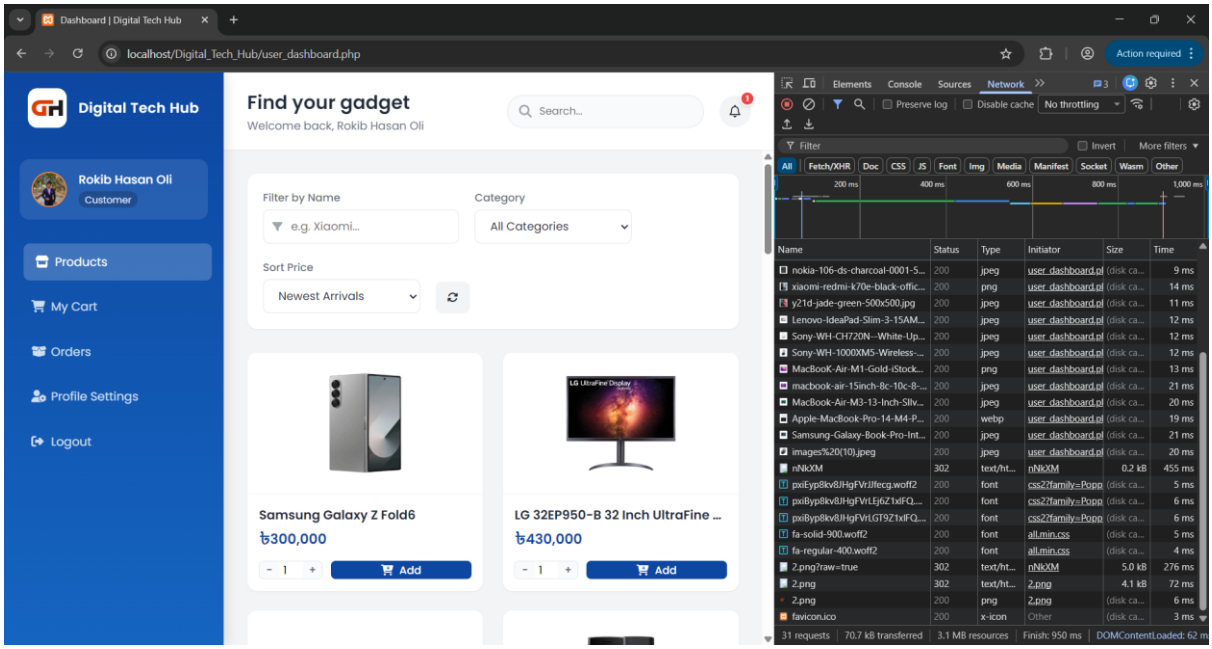
Non-Functional Requirements/Specifications

2.1 Performance Specifications (NFR-P1: Load Time)

- **Proposed Requirement (NFR-P1):** "The website's main pages (homepage, category pages) shall load within **10 seconds** under normal load conditions."
- **Completed Implementation:** To achieve optimal performance, we implemented the **Singleton Design Pattern** for database management. This ensures that the application reuses a single database connection instance per request lifecycle instead of opening a new connection for every query, significantly reducing server overhead and latency.
- **Screenshot of Code :**

```php
public static function getInstance() {
    if (self::$instance === null) {
        self::$instance = new Database();
    }
    return self::$instance;
}
```

- **Screenshot of UI :**



2.2 Security Specifications (NFR-S1: Password Hashing)

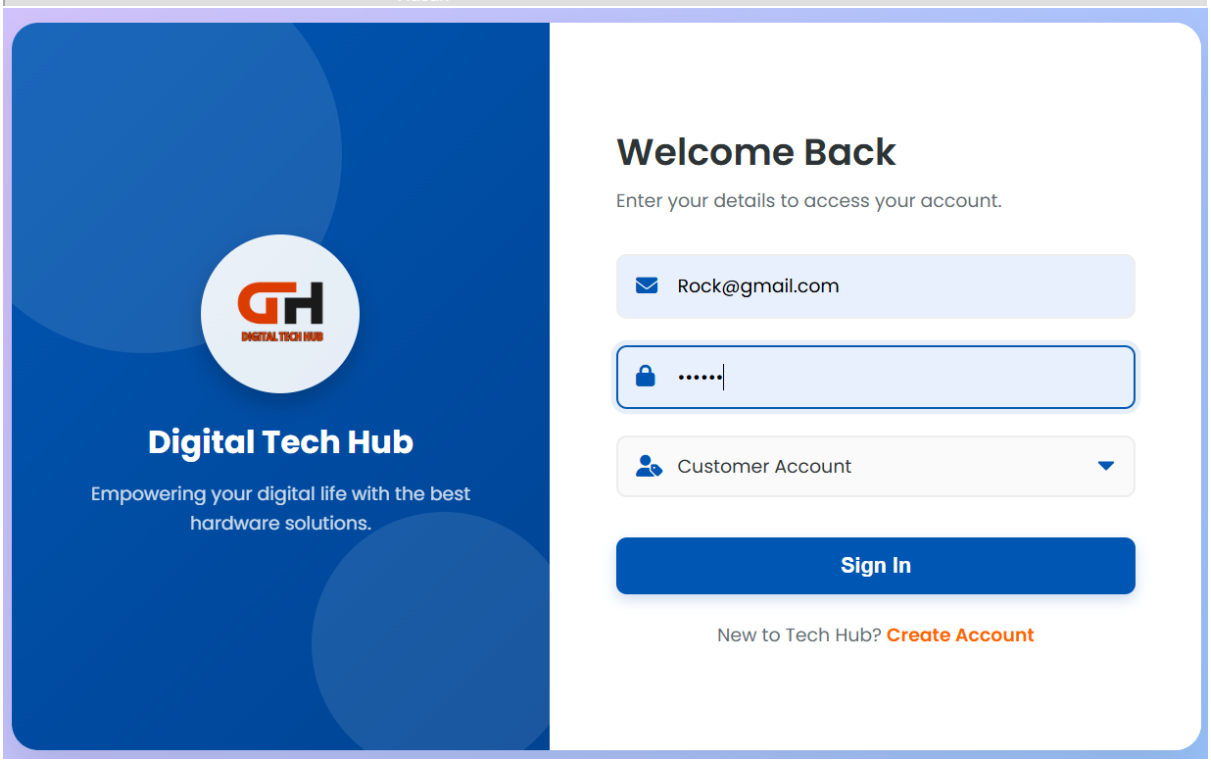
- **Proposed Requirement (NFR-S1):** "All user and admin login credentials shall be **hashed** (e.g., using PHP's password hashing functions) and stored securely in the MySQL database."
- **Completed Implementation:** We strictly avoided storing plain-text passwords. During user registration, the system uses PHP's **password_hash()** function (BCRYPT algorithm) to encrypt the password before insertion. During login, **password_verify()** is used to authenticate the user against the stored hash.

- **Screenshot of Code :**

```php
if (isset($_POST['register'])) {
    $name = $db->real_escape_string($_POST['name']);
    $email = $db->real_escape_string($_POST['email']);
    $pass = password_hash($_POST['password'], PASSWORD_DEFAULT);
    $check_query = "SELECT * FROM users WHERE email = '$email'";
    $check_result = $db->query($check_query);

    if ($check_result->num_rows > 0) {
        $msg = "This account already exists on your device";
        $msg_type = "error";
    } else {
        $sql = "INSERT INTO users (full_name, email, password) VALUES ('$name', '$email', '$pass')";
        if ($db->query($sql)) {
            $msg = "Registration Successful! Please Login.";
            $msg_type = "success";
        } else {
            $msg = "Error: " . $db->error;
            $msg_type = "error";
        }
    }
```

- **Screenshot of UI :**

	user_id	full_name	email	password
Edit Copy Delete	1	Rokib Hasan	rokib@nsu.com	$2y$10$XJ6pOsEDj/w3.g2CGVqXpODwGnL6VlOPhlnevAzRYk1...
Edit Copy Delete	2	Rakibul Hasan	rakibul@nsu.com	$2y$10$9hOwgt867nQdhY6RlSbcgeMiwN4iWd8o5FbpzNThFz3...
Edit Copy Delete	3	Md Rakibul Hasan	lol@gmail.com	$2y$10$9Dk/0oORxK1if4NbXOMuZesEjDh.mf3k628A1g63h5d...
Edit Copy Delete	10	Md Rakibul Hasan	Rock@gmail.com	$2y$10$8Jyw4Q2ZbS2Xf7OUh5MAMu5hQuvcKSe5lqvErRgVaBu...



2.3 Security Specifications (NFR-S2: Input Validation)

- **Proposed Requirement (NFR-S2):** "The system shall employ input validation on all forms (login, registration, profile update, product entry) to prevent SQL Injection and XSS attacks."

- **Completed Implementation:** We implemented defense-in-depth security. For critical transactions, we used Prepared Statements (in **OrderFacade.php**) which separate SQL code from user data. For general form inputs, we used **$db->real_escape_string()** to sanitize characters that could alter SQL commands**.**
- **Screenshot of Code :**

SQL Injection Prevention:

```php
if (isset($_POST['register'])) {
    $name = $db->real_escape_string($_POST['name']);
    $email = $db->real_escape_string($_POST['email']);
```

XSS Prevention:

```php
<div class="user-info">
    <h3><?php echo htmlspecialchars($_SESSION['name']); ?></h3>
    <span class="badge">Customer</span>
</div>
```

2.4 Usability Specifications (NFR-U1: Intuitive Interface)

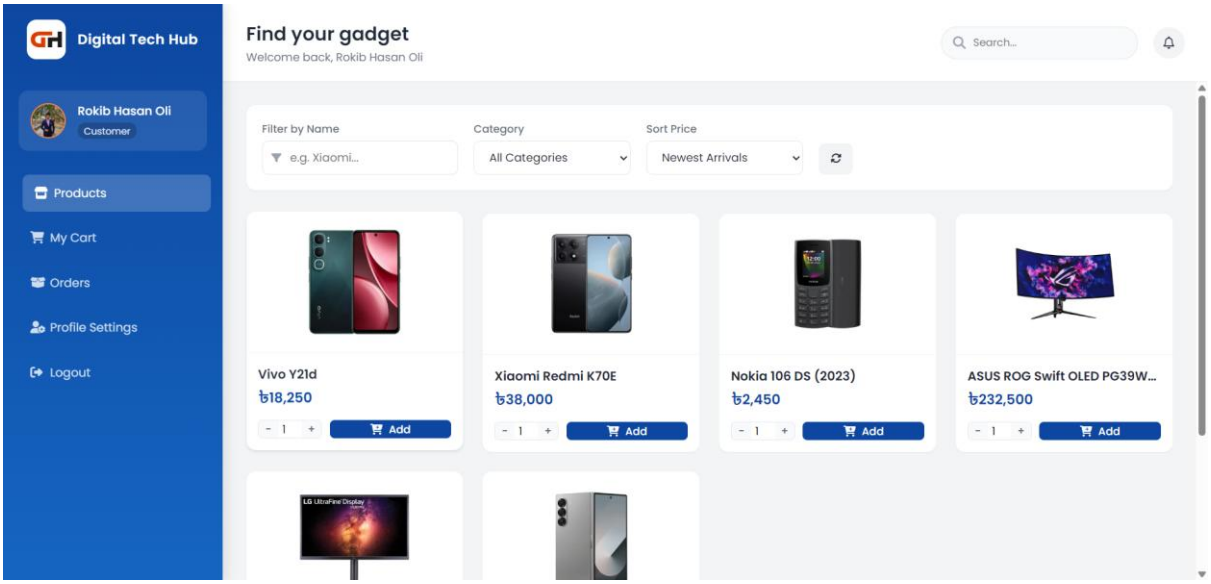
- **Proposed Requirement (NFR-U1):** "The User interface shall be intuitive and easy to navigate, allowing first-time users to complete a purchase within 5 minutes of browsing."
- **Completed Implementation:** We designed a clean, grid-based layout for the User Dashboard with a persistent sidebar for easy navigation. Visual cues, such as the "Order Successful" animation and clear "Add to Cart" buttons, guide the user effortlessly through the purchasing process.
- **Screenshot of Code :**

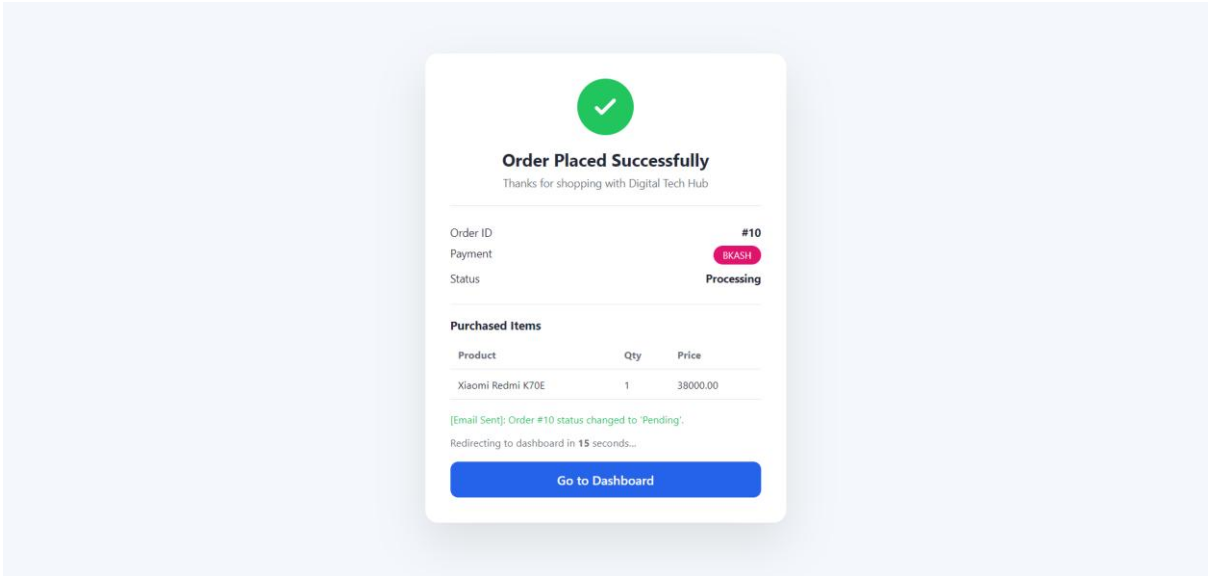
checkout_process.php

D: > XAMPP > htdocs > Digital_Tech_Hub > checkout_process.php

```
211  <div class="card">
212
213      <div class="checkmark">
214          <svg fill="none" viewBox="0 0 24 24" stroke="currentColor">
215              <path stroke-width="3" stroke-linecap="round" stroke-linejoin="round"
216                    d="M5 13l4 4L19 7" />
217          </svg>
218      </div>
219
220      <h2>Order Placed Successfully</h2>
221      <div class="subtitle">Thanks for shopping with Digital Tech Hub</div>
```

- **Screenshot of UI :**

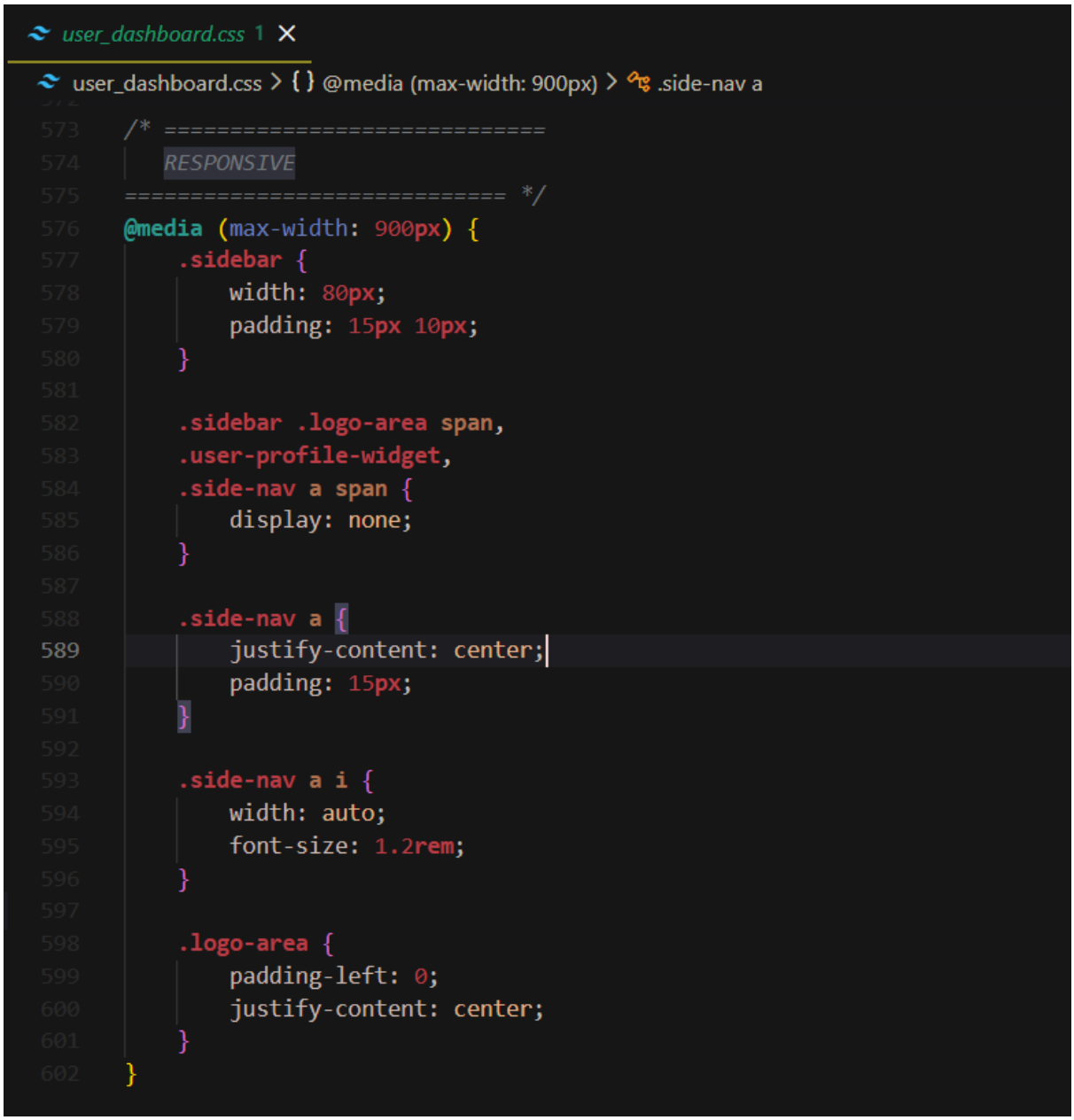




2.5 Usability Specifications (NFR-U2: Responsiveness)

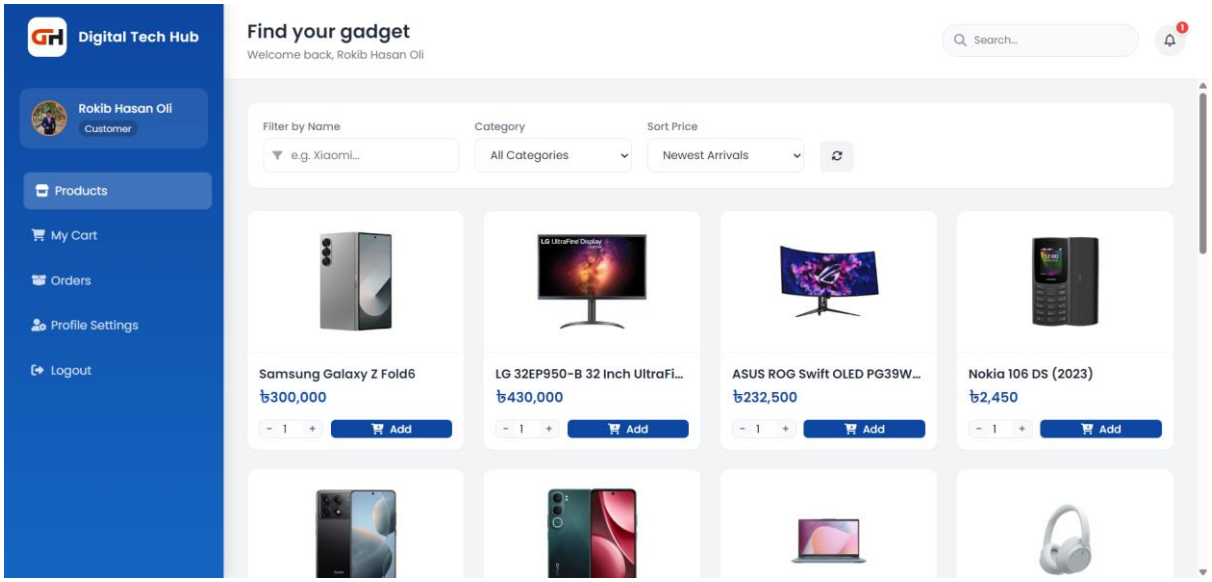
- **Proposed Requirement (NFR-U2):** "The system shall be responsive, ensuring the interface is accessible and functional across various devices (desktop, Laptop, tablet)."
- **Completed Implementation:** We utilized CSS3 Media Queries to ensure the layout adapts dynamically to different screen sizes. The dashboard sidebar and grids automatically stack or resize when the viewport width decreases (e.g., on mobile devices).

- **Screenshot of Code :**

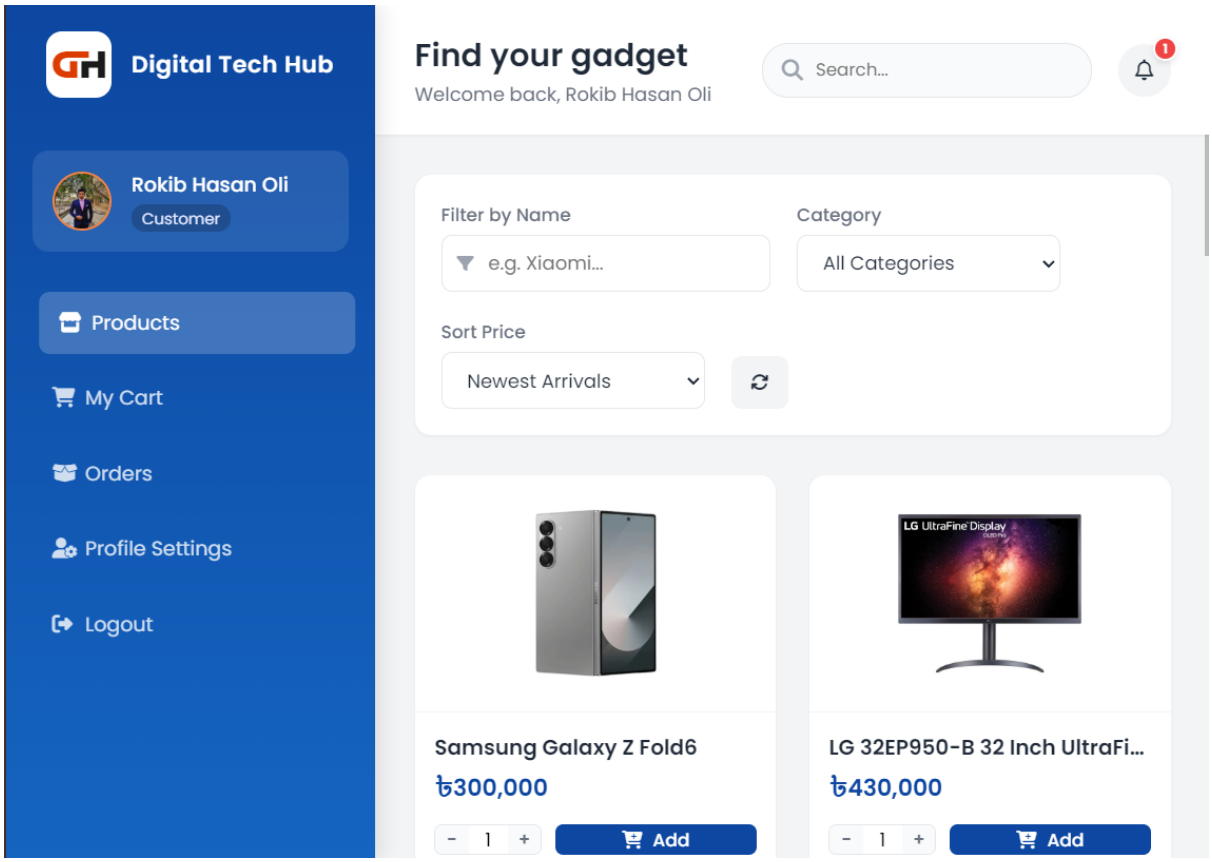


- **Screenshot of UI :**

  Desktop, Laptop:



Tablet:



2.6 Reliability Specifications (NFR-R1: Data Integrity)

- **Proposed Requirement (NFR-R1):** "The database (MySQL) must ensure data integrity for all critical transactions (e.g., order placement, inventory updates)."

- **Completed Implementation:** We enforced Referential Integrity using **Foreign Keys** in our MySQL database. This ensures that an order cannot exist without a valid user, and order items must be linked to valid products. Additionally, the ON DELETE CASCADE constraint cleans up related data automatically to prevent orphaned records.

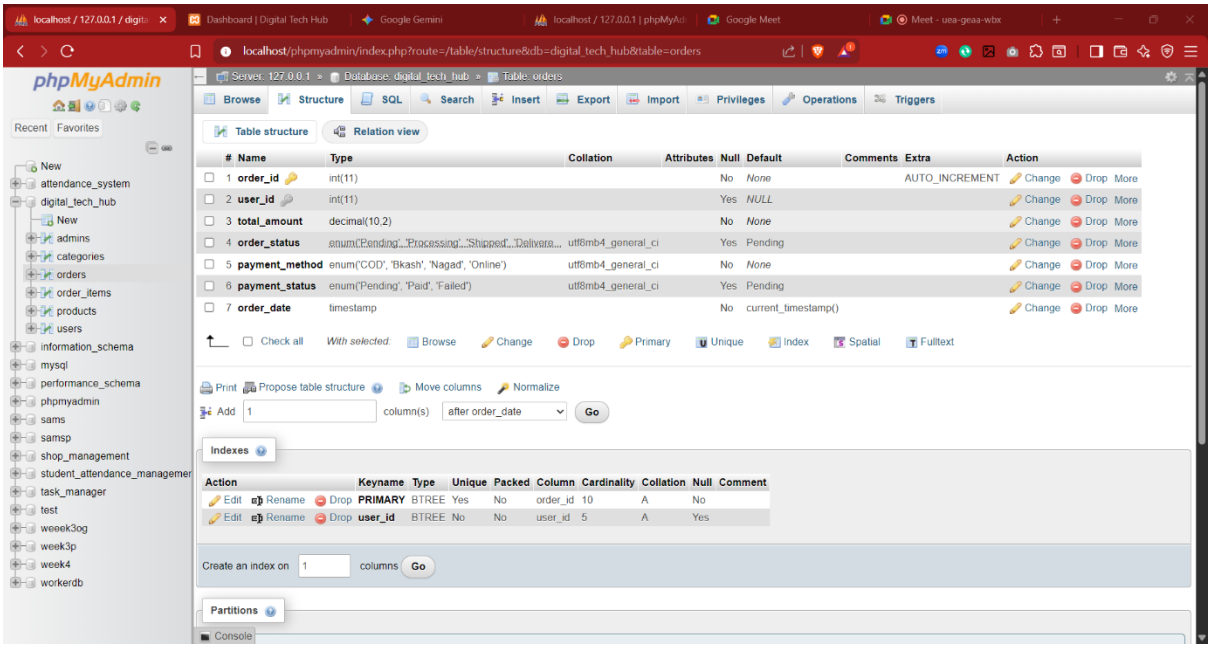
- **Screenshot of Code :**

Digital_Tech_Hub.sql

D: > XAMPP > htdocs > Digital_Tech_Hub > Database > Digital_Tech_Hub.sql

```sql
CREATE TABLE IF NOT EXISTS orders (
    order_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    total_amount DECIMAL(10, 2) NOT NULL,
    order_status ENUM('Pending', 'Processing', 'Shipped', 'Delivered', 'Cancelled') DEFAULT 'Pending',
    payment_method ENUM('COD', 'Bkash', 'Nagad', 'Online') NOT NULL,
    payment_status ENUM('Pending', 'Paid', 'Failed') DEFAULT 'Pending',
    order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS order_items (
    order_item_id INT AUTO_INCREMENT PRIMARY KEY,
    order_id INT,
    product_id INT,
    quantity INT NOT NULL,
    unit_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) ON DELETE CASCADE
);
```

- **Screenshot of UI :**



2.7 Maintainability Specifications (NFR-M1: Documentation & Standards)

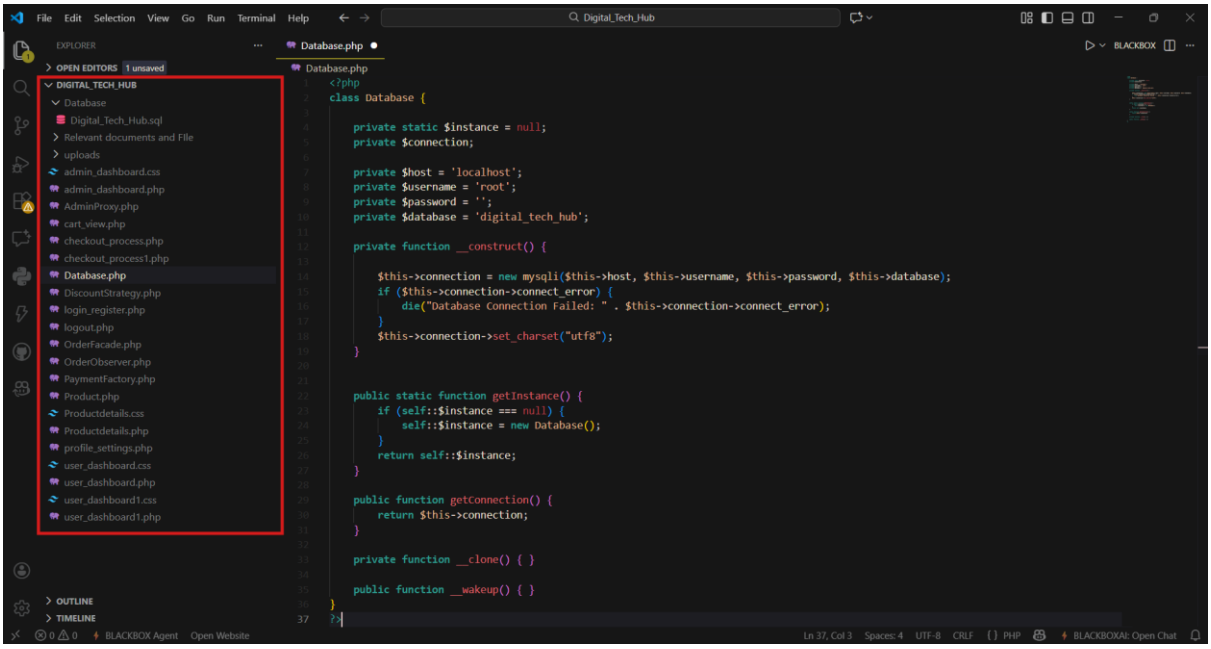
- **Proposed Requirement (NFR-M1):** "The code base (PHP, HTML, CSS) shall be well-documented and follow standard coding practices to facilitate future updates and bug fixes."

- **Completed Implementation:** We adopted standard **Design Patterns** (Singleton, Factory, Facade) which inherently organize code into modular, maintainable structures. We also included comments explaining the purpose of complex logic, such as the OrderFacade and Database connection classes.

- **Screenshot of Code :**

OrderFacade.php

```php
<?php
require_once 'Database.php';
require_once 'PaymentFactory.php';
require_once 'OrderObserver.php';

class OrderFacade {
    public function placeOrder($userId, $cartTotal, $paymentMethod, $cartItems) {
        $db = Database::getInstance()->getConnection();

        // 1. Handle Payment (Factory)
        $paymentObj = PaymentFactory::create($paymentMethod);
        $paymentStatus = $paymentObj->pay($cartTotal); // In real app, check return value

        // 2. Create Order (Database)
        $stmt = $db->prepare("INSERT INTO orders (user_id, total_amount, payment_method, order_status) VALUES (?, ?, ?, 'Pending')");
        $stmt->bind_param("ids", $userId, $cartTotal, $paymentMethod);
        $stmt->execute();
        $orderId = $db->insert_id;

        // 3. Save Items
        foreach ($cartItems as $item) {
            $stmtItem = $db->prepare("INSERT INTO order_items (order_id, product_id, quantity, unit_price) VALUES (?, ?, ?, ?)");
            $stmtItem->bind_param("iiid", $orderId, $item['id'], $item['qty'], $item['price']);
            $stmtItem->execute();
        }

        // 4. Notify (Observer)
        $subject = new OrderSubject();
        $subject->attach(new EmailNotifier());
        $subject->attach(new AdminLogObserver());
        $subject->notify($orderId, 'Pending');

        return $orderId;
    }
}
?>
```



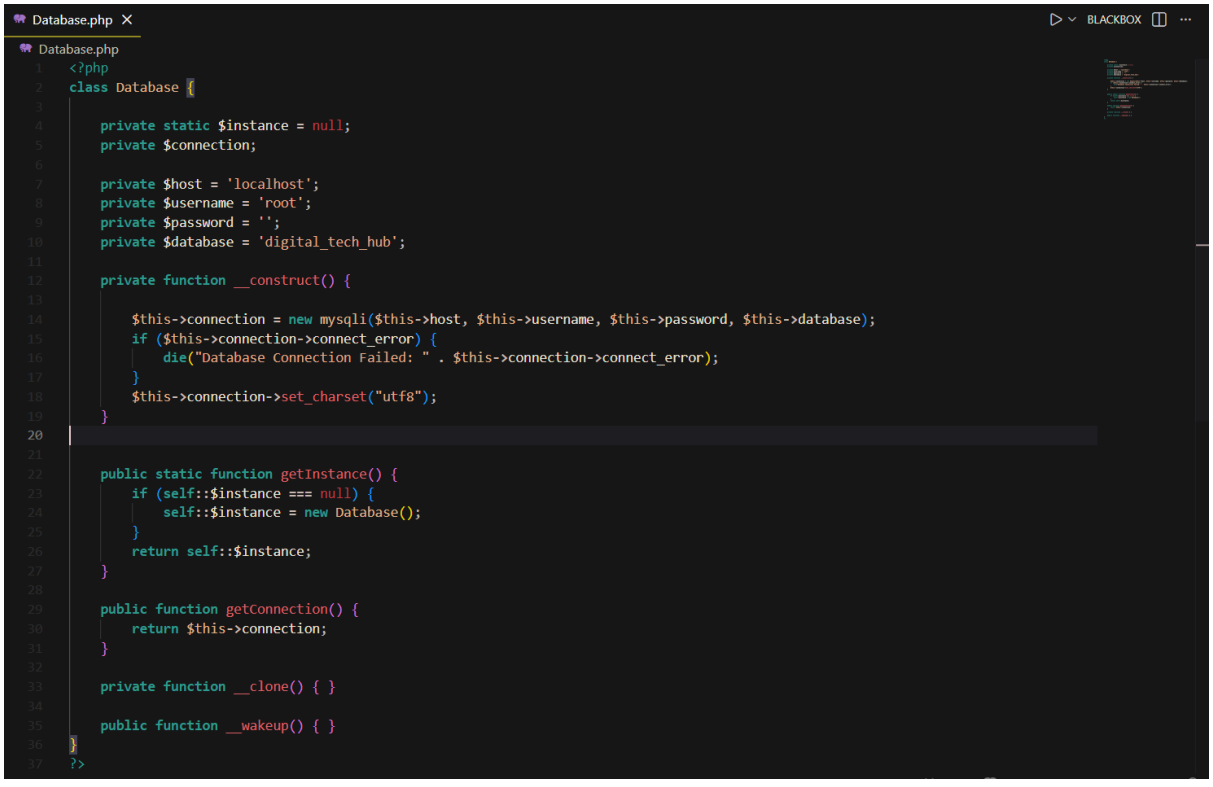

```php
<?php
class Database {

    private static $instance = null;
    private $connection;

    private $host = 'localhost';
    private $username = 'root';
    private $password = '';
    private $database = 'digital_tech_hub';

    private function __construct() {

        $this->connection = new mysqli($this->host, $this->username, $this->password, $this->database);
        if ($this->connection->connect_error) {
            die("Database Connection Failed: " . $this->connection->connect_error);
        }
        $this->connection->set_charset("utf8");
    }


    public static function getInstance() {
        if (self::$instance === null) {
            self::$instance = new Database();
        }
        return self::$instance;
    }

    public function getConnection() {
        return $this->connection;
    }

    private function __clone() { }

    public function __wakeup() { }
}
?>
```

3. Implementation of Design Patterns

To ensure the system is maintainable, scalable, and secure, we implemented **7 standard Software Engineering Design Patterns**. Below is the explanation of how each pattern addresses a specific problem in the **Digital Tech Hub** system.

3.1 Singleton Pattern (Database Connection)

- **Problem:** Opening a new database connection for every single query consumes server memory and slows down the website (NFR-P1).
- **Implementation:** We created a Database class with a private constructor and a static getInstance() method. This ensures that only **one** connection object exists per user session, no matter how many times the database is accessed.
- **Screenshot of Code:**



3.2 Factory Pattern (Payment Processing)

- **Problem:** The checkout system needs to handle multiple payment methods (Bkash, Nagad, COD). Hardcoding if-else logic for every method makes the code messy and hard to extend.
- **Implementation:** We used the **Factory Pattern** to create a PaymentFactory. The checkout logic simply asks the factory for a payment object based on the user's selection, without worrying about the specific class logic.
- **Screenshot of Code:**

PaymentFactory.php

```php
<?php
require_once 'Database.php';
interface IPayment {
    public function pay($amount);
}

class BkashPayment implements IPayment {
    public function pay($amount) { return "Paid $amount Tk via Bkash (Trans ID: BK-" . rand(1000,9999) . ")"; }
}

class NagadPayment implements IPayment {
    public function pay($amount) { return "Paid $amount Tk via Nagad (Trans ID: NG-" . rand(1000,9999) . ")"; }
}

class CODPayment implements IPayment {
    public function pay($amount) { return "Amount $amount Tk to be paid on Delivery."; }
}

class PaymentFactory {
    public static function create($method) {
        if ($method == 'Bkash') return new BkashPayment();
        if ($method == 'Nagad') return new NagadPayment();
        return new CODPayment(); // Default
    }
}
?>
```

3.3 Facade Pattern (Order Processing)

- **Problem:** Placing an order involves a complex sequence: processing payment, saving to the database, and sending notifications. If the checkout page handles all this, it violates the "Single Responsibility Principle."
- **Implementation:** We introduced OrderFacade as a simplified interface. The client (checkout page) calls one method placeOrder(), and the Facade handles the complex interactions between the Payment, Database, and Notification subsystems.

- **Screenshot of Code:**

OrderFacade.php

```php
<?php
require_once 'Database.php';
require_once 'PaymentFactory.php';
require_once 'OrderObserver.php';

class OrderFacade {
    public function placeOrder($userId, $cartTotal, $paymentMethod, $cartItems) {
        $db = Database::getInstance()->getConnection();

        $paymentObj = PaymentFactory::create($paymentMethod);
        $paymentObj->pay($cartTotal);

        $stmt = $db->prepare("INSERT INTO orders (user_id, total_amount, payment_method, order_status) VALUES (?, ?, ?, 'Pending')"
        $stmt->bind_param("ids", $userId, $cartTotal, $paymentMethod);
        $stmt->execute();
        $orderId = $db->insert_id;

        foreach ($cartItems as $item) {
            $stmtItem = $db->prepare("INSERT INTO order_items (order_id, product_id, quantity, unit_price) VALUES (?, ?, ?, ?)");
            $stmtItem->bind_param("iiid", $orderId, $item['id'], $item['qty'], $item['price']);
            $stmtItem->execute();
        }

        $subject = new OrderSubject();

        $subject->attach(new DatabaseNotifier($userId));

        $subject->attach(new EmailNotifier());
        $subject->attach(new AdminLogObserver());

        $subject->notify($orderId, 'Pending');

        return $orderId;
    }
}
?>
```

3.4 Observer Pattern (Notifications)

- **Problem:** When an order is placed, we need to perform multiple independent actions (Email User, Log for Admin, Alert Dashboard). Hardcoding these into the order logic makes it tightly coupled.
- **Implementation:** We used the **Observer Pattern**. The Order system acts as the "Subject" and broadcasts a notification. Independent "Observers" (EmailNotifier, AdminLogObserver) listen for this event and react automatically.

- **Screenshot of Code:**

OrderObserver.php

```php
<?php
require_once 'Database.php';

// 1. The Observer Interface
interface IObserver {
    public function update($orderId, $status);
}

// 2. Concrete Observer: Database Notification (The Real Feature)
class DatabaseNotifier implements IObserver {
    private $userId;

    // We need the User ID to know WHO to notify
    public function __construct($userId) {
        $this->userId = $userId;
    }

    public function update($orderId, $status) {
        $db = Database::getInstance()->getConnection();

        $message = "Your Order #$orderId is now confirmed! Status: $status";

        // Save to Database
        $stmt = $db->prepare("INSERT INTO notifications (user_id, message) VALUES (?, ?)");
        if ($stmt) {
            $stmt->bind_param("is", $this->userId, $message);
            $stmt->execute();
        }
    }
}
```

OrderObserver.php

```php
// 3. Concrete Observer: Email (Simulation)
class EmailNotifier implements IObserver {
    public function update($orderId, $status) {
        $_SESSION['msg_email'] = "Email sent to user for Order #$orderId";
    }
}

// 4. Concrete Observer: Admin Log
class AdminLogObserver implements IObserver {
    public function update($orderId, $status) {
        // Keeps a log for admins
        error_log("Order #$orderId placed at " . date('Y-m-d H:i:s'));
    }
}

// 5. The Subject (Observable)
class OrderSubject {
    private $observers = [];

    public function attach(IObserver $observer) {
        $this->observers[] = $observer;
    }

    public function notify($orderId, $status) {
        foreach ($this->observers as $observer) {
            $observer->update($orderId, $status);
        }
    }
}
?>
```

3.5 Strategy Pattern (Discounts)

- **Problem:** We need to apply different discount rules (e.g., Holiday Sale, No Discount) dynamically without rewriting the cart logic.
- **Implementation:** We defined a DiscountContext that can switch between different strategies (algorithms) at runtime. For example, if the coupon "HOLIDAY10" is used, the strategy switches to HolidayDiscount.
- **Screenshot of Code:**

DiscountStrategy.php

```php
<?php
interface IDiscountStrategy {
    public function calculate($total);
}

class NoDiscount implements IDiscountStrategy {
    public function calculate($total) { return $total; }
}

class HolidayDiscount implements IDiscountStrategy {
    public function calculate($total) { return $total * 0.90; }
}

class DiscountContext {
    private $strategy;

    public function __construct() {
        $this->strategy = new NoDiscount();
    }

    public function setStrategy(IDiscountStrategy $strategy) {
        $this->strategy = $strategy;
    }

    public function getFinalPrice($rawTotal) {
        return $this->strategy->calculate($rawTotal);
    }
}
?>
```

cart_view.php

```php
<?php
session_start();
require_once 'DiscountStrategy.php';

$rawTotal = 0;
if (isset($_SESSION['cart'])) {
    foreach ($_SESSION['cart'] as $item) {
        $rawTotal += ($item['price'] * $item['qty']);
    }
}

// Strategy Pattern Usage
$context = new DiscountContext();
$msg = "";

if (isset($_POST['apply_coupon'])) {
    if ($_POST['code'] == 'HOLIDAY10') {
        $context->setStrategy(new HolidayDiscount());
        $msg = "Coupon Applied: 10% Off";
    }
}
```

3.6 Decorator Pattern (Product Warranty)

- **Problem:** Users can add an optional "1 Year Warranty" to a product. Creating a separate class for every product-warranty combination (e.g., iPhoneWithWarranty) would lead to class explosion.
- **Implementation:** We used the **Decorator Pattern**. We wrap the BasicProduct object inside a WarrantyDecorator. This dynamically adds the warranty cost (+2000 Tk) and updates the description without altering the original product class.

- **Screenshot of Code:**

Product.php

```php
<?php
require_once 'Database.php';
interface IProduct {
    public function getPrice();
    public function getDescription();
}

class BasicProduct implements IProduct {
    protected $name;
    protected $price;

    public function __construct($name, $price) {
        $this->name = $name;
        $this->price = $price;
    }

    public function getPrice() { return $this->price; }
    public function getDescription() { return $this->name; }
}

class WarrantyDecorator implements IProduct {
    protected $product;

    public function __construct(IProduct $product) {
        $this->product = $product;
    }

    public function getPrice() {
        return $this->product->getPrice() + 2000; // Warranty costs 2000 Tk
    }

    public function getDescription() {
        return $this->product->getDescription() . " + 1 Year Official Warranty";
    }
}
?>
```

3.7 Proxy Pattern (Admin Security)

- **Problem:** The Admin Dashboard handles sensitive operations. We need to ensure that no unauthorized user can even attempt to load the dashboard logic.
- **Implementation:** We used the **Proxy Pattern**. The AdminProxy sits in front of the RealAdminDashboard. It intercepts the request, checks if the user has the "admin" role, and only then loads the real dashboard.

- **Screenshot of Code:**

AdminProxy.php

```php
}

// Real Subject
class RealAdminDashboard implements IAdminDashboard {
    public function render() {
        return true; // Access Granted signal
    }
}

// Proxy
class AdminProxy implements IAdminDashboard {
    private $realAdmin;

    public function render() {
        if (session_status() == PHP_SESSION_NONE) session_start();

        // Check Authentication & Role
        if (!isset($_SESSION['role']) || $_SESSION['role'] !== 'admin') {
            echo "<h2>Access Denied: You do not have Admin privileges.</h2>";
            echo "<a href='login_register.php'>Login</a>";
            exit();
        }

        // Lazy Initialization
        if ($this->realAdmin == null) {
            $this->realAdmin = new RealAdminDashboard();
        }

        return $this->realAdmin->render();
    }
}
?>
```

4. Diagram vs. Implementation Comparison

This section analyses the differences between the initial system design (Class and Sequence Diagrams) and the final implemented code. While the high-level architecture remains consistent, several refinements were made during development to incorporate **Design Patterns** for better scalability and maintainability.

4.1 Class Diagram Analysis

Original Design: The initial Class Diagram depicted a simplified object structure:

- A single **Product** class containing all attributes (Price, Description).
- A generic association between **Order** and **Payment**.
- A direct **Admin** class for managing the dashboard.

Implementation Changes (Refinements):

1. **Product Structure (Decorator Pattern):** In the implementation, we moved away from a rigid **Product** class. Instead, we implemented the ***Decorator Pattern*** using an **IProduct** interface and a **WarrantyDecorator** class. This allows us to dynamically add features (like "1 Year Warranty") and modify the price at runtime, which was not represented in the static class diagram.
2. **Payment Logic (Factory Pattern):** The diagram showed a simple **Payment** entity. In the code, we refactored this into a **Factory Pattern** structure. We created a **PaymentFactory** that generates specific payment objects (**BkashPayment**, **NagadPayment**) based on user selection.
3. **Admin Security (Proxy Pattern):** To enhance security (NFR-S1), we interposed an **AdminProxy** class between the client and the **RealAdminDashboard**. The original diagram showed direct access, whereas the implementation enforces a security check via this Proxy.

Screenshot of Code:

Product.php

```php
<?php
require_once 'Database.php';
interface IProduct {
    public function getPrice();
    public function getDescription();
}

class BasicProduct implements IProduct {
    protected $name;
    protected $price;

    public function __construct($name, $price) {
        $this->name = $name;
        $this->price = $price;
    }

    public function getPrice() { return $this->price; }
    public function getDescription() { return $this->name; }
}

class WarrantyDecorator implements IProduct {
    protected $product;

    public function __construct(IProduct $product) {
        $this->product = $product;
    }

    public function getPrice() {
        return $this->product->getPrice() + 2000; // Warranty costs 2000 Tk
    }

    public function getDescription() {
        return $this->product->getDescription() . " + 1 Year Official Warranty";
    }
}
?>
```

4.2 Sequence Diagram Analysis

Original Design: The initial Sequence Diagram for the "**Buy Product**" use case illustrated a synchronous flow:

1. User clicks "Buy".
2. Server checks price.
3. Server shows "Total Bill".

Implementation Changes (Refinements):

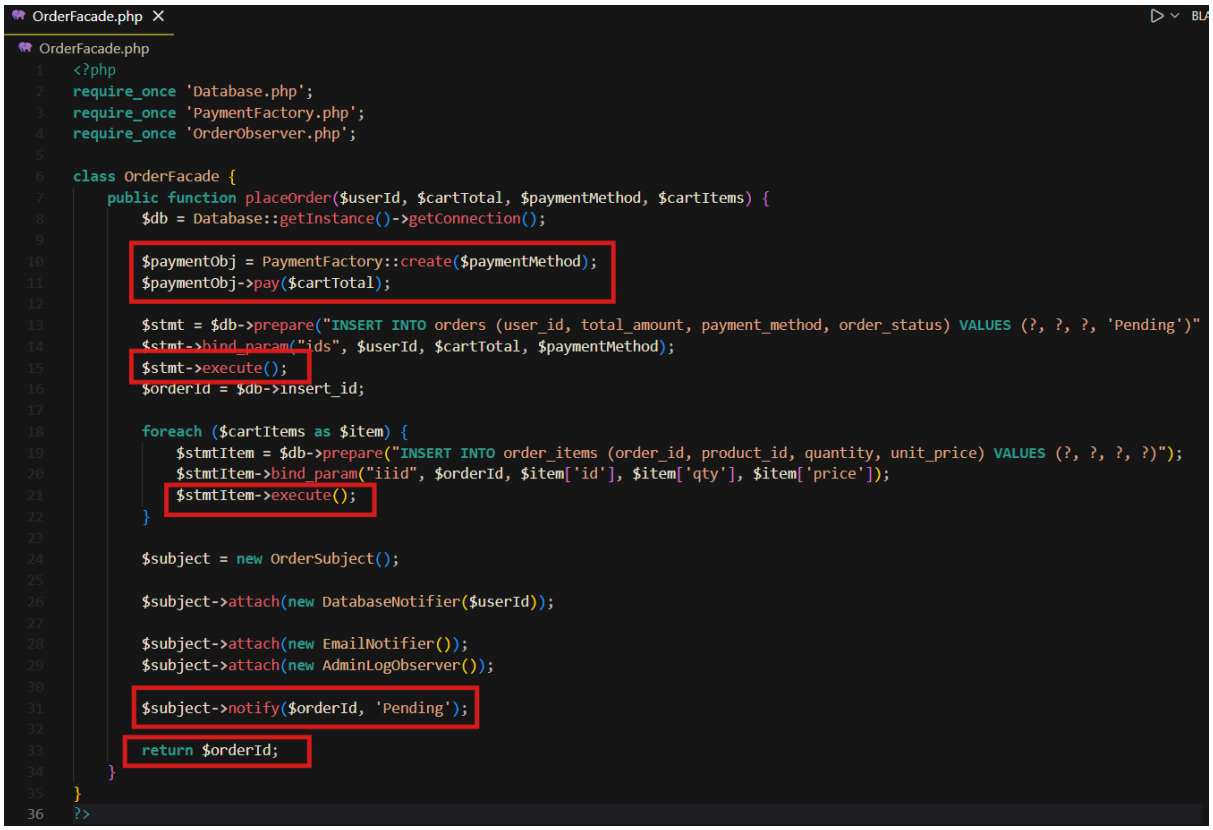
1. **Orchestration (Facade Pattern):** The actual checkout process is much more sophisticated than the simple "Check Price" step shown in the diagram. We implemented an **OrderFacade** that orchestrates multiple subsystems in a specific order:

   Payment Processing → Database Insertion → Session Cleanup.

2. **Asynchronous-like Events (Observer Pattern):** The most significant addition is the **Notification System**. The original sequence diagram ended at "Show Total Bill". In the implementation, immediately after the order is saved, the **OrderSubject** fires a notification event. This triggers the **EmailNotifier** and **AdminLogObserver** to run background tasks

(logging and emailing) before the final success message is rendered to the user.

Screenshot of Code:



```php
<?php
require_once 'Database.php';
require_once 'PaymentFactory.php';
require_once 'OrderObserver.php';

class OrderFacade {
    public function placeOrder($userId, $cartTotal, $paymentMethod, $cartItems) {
        $db = Database::getInstance()->getConnection();

        $paymentObj = PaymentFactory::create($paymentMethod);
        $paymentObj->pay($cartTotal);

        $stmt = $db->prepare("INSERT INTO orders (user_id, total_amount, payment_method, order_status) VALUES (?, ?, ?, 'Pending')");
        $stmt->bind_param("ids", $userId, $cartTotal, $paymentMethod);
        $stmt->execute();
        $orderId = $db->insert_id;

        foreach ($cartItems as $item) {
            $stmtItem = $db->prepare("INSERT INTO order_items (order_id, product_id, quantity, unit_price) VALUES (?, ?, ?, ?)");
            $stmtItem->bind_param("iiid", $orderId, $item['id'], $item['qty'], $item['price']);
            $stmtItem->execute();
        }

        $subject = new OrderSubject();

        $subject->attach(new DatabaseNotifier($userId));

        $subject->attach(new EmailNotifier());
        $subject->attach(new AdminLogObserver());

        $subject->notify($orderId, 'Pending');

        return $orderId;
    }
}
?>
```