Gyalpozhing College of Information Technology
Royal University of Bhutan

Assignment: Secure E-commerce Application Development

Submitted By:

Pema Chozom (12210024)

Mr. Karma Dorji

AI and Data Science Section ‘A’

Bachelor in Computer Science

Gyalpozhing College of Information Technology

27/11/2023

Objective:

To Design and develop an e-commerce web application that adheres to secure coding principles. This project will require you to demonstrate your understanding of security vulnerabilities, input validation, output sanitization, error handling, and more.

Instruction:

You are tasked with developing an e-commerce web application for a client who is concerned about the security of their online store. The client wants a robust and secure platform to ensure the safety of both the business and its customers. Your assignment is to design and develop this e-commerce platform with a focus on secure coding principles.Your application should include the following features:

- **Validation and Sanitization:** **[20 Marks]**

★ Implement both client-side and server-side input validation for user registration and login. Explain how validation enhances security. Provide examples of input that should be validated and implement this validation in your application.

- ❖ Validation is an essential component of online application security, which guards against a variety of attacks including cross-site scripting (XSS) and injection attacks. By verifying input data, you lower the possibility of fraudulent activity by making sure users only provide expected and correct information.
- ❖ Here are some client-side input validation implemented:
    1. User Registration:
        - ***Required Fields***

          Ensure that certain fields are not left empty

```
<h1>Sign Up</h1>
    <form action="process_signup.php" method="post">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username" required><br><br>

        <label for="email">Email:</label>
        <input type="email" id="email" name="email" required><br><br>
        <label for="phone">Phone Number:</label>
        <input type="tel" id="phone" name="phone" required><br><br>

        <label for="password">Password:</label>
        <input type="password" id="password" name="password" required><br><br>

        <label for="confirm-password">Confirm Password:</label>
        <input type="password" id="confirm-password" name="confirm-password"
required><br><br>
   </form>
```

- ***Password Matching***

  Ensure that the user enters the same password in both password and confirm password fields.

```
// Password matching validation
    var password = document.getElementById('password').value;
    var confirmPassword =
document.getElementById('confirm-password').value;
    if (password !== confirmPassword) {
        alert('Passwords do not match.');
        return false;
    }
```

- ***Email Input***

  Ensure that the user has valid email address with correct format

```
// Email validation
    var email = document.getElementById('email').value;
    var emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    if (!emailRegex.test(email)) {
        alert('Please enter a valid email address.');
        return false;
    }
```

2. User Login

- ***Required Fields***

```
<h1>Log In</h1>
    <form action="login.php" method="post">
        <label for="email">Email:</label>
        <input type="text" id="email" name="email" required><br><br>

        <label for="password">Password:</label>
        <input type="password" id="password" name="password"
required><br><br>
     </form>
```

❖ Here are some server-side input validation implemented:

1. User Registration:

- ***Generating and Handling OTP (One-Time Password):***

  Ensures that a one-time password is generated and stored for the user.

```
// An OTP is generated using the generateOTP function, and it is stored in the
database.
    function generateOTP($length = 6) {
        $otp = '';
        for ($i = 0; $i < $length; $i++) {
            $otp .= rand(0, 9); // Generate a random digit between 0 and 9
        }
        return $otp;
    }

    $OTP = generateOTP();

    // The generated OTP is then included in the database insert statement.
    // Insert data into the database using prepared statement
    $sql = "INSERT INTO users (username, email, password, otp, phone, role)
VALUES (?, ?, ?, ?, ?, ?)";
    $stmt = $conn->prepare($sql);
    $stmt->bind_param("ssssss", $username, $email, $password, $OTP, $phone,
$role);
```

- ***Sending Email with OTP***

  Process involves generating a unique OTP, associating it with a user, and then sending it securely via email.Ensures that the user is notified of the registration process and receives the OTP for verification.

```
if ($stmt->execute()) {
        $mail = new PHPMailer(true);
        // Server settings for Gmail SMTP
        $mail->isSMTP();
        $mail->Host = 'smtp.gmail.com';
        $mail->SMTPAuth = true;
        $mail->Username = '12210024.gcit@rub.edu.bt'; // Your Gmail address
        $mail->Password = 'jemyadslmjyqtyqg'; // Your Gmail password
        $mail->SMTPSecure = 'ssl'; // Use TLS encryption
        $mail->Port = 465; // TLS port (587)

        // Recipient and message settings
        $mail->setFrom('12210024.gcit@rub.edu.bt');
        $mail->addAddress($email); // Replace with the recipient's email
address
        $mail->Subject = 'Registration Successful! Use the OTP before 5
minutes';
        $mail->Body = 'Your OTP: ' . $OTP;

        // Send the email
        if ($mail->send()) {
            header("Location: otp.html");
        } else {
            echo '<script>alert("Email not sent!");</script>';
        }
    } else {
        echo "Error: " . $stmt->error;
    }
```

★ Use regular expressions to validate user input. Describe how regular expressions work and demonstrate their use in your code.

- ❖ Regular expressions (regex or regexp) are patterns that define a set of strings. They are an effective tool for manipulating text and matching patterns. Regular expressions are frequently used in the context of form validation to determine whether user input matches a particular pattern, such as a password that must meet certain requirements or a legitimate email address.
  - ***User validation***

    /^[a-zA-Z]+$/ ensures that the username contains only alphanumeric characters (letters and numbers).

```
// Username validation (Capital and small letters)
    var username = document.getElementById('username').value;
    var usernameRegex = /^[a-zA-Z ]+$/;
    if (!usernameRegex.test(username)) {
        alert('Please enter valid username.');
        return false;
    }
```

  - ***Email Validation:***

    ^[^\s@]+@[^\s@]+\.[^\s@]+$ checks for a valid email address format. It ensures there are no spaces, and the email has a proper structure (username@domain).

```
// Email validation
    var email = document.getElementById('email').value;
    var emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    if (!emailRegex.test(email)) {
        alert('Please enter a valid email address.');
        return false;
    }
```

- ***Password Validation:***

  ^(?=.*[!@#$%^&*()_+\-=\[\]{};':"\\|,.<>\/?])(?=.*[0-9])(?=.*[A-Z]).{8,}$ enforces the password to have at least 1 symbol, 1 number, 1 capital letter, and be at least 8 characters long.

```
// Password validation (at least 1 symbol, 1 number, and 1 capital letter)
    var password = document.getElementById('password').value;
    var passwordRegex =
/^(?=.*[!@#$%^&*()_+\-=\[\]{};':"\\|,.<>\/?])(?=.*[0-9])(?=.*[A-Z]).{8,}$/;

    if (!passwordRegex.test(password)) {
        alert('Please enter a valid password with at least 1 symbol, 1 number,
and 1 capital letter.');
        return false;
    }
```

- ***Phone Number Validation***

  ^(17|77)\d{6}$ verifies that the phone number starts with either 17 or 77 and is followed by exactly 6 digits.

```
// Phone number validation
    var phone = document.getElementById('phone').value;
    var phoneRegex = /^(17|77)\d{6}$/;
    if (!phoneRegex.test(phone)) {
        alert('Please enter a valid phone number starting with 17 or 77, and 8
digits in total.');
        return false;
    }
```

★ Implement output sanitization to protect against potential XSS attacks. Explain the concept and provide examples of how output should be sanitized. Apply this technique in your application.

- ❖ Output sanitization

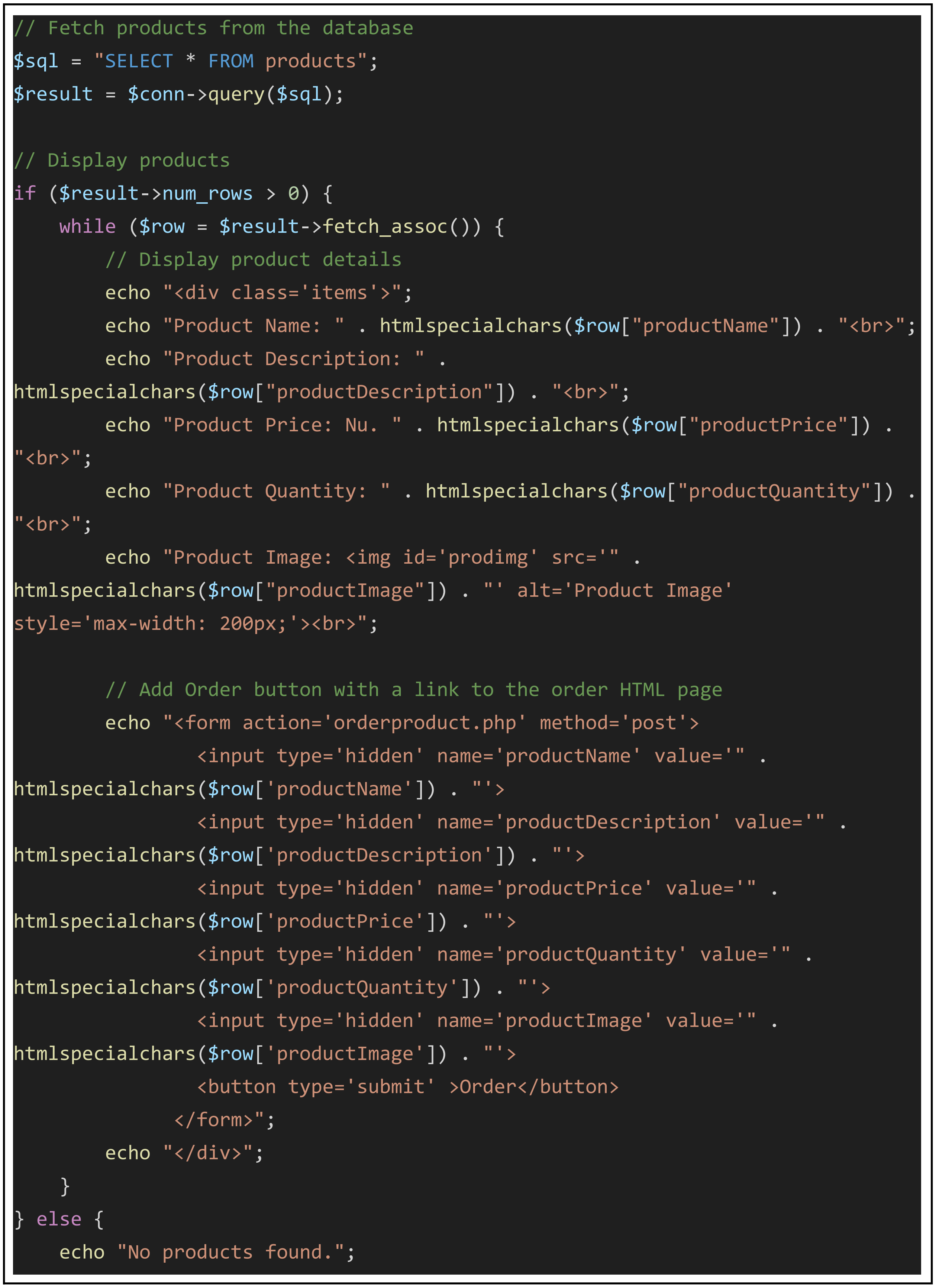
  Output sanitization is a crucial security measure to protect against Cross-Site Scripting (XSS) attacks. XSS attacks happen when a hacker inserts malicious scripts into websites, causing the victim's browser to run them. Before user-generated content is rendered in HTML, it must be encoded or escaped as part of output sanitization to guarantee that any potentially harmful content is handled as plain text rather than executable code.

  - ***htmlspecialchars function***

    Htmlspecialchar function is used to sanitize the input fields before including it in the main body. This function converts characters like '<' and '>' into their respective HTML entities ('&lt' ; and '&gt';), preventing them from being treated as part of an HTML tag. This helps protect against potential XSS attacks that might exploit the inclusion of user-generated content in the HTML output.

```
<!-- // process_signup.php -->
$mail->Body = 'Your OTP: ' . htmlspecialchars($OTP, ENT_QUOTES, 'UTF-8');
```

```
<!-- // view_product.php -->
<?php
// Connect to the database (modify these credentials)
$dbHost = 'localhost';
$dbUser = 'root';
$dbPass = '';
$dbName = 'user'; // Change this to your actual database name
$conn = new mysqli($dbHost, $dbUser, $dbPass, $dbName);

// Check the connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
```



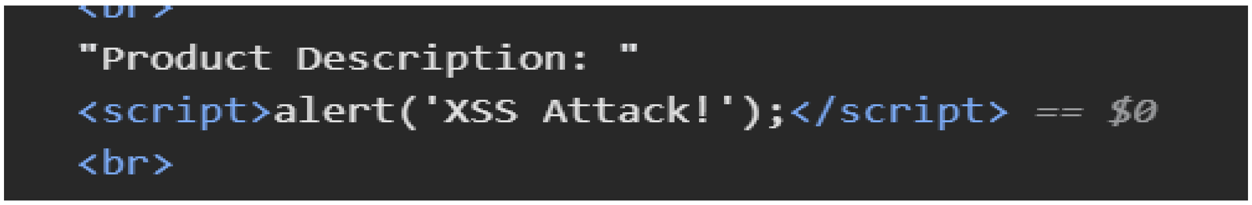
```php
// Fetch products from the database
$sql = "SELECT * FROM products";
$result = $conn->query($sql);

// Display products
if ($result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        // Display product details
        echo "<div class='items'>";
        echo "Product Name: " . htmlspecialchars($row["productName"]) . "<br>";
        echo "Product Description: " .
htmlspecialchars($row["productDescription"]) . "<br>";
        echo "Product Price: Nu. " . htmlspecialchars($row["productPrice"]) .
"<br>";
        echo "Product Quantity: " . htmlspecialchars($row["productQuantity"]) .
"<br>";
        echo "Product Image: <img id='prodimg' src='" .
htmlspecialchars($row["productImage"]) . "' alt='Product Image'
style='max-width: 200px;'><br>";

        // Add Order button with a link to the order HTML page
        echo "<form action='orderproduct.php' method='post'>
                <input type='hidden' name='productName' value='" .
htmlspecialchars($row['productName']) . "'>
                <input type='hidden' name='productDescription' value='" .
htmlspecialchars($row['productDescription']) . "'>
                <input type='hidden' name='productPrice' value='" .
htmlspecialchars($row['productPrice']) . "'>
                <input type='hidden' name='productQuantity' value='" .
htmlspecialchars($row['productQuantity']) . "'>
                <input type='hidden' name='productImage' value='" .
htmlspecialchars($row['productImage']) . "'>
                <button type='submit' >Order</button>
              </form>";
        echo "</div>";
    }
} else {
    echo "No products found.";
}

$conn->close();
?>
```

Checking whether output sanitization is implemented or not:

- Updated the product description for the specific product (‘Gho’) to include the injected HTML/script tag

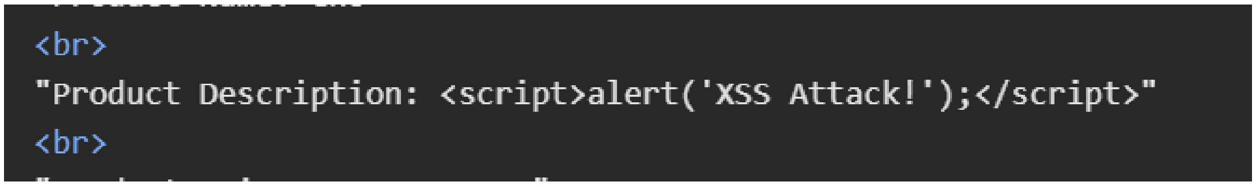
```
UPDATE products
SET productDescription = '<script>alert(\'XSS Attack!\');</script>'
WHERE productName = 'Gho';
```

- Before Output Sanitization, in inspect console Element



```
"Product Description: "
<script>alert('XSS Attack!');</script> == $0
<br>
```

- After Output Sanitization



```
<br>
"Product Description: <script>alert('XSS Attack!');</script>"
<br>
```

● **Error Handling and Logging:** **[10 Marks]**

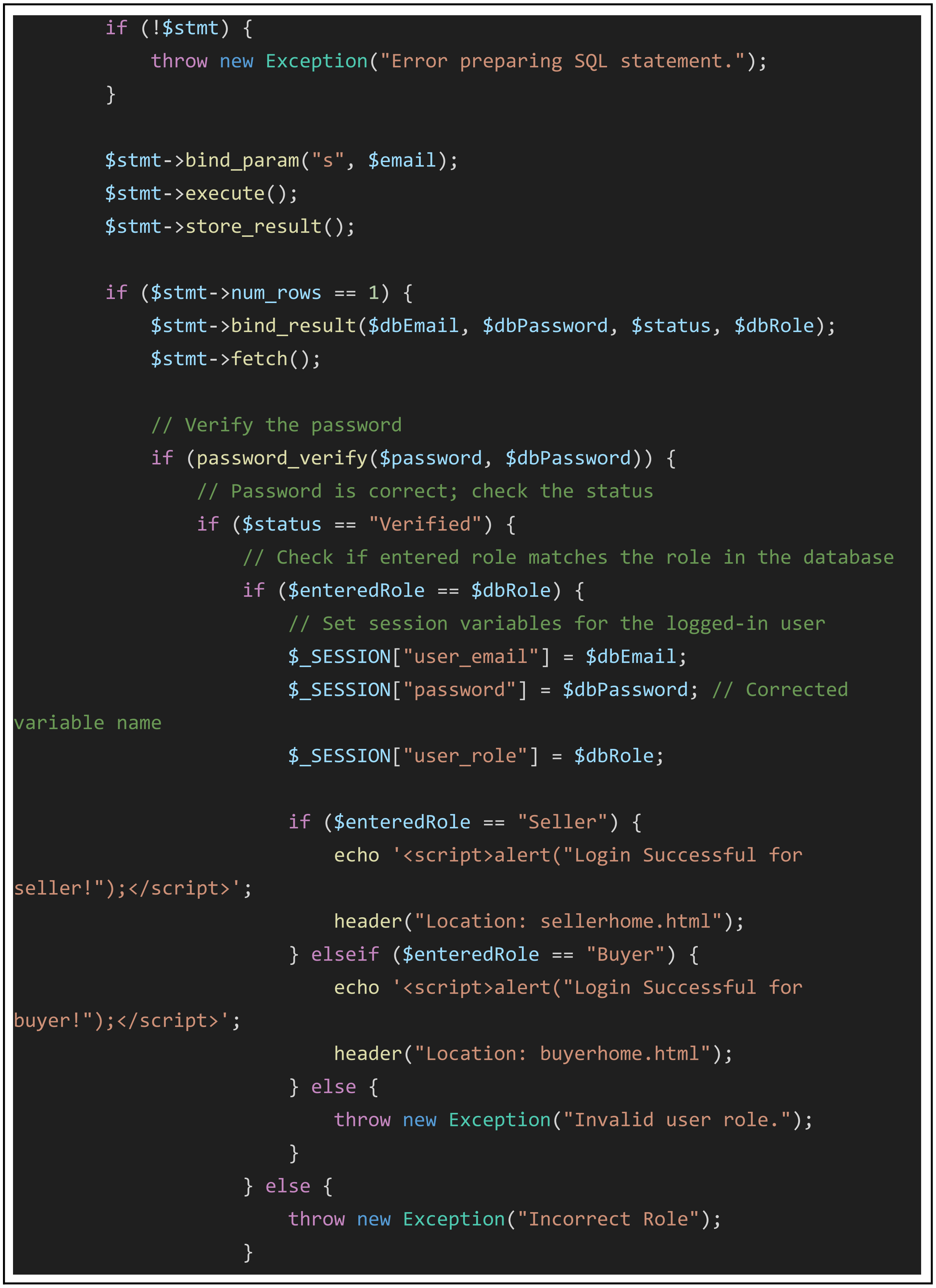
★ Exception Handling: Discuss the importance of exception handling in a web application. Implement secure exception handling in your code using try-catch blocks.

❖ Exception handling is a crucial aspect of building robust and secure web applications. During the execution of a web application, a number of unexpected occurrences or faults may arise, including failed database connections, network problems, unexpected user inputs, or code flaws. By detecting, managing, and responding to these problems in a regulated and safe way, exception management enables developers to stop applications from crashing or from disclosing private information to users.

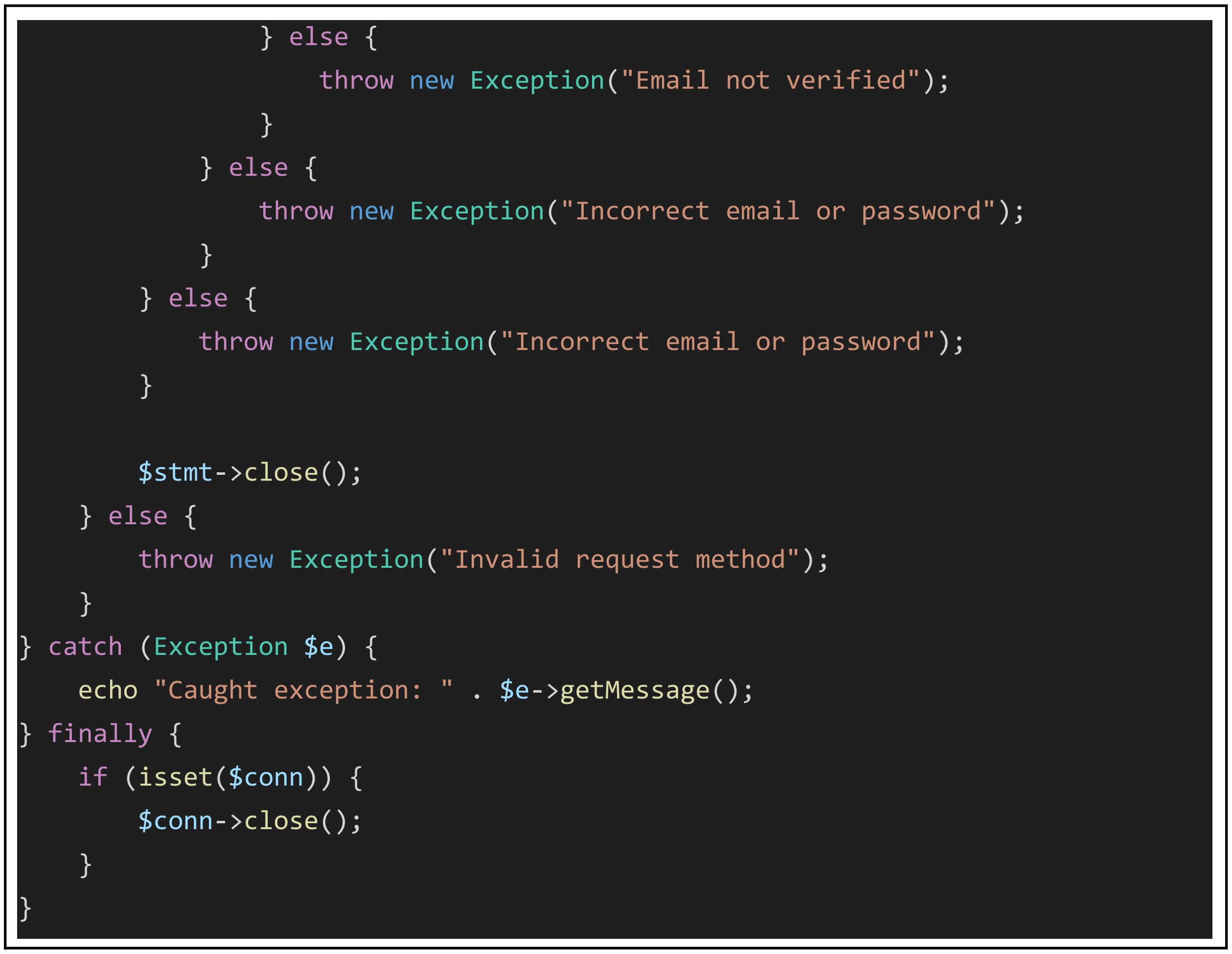
- ***try-catch blocks***

  If an exception occurs (e.g., connection error, query execution error), it will be caught in the catch (Exception $e) block, and an informative error message will be echoed. The finally block ensures that the database connection is closed, whether or not an exception occurred.

```
// Login.php
try {
    $conn = new mysqli($dbHost, $dbUser, $dbPass, $dbName);

    // Check the connection
    if ($conn->connect_error) {
        throw new Exception("Connection failed: " . $conn->connect_error);
    }

    if ($_SERVER["REQUEST_METHOD"] == "POST") {
        // Get user input (sanitize and validate as needed)
        $email = htmlspecialchars($_POST["email"]);
        $password = htmlspecialchars($_POST["password"]);
        $enteredRole = htmlspecialchars($_POST["role"]);

        $sql = "SELECT email, password, status, role FROM users WHERE email =
?";
        $stmt = $conn->prepare($sql);
```



```php
        if (!$stmt) {
            throw new Exception("Error preparing SQL statement.");
        }

        $stmt->bind_param("s", $email);
        $stmt->execute();
        $stmt->store_result();

        if ($stmt->num_rows == 1) {
            $stmt->bind_result($dbEmail, $dbPassword, $status, $dbRole);
            $stmt->fetch();

            // Verify the password
            if (password_verify($password, $dbPassword)) {
                // Password is correct; check the status
                if ($status == "Verified") {
                    // Check if entered role matches the role in the database
                    if ($enteredRole == $dbRole) {
                        // Set session variables for the logged-in user
                        $_SESSION["user_email"] = $dbEmail;
                        $_SESSION["password"] = $dbPassword; // Corrected
variable name
                        $_SESSION["user_role"] = $dbRole;

                        if ($enteredRole == "Seller") {
                            echo '<script>alert("Login Successful for
seller!");</script>';
                            header("Location: sellerhome.html");
                        } elseif ($enteredRole == "Buyer") {
                            echo '<script>alert("Login Successful for
buyer!");</script>';
                            header("Location: buyerhome.html");
                        } else {
                            throw new Exception("Invalid user role.");
                        }
                    } else {
                        throw new Exception("Incorrect Role");
                    }
```



```
                } else {
                    throw new Exception("Email not verified");
                }
            } else {
                throw new Exception("Incorrect email or password");
            }
        } else {
            throw new Exception("Incorrect email or password");
        }

        $stmt->close();
    } else {
        throw new Exception("Invalid request method");
    }
} catch (Exception $e) {
    echo "Caught exception: " . $e->getMessage();
} finally {
    if (isset($conn)) {
        $conn->close();
    }
}
```

★ Logging with Winston: Utilize the Winston logging library to capture errors and user activities in your application. Describe the benefits of proper logging in maintaining application security.

❖ Winston is a popular logging library for Node.js applications, designed to be flexible and efficient. In addition to assisting with problem identification and resolution, it offers a proactive method for continuously monitoring, identifying, and reacting to security concerns.

```
//app.js
const express = require('express');
const bodyParser = require('body-parser');
const winston = require('winston');
const MongoDB = require('winston-mongodb').MongoDB;

const app = express();
app.use(bodyParser.json());

// Configure Winston
const logger = winston.createLogger({
  level: 'error',
  format: winston.format.json(),
  transports: [
    new winston.transports.File({ filename: 'logs/error.log' }),
    // new MongoDB({
    //   db: 'mongodb://12210024gcit:qwertyuiop@cluster0.7eaqbsc.mongodb.net/',
    //   options: { useUnifiedTopology: true },
    //   collection: 'error_logs',
    // }),
  ],
});

// Endpoint to receive error logs from PHP
app.post('/logs', (req, res) => {
  try {
    const { level, message } = req.body;
    logger.log(level, message);
    res.status(200).send('Log received successfully.');
  } catch (error) {
    res.status(500).send('Internal Server Error');
  }
});

const PORT = 8080;
app.listen(PORT, () => {
  console.log(`Server listening on port ${PORT}`);
});
```

```
// login.php
$logData = [
        'level' => 'error',
        'message' => 'An error occurred in login.php: ' . $e->getMessage(),
    ];

    $nodeJsUrl = 'http://localhost:8080/Assignment/logs'; // Update with your
Node.js server URL
    $ch = curl_init($nodeJsUrl);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
    curl_setopt($ch, CURLOPT_POST, true);
    curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($logData));
    curl_setopt($ch, CURLOPT_HTTPHEADER, ['Content-Type: application/json']);
    curl_exec($ch);
    curl_close($ch);
```

● **SQL Injection:** **[10 Marks]**

★ Explain what SQL injection is and provide an example of how it can be exploited. In your application, implement secure coding techniques to prevent SQL injection, including the use of prepared statements.

- ❖ SQL injection is a type of cyber attack that occurs when an attacker inserts malicious SQL code into a query, which is then executed by a database. This may result in other harmful activities as well as unathorized access, data alteration, or deletion. SQL injection attacks may occur from inadequate validation or sanitization of user input prior to its inclusion in a SQL query.
- ❖ Here's a simple example of a vulnerable login form that is susceptible to SQL injection attacks due to improper input validation:

```
// Vulnerable SQL query
$username = $_POST['username'];
$password = $_POST['password'];
$sql = "SELECT * FROM users WHERE username='$username' AND
password='$password'";
```

- ❖ If an attacker enters the following in the username field:

```
' OR '1'='1'; --
```

- ❖ The resulting SQL query would be:

```
SELECT * FROM users WHERE username='' OR '1'='1'; --' AND password='';
```

This query always evaluates to true (1=1), allowing the attacker to bypass the login and potentially gain unauthorized access.

Prepared statements and parameterized queries are examples of secure coding practices that can help avoid SQL injection. Here is an example of how to use MySQLi with prepared statements in PHP:

- Here, the MySQLi connection's prepare method is used to construct a prepared statement. A placeholder (?) for the value that will be supplied later is present in the SQL query. Ensures that user input is treated as data and not as executable SQL code.

```
$sql = "SELECT email, password, status, role FROM users WHERE email = ?";
        $stmt = $conn->prepare($sql);
```

- The bind_param method is used to bind the parameter ($email) to the prepared statement. The letter "s" indicates that the parameter is a string. This step helps prevent SQL injection by ensuring that user input is properly escaped and treated as data.

```
$stmt->bind_param("s", $email);
```

- The execute method is then called to execute the prepared statement with the bound parameter.

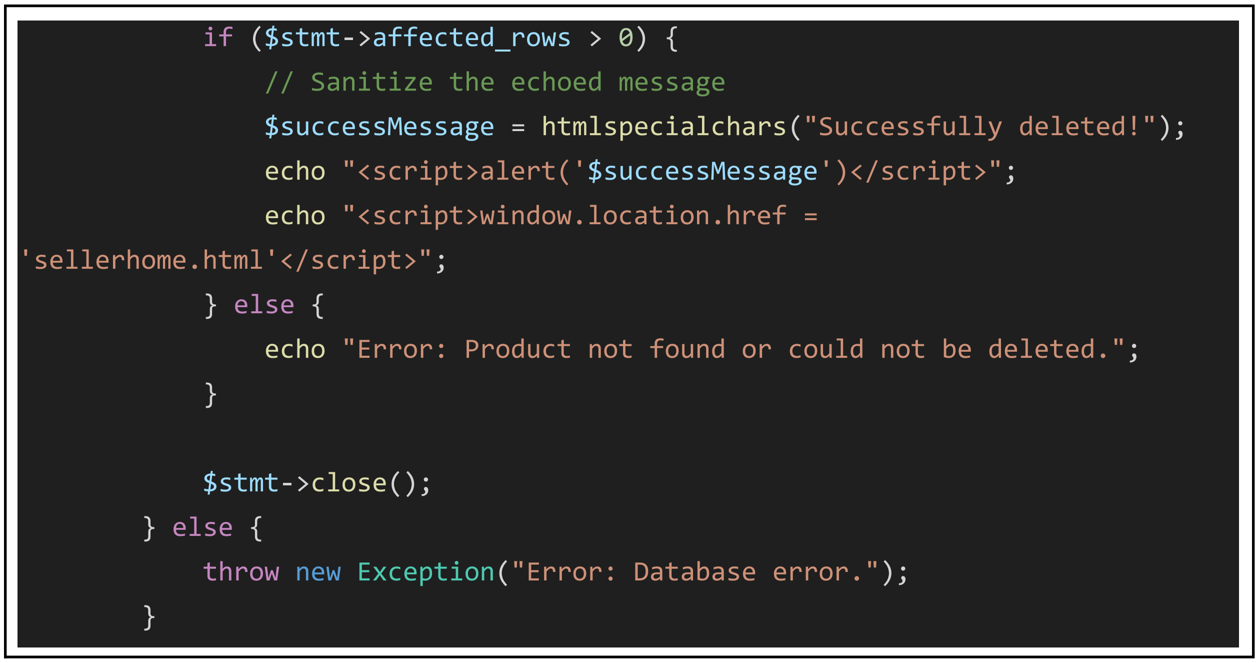
```
$stmt->execute();
```

- The bind_result method is used to bind the result columns to variables. This is followed by the fetch method to fetch the result.

```
$stmt->bind_result($dbEmail, $dbPassword, $status, $dbRole);
            $stmt->fetch();
```

❖ In the following code provide the usage of prepared statements and parameterized queries to delete product from database.

```
// Prepare and execute the SQL query to delete the product
        $sql = "DELETE FROM products WHERE productName = ?";
        $stmt = $conn->prepare($sql);

        if ($stmt) {
            $stmt->bind_param('s', $productName);
            $stmt->execute();

            // Check if the deletion was successful
```



```php
            if ($stmt->affected_rows > 0) {
                // Sanitize the echoed message
                $successMessage = htmlspecialchars("Successfully deleted!");
                echo "<script>alert('$successMessage')</script>";
                echo "<script>window.location.href =
'sellerhome.html'</script>";
            } else {
                echo "Error: Product not found or could not be deleted.";
            }

            $stmt->close();
        } else {
            throw new Exception("Error: Database error.");
        }
```

● **Cross-Site Scripting and Request Forgery:** **[10 Marks]**

★ Define what cross-site scripting is and describe various techniques to perform it. Implement secure coding techniques to prevent XSS and request forgery attacks in your application. Explain the role of input validation and output sanitization in preventing these attacks.

❖ Cross-Site Scripting (XSS) is a type of security vulnerability that occurs when an attacker injects malicious scripts into web pages viewed by other users. The victim's browser may run these scripts, which could result in harmful activity, theft of private data, or illegal access.

❖ Techniques to Perform XSS:

Several techniques can be employed to execute XSS attacks:

- Script Tags: Injecting malicious code within HTML <script> tags.

```
<!-- Input reflected in JavaScript without proper validation -->
<script>
    var userInput = 'USER_INPUT'; // Attacker injects malicious code here
 </script>
```

```
<!-- An attacker could try to inject a script by providing input like: -->
<script>alert('Malicious Code');</script>
```

```
<!-- If this input is echoed back into the JavaScript block without proper
validation or encoding, the resulting code would be: -->
<script>
    var userInput = '<script>alert('Malicious Code');</script>'; // Injected
script
  </script>
```

- Event Handlers: Exploiting JavaScript event handlers, such as onclick or onmouseover, to execute malicious code.

```
<!-- Suppose there's an HTML element with an onclick event that is dynamically
generated based on user input: -->
<button onclick="executeAction('USER_INPUT')">Click me</button>
```

```
<!-- If the application blindly includes user input in the onclick attribute
without proper validation or sanitization, an attacker might try to inject
malicious code: -->
"'); alert('Malicious Code'); // (Injected code)
```

```
<!-- If the user input is not properly validated or sanitized and is directly
inserted into the onclick attribute, the resulting HTML could look like this:
-->
<button onclick="executeAction(''); alert('Malicious Code'); //')">Click
me</button>
```

- HTML Attributes: Injecting malicious code into HTML attributes, like <img src="javascript:maliciousCode()">.

```
<!-- Suppose there's an HTML image element that dynamically generates the src
attribute based on user input: -->
<img src="user-input.jpg" alt="User Image">
```

```
<!-- If the application blindly includes user input in the src attribute
without proper validation or sanitization, an attacker might attempt to inject
malicious JavaScript code: -->
user-input.jpg" onerror="maliciousCode();" alt="User Image"
```

```
<!-- If the user input is not properly validated or sanitized and is directly
inserted into the src attribute, the resulting HTML could look like this: -->
<img src="user-input.jpg" onerror="maliciousCode();" alt="User Image">
```

- Data URLs: Embedding malicious code in data URLs within a link or image tag.

```
<!-- Suppose there's a link element where the href attribute is dynamically
generated based on user input: -->
<a href="user-input.html">Click me</a>
```

```
<!-- If the application blindly includes user input in the href attribute
without proper validation or sanitization, an attacker might try to inject a
data URL containing malicious JavaScript code: -->
user-input.html"><img src="data
```

```
<!-- If the application blindly includes user input in the href attribute
without proper validation or sanitization, an attacker might try to inject a
data URL containing malicious JavaScript code: -->
user-input.html"><img
src="data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAJYAAACWAQAAAAAbDI...
(malicious code)
.../KAAAAAAAAAAALAAAAAAjAAEAAAQIMRFI7MhAP0XYR2q51AAAAAElFTkSuQmCC">
```

- ❖ Secure Coding Techniques to Prevent XSS:
  - Input Validation: Validate and sanitize user inputs on both client and server sides. Ensure that only expected and valid input is accepted.

Server-side validation:

Ensures that the form is being submitted using the POST method.

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
        // Get product input (sanitize and validate as needed)
        $productName = htmlspecialchars($_POST["productName"]);
        $productDescription = htmlspecialchars($_POST["productDescription"]);
        $productPrice = floatval($_POST["productPrice"]); // Convert to float
for security
        $productQuantity = htmlspecialchars($_POST["productQuantity"]);
```

Sanitization:

The code uses htmlspecialchars to sanitize the input data obtained from the form. The code also converts the product price to a float using floatval to ensure it is a valid numeric value. This helps prevent Cross-Site Scripting (XSS) attacks by converting special characters to HTML entities.

```
$productName = htmlspecialchars($_POST["productName"]);
        $productDescription = htmlspecialchars($_POST["productDescription"]);
        $productPrice = floatval($_POST["productPrice"]); // Convert to float for security
        $productQuantity = htmlspecialchars($_POST["productQuantity"]);
```

Client-side validation:

```
// Email validation
    var email = document.getElementById('email').value;
    var emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    if (!emailRegex.test(email)) {
        alert('Please enter a valid email address.');
        return false;
    }
 // Password validation (at least 1 symbol, 1 number, and 1 capital letter)
    var password = document.getElementById('password').value;
    var passwordRegex = /^(?=.*[!@#$%^&*()_+\-=\[\]{};':"\\|,.<>\/?])(?=.*[0-9])(?=.*[A-Z]).{8,}$/;

    if (!passwordRegex.test(password)) {
        alert('Please enter a valid password with at least 1 symbol, 1 number, and 1 capital letter.');
        return false;
    }
```

❖ Role of Input Validation and Output Sanitization:

- Input Validation: Validates user inputs to ensure they conform to expected formats and ranges. This helps prevent injection attacks, including XSS, by rejecting or sanitizing inputs that contain malicious code.
- Output Sanitization: Sanitizes and encodes output before rendering it in HTML to ensure that any user-generated content does not execute as code in the browser. This helps in preventing XSS attacks by neutralizing the impact of injected scripts.

By combining these techniques, developers can significantly reduce the risk of XSS attacks in their applications. It's crucial to stay informed about the latest security best practices and update security measures regularly to address emerging threats

● **Authentication and Authorization:** **[10 Marks]**

★ Implement user authentication in your application. Explain how it works and how it contributes to overall security.

- Implementing user authentication is a crucial aspect of building secure applications. It involves verifying the identity of users attempting to access a system or application.
- How User Authentication Works:
  - ***User Registration***:
    Users create an account by providing a unique username or email and a password.

```
// Email is the unique attribute
    var email = document.getElementById('email').value;
    var emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    if (!emailRegex.test(email)) {
        alert('Please enter a valid email address.');
        return false;
    }
```

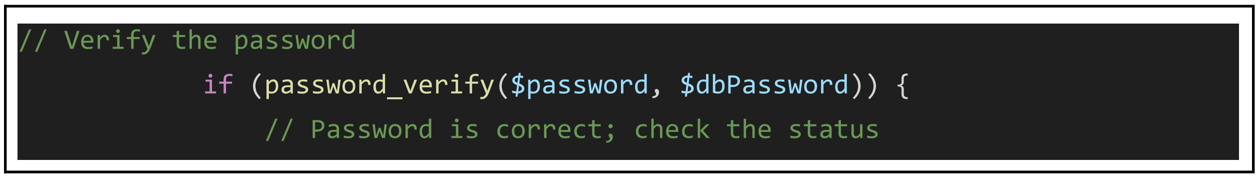
  - ***Credential Storage***:
    Passwords are securely hashed and stored in a database.

```
$password = password_hash($_POST['password'], PASSWORD_BCRYPT); // Hash the
password for security
```

  - ***Login Process:***
    Users enter their credentials (email and password) during the login process. The application retrieves the hashed password from the database using the provided email.

```
$sql = "SELECT email, password, status, role FROM users WHERE email = ?";
if (password_verify($password, $dbPassword)) // dbPassword is the hashed
password
```

- ***Password Verification***:
  The entered password is hashed using the same algorithm used during registration.The hashed input is compared with the stored hashed password. If they match, the user is granted access.



```
// Verify the password
            if (password_verify($password, $dbPassword)) {
                // Password is correct; check the status
```

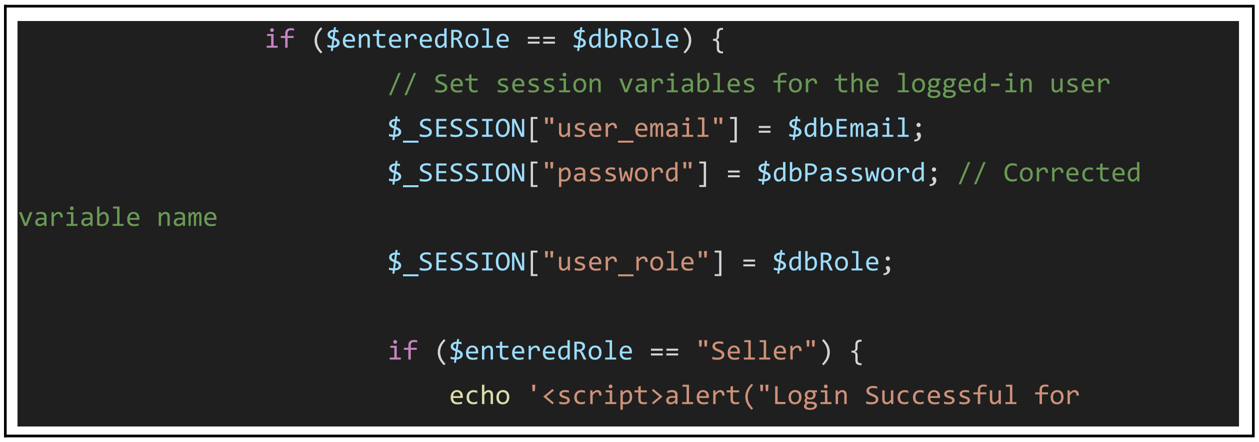
- ***Two Factor Authentication***
  2FA is a subset of MFA, using two authentication factors. MFA employs multiple distinct factors to bolster security. For example, in my assignment common MFA method involves a email and password (knowledge) and an auto-generated OTP sent to the email (possession).

★ Define user roles and authorization rights within your application. Implement role-based authorization to restrict access to certain parts of your e-commerce platform.

❖ In an e-commerce platform, defining user roles and implementing role-based authorization is crucial for managing access to different parts of the system.

❖ User roles and authorization

- ***Buyer***
  Basic role for users who can view products, and place orders.



```
                if ($enteredRole == $dbRole) {
                        // Set session variables for the logged-in user
                        $_SESSION["user_email"] = $dbEmail;
                        $_SESSION["password"] = $dbPassword; // Corrected
variable name
                        $_SESSION["user_role"] = $dbRole;

                        if ($enteredRole == "Seller") {
                            echo '<script>alert("Login Successful for
seller!");</script>';
                            header("Location: sellerhome.html");
                        } elseif ($enteredRole == "Buyer") {
                            echo '<script>alert("Login Successful for
buyer!");</script>';
                            header("Location: buyerhome.html");
                        } else {
                            throw new Exception("Invalid user role.");
                        }
                    } else {
                        throw new Exception("Incorrect Role");
                    }
```

- ***Seller***

  Basic role for users who can add products, and delete product.

```
 if ($enteredRole == $dbRole) {
                        // Set session variables for the logged-in user
                        $_SESSION["user_email"] = $dbEmail;
                        $_SESSION["password"] = $dbPassword; // Corrected
variable name
                        $_SESSION["user_role"] = $dbRole;

                        if ($enteredRole == "Seller") {
                            echo '<script>alert("Login Successful for
seller!");</script>';
                            header("Location: sellerhome.html");
                        } elseif ($enteredRole == "Buyer") {
                            echo '<script>alert("Login Successful for
buyer!");</script>';
                            header("Location: buyerhome.html");
                        } else {
                            throw new Exception("Invalid user role.");
                        }
                    } else {
                        throw new Exception("Incorrect Role");
                    }
```

● **Session Management:** **[10 Marks]**

★ Describe the role of session management in web applications and explain how session hijacking can occur. Implement secure session management in your e-commerce Application.

❖ Session management is a crucial aspect of web application security that involves the creation, maintenance, and termination of user sessions. A session, which usually begins when a user logs in and ends when they log out or after a period of inactivity, is a means to uniquely identify and trace a user's interactions with a web application throughout a certain period of time.

❖ The primary goals of session management in web applications are:
  - Authentication: Verifying the identity of users during login.
  - Authorization: Granting appropriate access rights to authenticated users.
  - Data integrity: Ensuring that user data is not tampered with during the session.
  - Confidentiality: Protecting sensitive information from unauthorized access.

❖ Session hijacking
Session hijacking, also known as session theft or session stealing, occurs when an attacker gains unauthorized access to a user's session. This can lead to severe security breaches, unauthorized access to sensitive information, and potential misuse of the user's account. Several methods can be employed to hijack sessions:
  - Session Sniffing: Attackers intercept network traffic to capture session data, often achieved through techniques like packet sniffing on unsecured Wi-Fi networks.
  - Session Sidejacking: Also known as session hijacking over non-encrypted connections, attackers use packet sniffers to capture session cookies transmitted over unsecured HTTP connections.
  - Cross-Site Scripting (XSS): Malicious scripts injected into web pages can steal session information if the application does not properly validate and sanitize user inputs.
  - Session Fixation: Attackers set or fix a user's session ID, either by predicting the session ID or forcing the user to use a predetermined session ID.
  - Man-in-the-Middle (MitM) Attacks: An attacker intercepts communication between the user and the server to capture session data.

❖ Added session_start() at the beginning to start or resume the session.Checked if the user is logged in and has the role of a seller or buyer before allowing access to the product listing functionality.

```
// Check if the user is logged in as a seller
    if (!isset($_SESSION['user_email']) || $_SESSION['user_role'] !== 'Seller')
{
        // Redirect to the login page for unauthorized users
        header("Location: login.html    ");
        exit();
    }
```

```
// Check if the user is logged in as a buyer
        if ($_SESSION['user_role'] !== 'Buyer') {
            // Redirect to the login page for unauthorized users
            header("Location: login.html");
            exit();
        }
```

- Checked if the logged-in seller is the owner of each product before allowing deletion. If not, the "Delete" button is disabled for non-owners.

```
// Check if the logged-in user is the owner of the product
             if ($_SESSION['user_role'] === 'Seller') {
                // Display the order button
                echo "<button type='submit'>Delete</button>";
            }
```

Discuss any challenges you encountered during development and how you resolved them.

- ❖ Firstly, I was developing website using sql and php so it was little difficult as I was learning it for the first time. But with time and with the help of chatGPT I was able to do it somehow.
- ❖ I had difficulties with OTP generation and setting the time for it. And i learnt it was simple spelling mistake that was unnoticed. And I seek helped a friend to help me generate the OTP and steps involved.
- ❖ I had trouble deciding how to make the authorization and decided to make role-based authorization where seller have rights to delete and add product. And where buyer has rights to view and order product.
- ❖ In order product, I had difficulty fetching already existing data from database. I solved that by adding Order button with a link to the order HTML page.
- ❖ In error logging i tried to use winston to log error, i tried to store in sql too, but i couldn’t. So i tried to store in mongodb and used session id in error logging.
- ❖ Though it was not a fully functional website, somehow I tried manage secure website