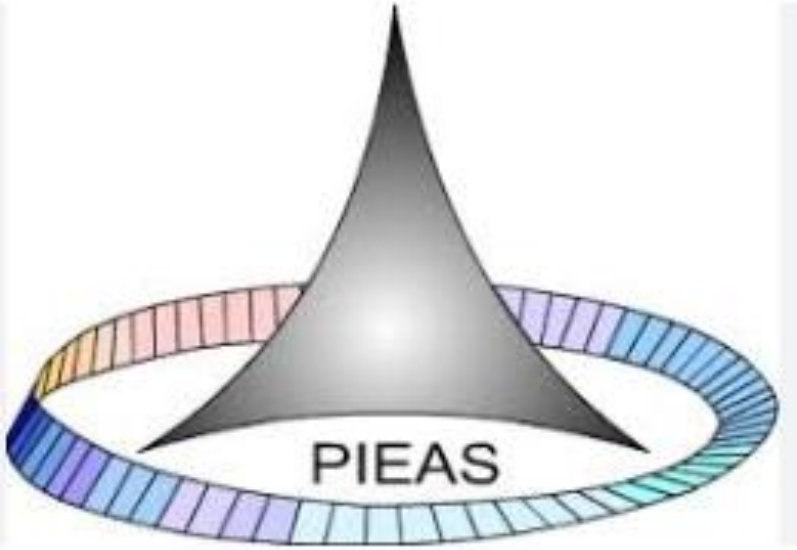
NAME: INAYAT ARSHAD

SUBMITTED TO: DR IRFAN HAMEED

IAD SEMESTER PROJECT

PROBLEM 3



(Problem 3)

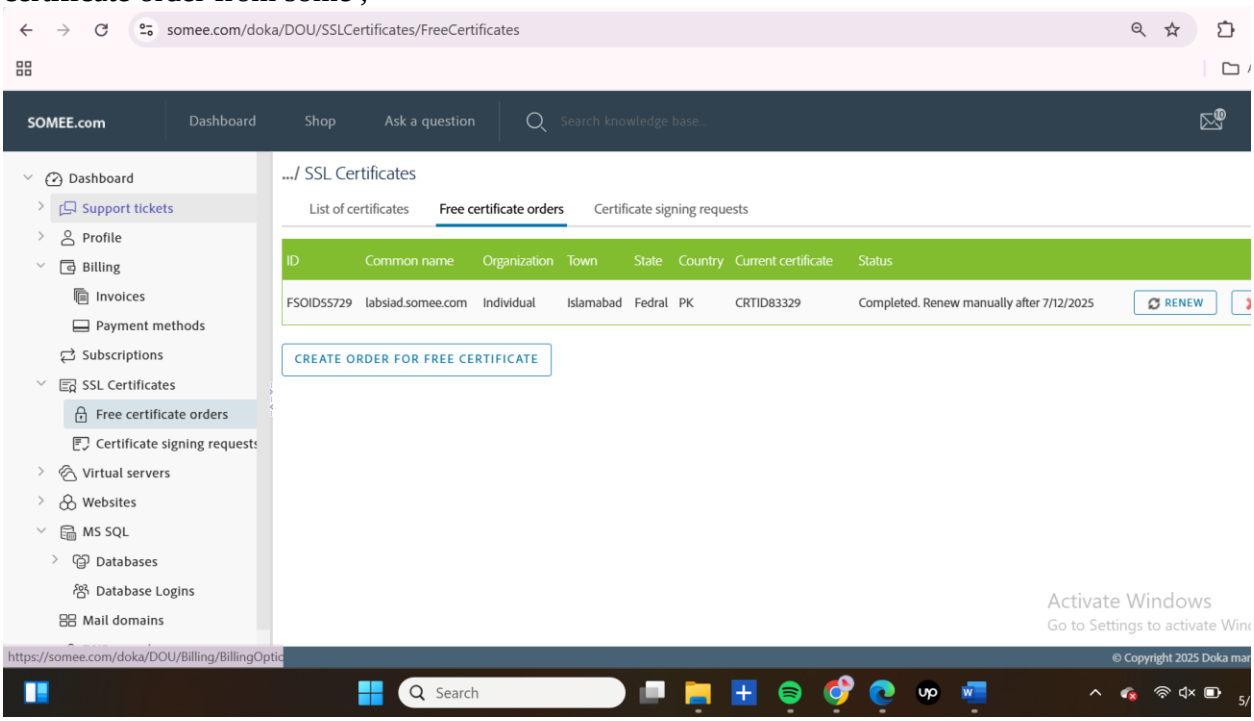
Develop test cases for all security features and prepare a report about testing security features.

(Ans)

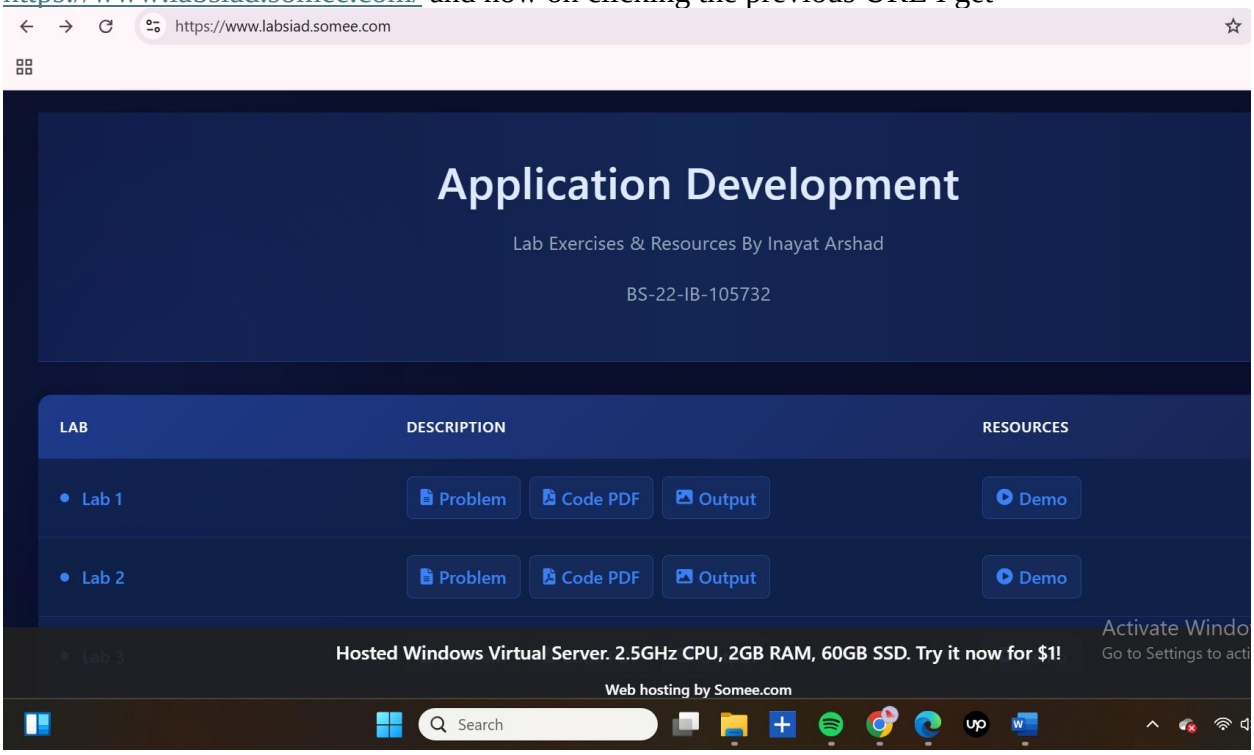
The following test cases are used for the security features I have applied:

1)Https redirection

To enhance the security of my ASP.NET web application, I implemented HTTPS redirection to ensure that all client-server communication is encrypted. Firstly , my site was not secured due to http only , no SSL , the URL was http://labsiad.somee.com but after that I bought a free certificate order from some ,



And then I was able to , use Https and system got secured , now my URL is https://www.labsiad.somee.com/ and now on clicking the previous URL I get



SSL (Secure Sockets Layer) provides a secure, encrypted connection between a user's browser and the web server, ensuring that all data transmitted—such as login credentials, personal details, or payment information—is protected from interception or tampering by hackers. It prevents **man-in-the-middle attacks**, ensures **data integrity**, and builds **user trust** by displaying the padlock icon and "https://" in the address bar. Additionally, SSL is essential for **SEO rankings**, as search engines like Google prioritize secure websites.

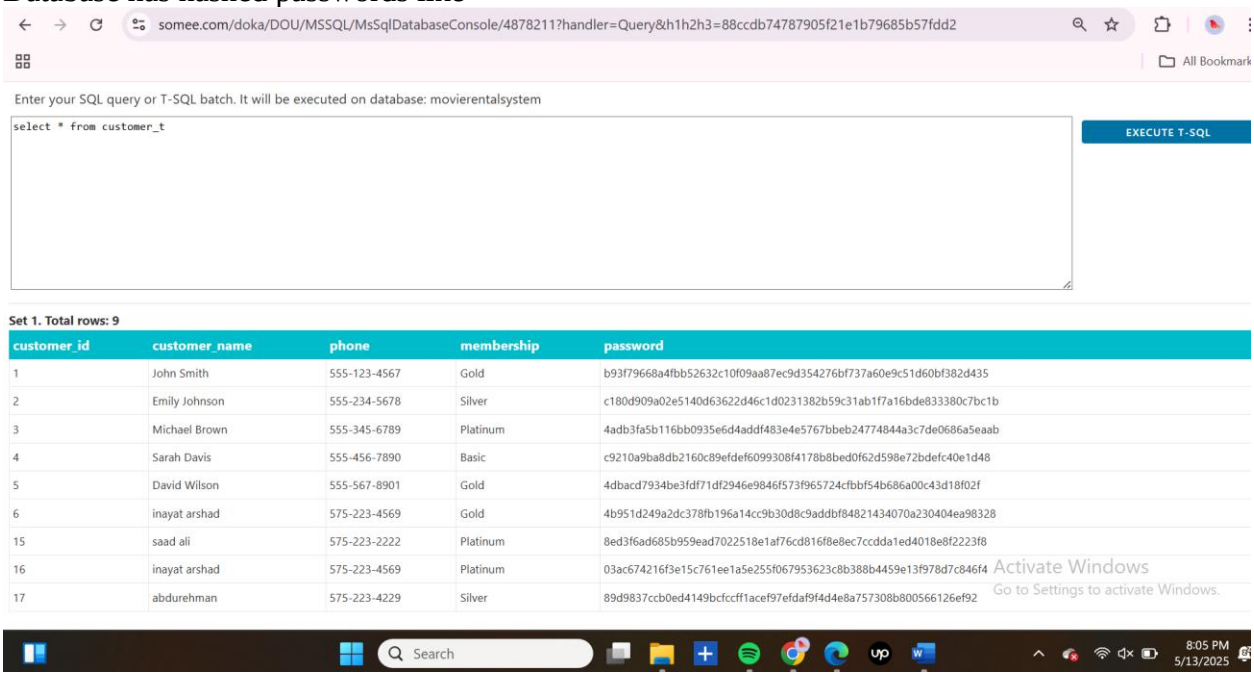
2) Secure Password Storage

When a user types a password like inayat123 during login:

1. Login.aspx.vb **hashes** that input using SHA-256.
2. The hashed version (e.g., 25f9e794323b453885f5181f1b624d0b) is compared with the **stored hashed password** in the database.
3. If it **matches**, login is successful.

You just type the plain password (inayat123) as usual — no change in user behavior.

Database has hashed passwords like



2 files has been added

HashPasswords.aspx

<%@ Page Language="VB" AutoEventWireup="false" CodeFile="HashPasswords.aspx.vb" Inherits="HashPasswords" %>

```
<!DOCTYPE html>
<html>
<head runat="server">
    <title>Hash Passwords Utility</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Label ID="lblStatus" runat="server" Text="Hashing passwords... Please wait."></asp:Label>
        </div>
    </form>
</body>
</html>
```

HashPasswords.aspx.vb

```
Imports System
Imports System.Data.SqlClient
Imports System.Security.Cryptography
Imports System.Text

Partial Class HashPasswords
    Inherits System.Web.UI.Page
```

```
    Dim connStr As String = "workstation id=movierentalsystem.mssql.somee.com;packet size=4096;user
id=inayat_arshad_SQLLogin_1;pwd=qhlgsg2nl9;data
source=movierentalsystem.mssql.somee.com;persist security info=False;initial
catalog=movierentalsystem;TrustServerCertificate=True"

    Protected Sub Page_Load(sender As Object, e As EventArgs) Handles Me.Load
        Try
            Dim customersUpdated As Integer = 0

            Using conn As New SqlConnection(connStr)
                conn.Open()

                ' Select all customers
                Dim selectCmd As New SqlCommand("SELECT customer_id, password FROM customer_t",
conn)
                Dim reader As SqlDataReader = selectCmd.ExecuteReader()

                Dim customersToUpdate As New List(Of KeyValuePair(Of Integer, String))()

                While reader.Read()
                    Dim customerId As Integer = Convert.ToInt32(reader("customer_id"))
                    Dim plainPassword As String = reader("password").ToString()

                    ' Check if already hashed (hashes are usually 64 characters long in hex)
                    If plainPassword.Length <> 64 Then
                        Dim hashedPassword As String = HashPassword(plainPassword)
                        customersToUpdate.Add(New KeyValuePair(Of Integer, String)(customerId,
hashedPassword))
                    End If
                End While
                reader.Close()

                ' Update passwords
                For Each customer In customersToUpdate
                    Dim updateCmd As New SqlCommand("UPDATE customer_t SET password = @hashed
WHERE customer_id = @id", conn)
                    updateCmd.Parameters.AddWithValue("@hashed", customer.Value)
                    updateCmd.Parameters.AddWithValue("@id", customer.Key)
                    updateCmd.ExecuteNonQuery()
                    customersUpdated += 1
                Next

                lblStatus.Text = "Passwords hashed successfully for {customersUpdated} customer(s). Please
delete this page now."
            End Using

        Catch ex As Exception
            lblStatus.Text = " Error: " & ex.Message
        End Try
    End Sub
```

```
    Private Function HashPassword(password As String) As String
        Using sha256 As SHA256 = SHA256.Create()
            Dim bytes As Byte() = Encoding.UTF8.GetBytes(password)
            Dim hash As Byte() = sha256.ComputeHash(bytes)
            Dim sb As New StringBuilder()
            For Each b As Byte In hash
                sb.Append(b.ToString("x2"))
            Next
            Return sb.ToString()
        End Using
    End Function
End Class
```

login.aspx.vb

```
Imports System
Imports System.Data.SqlClient
Imports System.Security.Cryptography
Imports System.Text

Partial Class Login
    Inherits System.Web.UI.Page

    Dim connStr As String = "workstation id=movierentalsystem.mssql.somee.com;packet size=4096;user
id=inayat_arshad_SQLLogin_1;pwd=qhlgsg2nl9;data
source=movierentalsystem.mssql.somee.com;persist security info=False;initial
catalog=movierentalsystem;TrustServerCertificate=True"

    Protected Sub btnLogin_Click(sender As Object, e As EventArgs)
        If String.IsNullOrWhiteSpace(txtUsername.Text) OrElse
String.IsNullOrWhiteSpace(txtPassword.Text) Then
            lblMessage.Text = "Username and Password are required."
            Return
        End If

        Dim hashedInputPassword As String = HashPassword(txtPassword.Text)

        Using conn As New SqlConnection(connStr)
            conn.Open()

            Dim cmd As New SqlCommand("SELECT customer_id, customer_name FROM customer_t
WHERE customer_name = @name AND password = @password", conn)
            cmd.Parameters.AddWithValue("@name", txtUsername.Text)
            cmd.Parameters.AddWithValue("@password", hashedInputPassword)

            Dim reader As SqlDataReader = cmd.ExecuteReader()

            If reader.Read() Then
                Session("username") = reader("customer_name").ToString()
                Session("customer_id") = reader("customer_id").ToString()
                Response.Redirect("dashboard.aspx")
            Else
```

```
            lblMessage.Text = "Invalid username or password."
        End If
        reader.Close()
    End Using
End Sub

Private Function HashPassword(password As String) As String
    Using sha256 As SHA256 = SHA256.Create()
        Dim bytes As Byte() = Encoding.UTF8.GetBytes(password)
        Dim hash As Byte() = sha256.ComputeHash(bytes)
        Dim sb As New StringBuilder()
        For Each b As Byte In hash
            sb.Append(b.ToString("x2"))
        Next
        Return sb.ToString()
    End Using
End Function
End Class
```

After adding HashPasswords.aspx ,I Deployed and visit https://labsiad.somee.com/HashPasswords.aspx once, It will hash all non-hashed passwords in the customer_t table. After confirming login works:
Delete the page immediately

Functionality:

In my web application, I implemented **password hashing** to enhance the security of the login system. Specifically, I used the **SHA-256 hashing algorithm** in VB.NET to hash the admin password before verifying it during login.

This had the following functions and advantages:

- **Prevented plain-text password storage**: Instead of storing or comparing passwords in plain text, I used a hashing function to transform the password into a fixed-length, irreversible string.
- **Improved security in case of data breaches**: Even if someone gains unauthorized access to the application or database, they won't be able to see the actual password.
- **Resisted reverse-engineering**: The one-way nature of the SHA-256 algorithm ensures that it's practically impossible to retrieve the original password from the hashed version.
- **Protected against common attacks**: Hashing helps defend against **brute force** and **rainbow table attacks**, especially when used with additional techniques like salting (which can be added later for even stronger protection).

By doing this, I ensured that sensitive login information, like admin credentials, is handled in a secure and industry-standard way.

3) Role-Based Access Control (RBAC)

Purpose:

RBAC was implemented to ensure that users (Admin and Customers) can only access functionality appropriate to their roles, thereby enforcing security, personalization, and access management.

How I Implemented RBAC

I implemented Role-Based Access Control (RBAC) by clearly separating the authentication process and access logic for two types of users: **Admin** and **Customer**. Each role has a distinct login system, session control, and access to role-specific pages.
On the welcome page, users are prompted to select their role:

- Clicking **Admin** redirects the user to AdminLogin.aspx
- Clicking **Customer** redirects the user to Login.aspx (customer login)

This initial choice ensures role-specific routing from the very beginning.

2. Admin Authentication and Access Control

a. AdminLogin.aspx.vb

- The admin credentials are hardcoded for simplicity (admin / admin123).
- The password is hashed using SHA-256 using a GetHashedPassword() function.
- Upon successful login, a session variable is set:
  Session("admin") = "true"
- The admin is redirected to AdminDashboard.aspx.

b. AdminDashboard.aspx

Admin-only features:

  - View all customers
  - Manage movies
  - Manage subscriptions
  - Logout

- Access to AdminDashboard.aspx and its features is protected using a session check

```
If Session("admin") Is Nothing OrElse Session("admin") <> "true" Then
    Response.Redirect("AccessDenied.aspx")
End If
```

3. Customer Authentication and Access Control

a. Login.aspx.vb

- Customers enter their username and password.
- The password is hashed using SHA-256 and validated against the database (customer_t table).
- Upon successful login:

```
Session("username") = reader("customer_name").ToString()
Session("customer_id") = reader("customer_id").ToString()
```

- The user is redirected to dashboard.aspx (customer dashboard).

b. CustomerDashboard.aspx

- Only customer features are shown:
  - Browse movies
  - Rent movies
  - Review payment history
- Customer pages check for a valid session before allowing access:

```
If Session("customer_id") Is Nothing Then
    Response.Redirect("AccessDenied.aspx")
End If
```

4. Role Isolation

- Admin and Customer dashboards are completely isolated in both logic and navigation.

- Users cannot simply change the URL and access another role's dashboard.
    - For example, a customer trying to access AdminDashboard.aspx without the correct session will be redirected.
- Logout pages clear session data to prevent reuse of old sessions.

Outcome

This role-based access system in my Movie Rental System:

- Prevents customers from accessing admin-only functionalities such as managing movies or viewing all customers.
- Provides a personalized experience where **admins** can manage data and **customers** can only browse, rent movies, and view their payment history.
- Keeps the application secure and organized by clearly separating **admin operations** from **customer activities**, both in logic and page access.

4) SQL Injection Prevention

In the implementation of the login functionality, **SQL Injection Prevention** has been effectively ensured using **parameterized SQL queries**. Rather than directly concatenating user inputs into SQL queries, the system employs parameterized queries to safely pass user input as data, not executable code. For example, in the btnLogin_Click event handler, the query to validate user credentials is written as:

```
Imports System
Imports System.Data.SqlClient
Imports System.Security.Cryptography
Imports System.Text

Partial Class Login
    Inherits System.Web.UI.Page

    Dim connStr As String = "workstation id=movierentalsystem.mssql.somee.com;packet size=4096;user
id=inayat_arshad_SQLLogin_1;pwd=qhlgsg2nl9;data
source=movierentalsystem.mssql.somee.com;persist security info=False;initial
catalog=movierentalsystem;TrustServerCertificate=True"

    Protected Sub btnLogin_Click(sender As Object, e As EventArgs)
        If String.IsNullOrWhiteSpace(txtUsername.Text) OrElse
String.IsNullOrWhiteSpace(txtPassword.Text) Then
            lblMessage.Text = "Username and Password are required."
            Return
        End If

        Dim hashedInputPassword As String = HashPassword(txtPassword.Text)

        Using conn As New SqlConnection(connStr)
            conn.Open()

            Dim cmd As New SqlCommand("SELECT customer_id, customer_name FROM customer_t
WHERE customer_name = @name AND password = @password", conn)
            cmd.Parameters.AddWithValue("@name", txtUsername.Text)
            cmd.Parameters.AddWithValue("@password", hashedInputPassword)
```

```
            Dim reader As SqlDataReader = cmd.ExecuteReader()

            If reader.Read() Then
                Session("username") = reader("customer_name").ToString()
                Session("customer_id") = reader("customer_id").ToString()
                Response.Redirect("dashboard.aspx")
            Else
                lblMessage.Text = "Invalid username or password."
            End If
            reader.Close()
        End Using
    End Sub

    Private Function HashPassword(password As String) As String
        Using sha256 As SHA256 = SHA256.Create()
            Dim bytes As Byte() = Encoding.UTF8.GetBytes(password)
            Dim hash As Byte() = sha256.ComputeHash(bytes)
            Dim sb As New StringBuilder()
            For Each b As Byte In hash
                sb.Append(b.ToString("x2"))
            Next
            Return sb.ToString()
        End Using
    End Function
End Class
```

Here, the user input for the username and password fields is bound to the SQL parameters @name and @password, preventing any malicious SQL code from being injected. The parameters are safely added to the SQL command using cmd.Parameters.AddWithValue(), ensuring that the values are treated as data rather than part of the SQL syntax. This approach eliminates the risk of SQL injection attacks, as user input cannot manipulate the query logic or access unauthorized data. By adopting parameterized queries, the system robustly secures the application from SQL injection vulnerabilities and ensures secure database interactions.

5) Session Management Implementation

Purpose:

The goal of session management is to ensure that users remain securely logged in during their session and to prevent attackers from hijacking active sessions. Proper session management includes using secure cookies, session timeouts, and proper session invalidation upon logout.

Setting Secure, HTTP-Only Cookies:

- After a successful login, the session is established, and the session cookie (ASP.NET_SessionId) is set with two critical attributes:
    - **HttpOnly:** Ensures that the cookie is not accessible via client-side JavaScript, thus preventing cross-site scripting (XSS) attacks.
    - **Secure:** Ensures that the cookie is only sent over HTTPS, preventing it from being transmitted over insecure HTTP connections.
- This is implemented in the btnLogin_Click method in both AdminLogin.aspx.vb and Login.aspx.vb files:

```
If Response.Cookies("ASP.NET_SessionId") IsNot Nothing Then
```

```
    Response.Cookies("ASP.NET_SessionId").HttpOnly = True
    Response.Cookies("ASP.NET_SessionId").Secure = True
End If
```

Session Timeout:

- By default, ASP.NET session management has a built-in session timeout feature that invalidates the session after a predefined period of inactivity. However, you can adjust the timeout value in the Web.config file to suit your security needs. For example:

```
<configuration>
    <system.web>
        <sessionState timeout="20" /> <!-- Session timeout in minutes -->
    </system.web>
</configuration>
```

This ensures that if a user does not interact with the application for 20 minutes, the session will be automatically invalidated.

Session Invalidation on Logout:

- When the user logs out, the session is invalidated using the Session.Abandon() method, which clears the session data and prevents unauthorized access to sensitive information.

For example, in a logout page, i would implement:

```
Session.Abandon()
Response.Redirect("Login.aspx")
```

Role-Based Session Management:

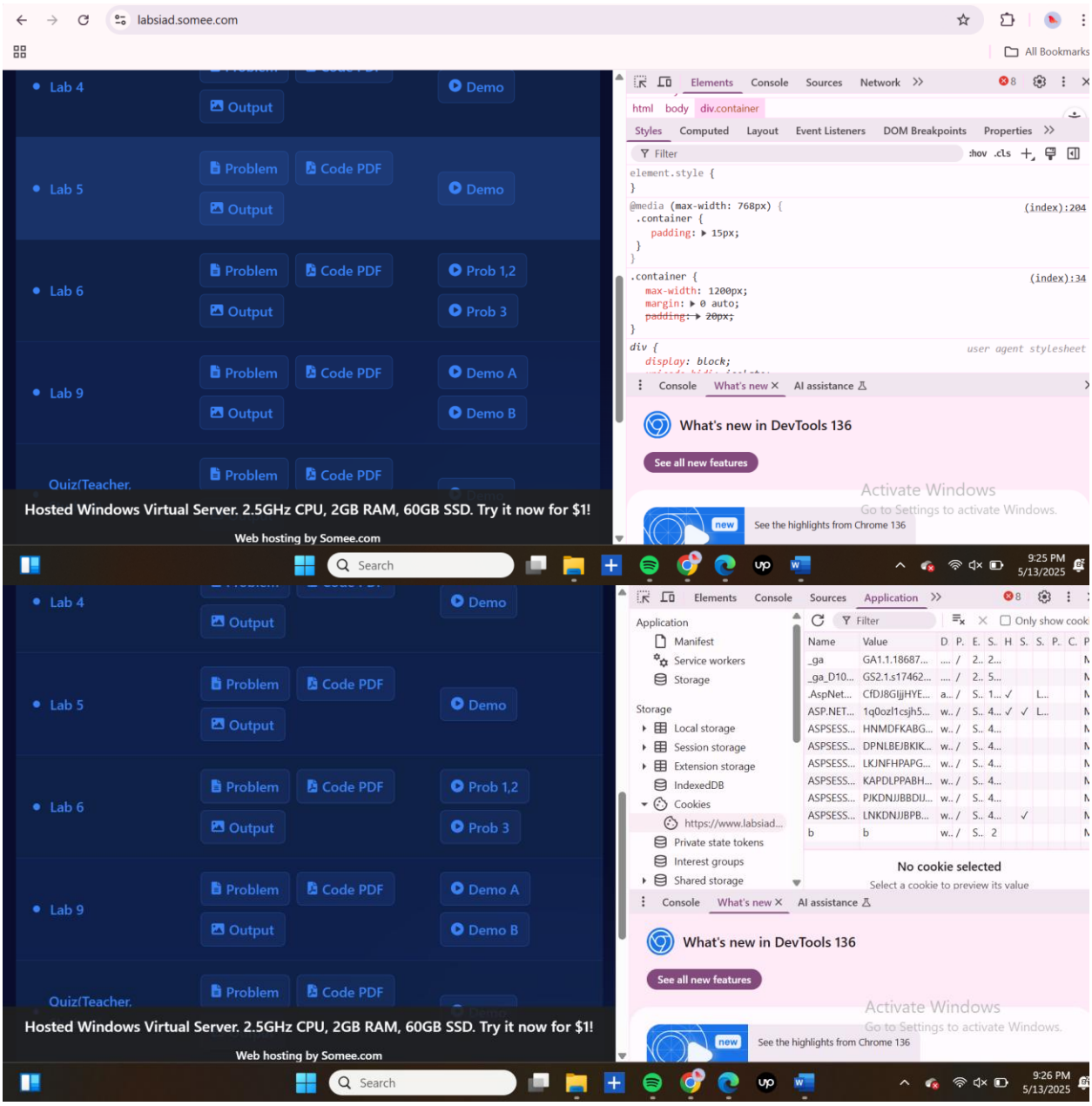
- When the user successfully logs in (either as admin or customer), the session stores the username and customer/admin status:

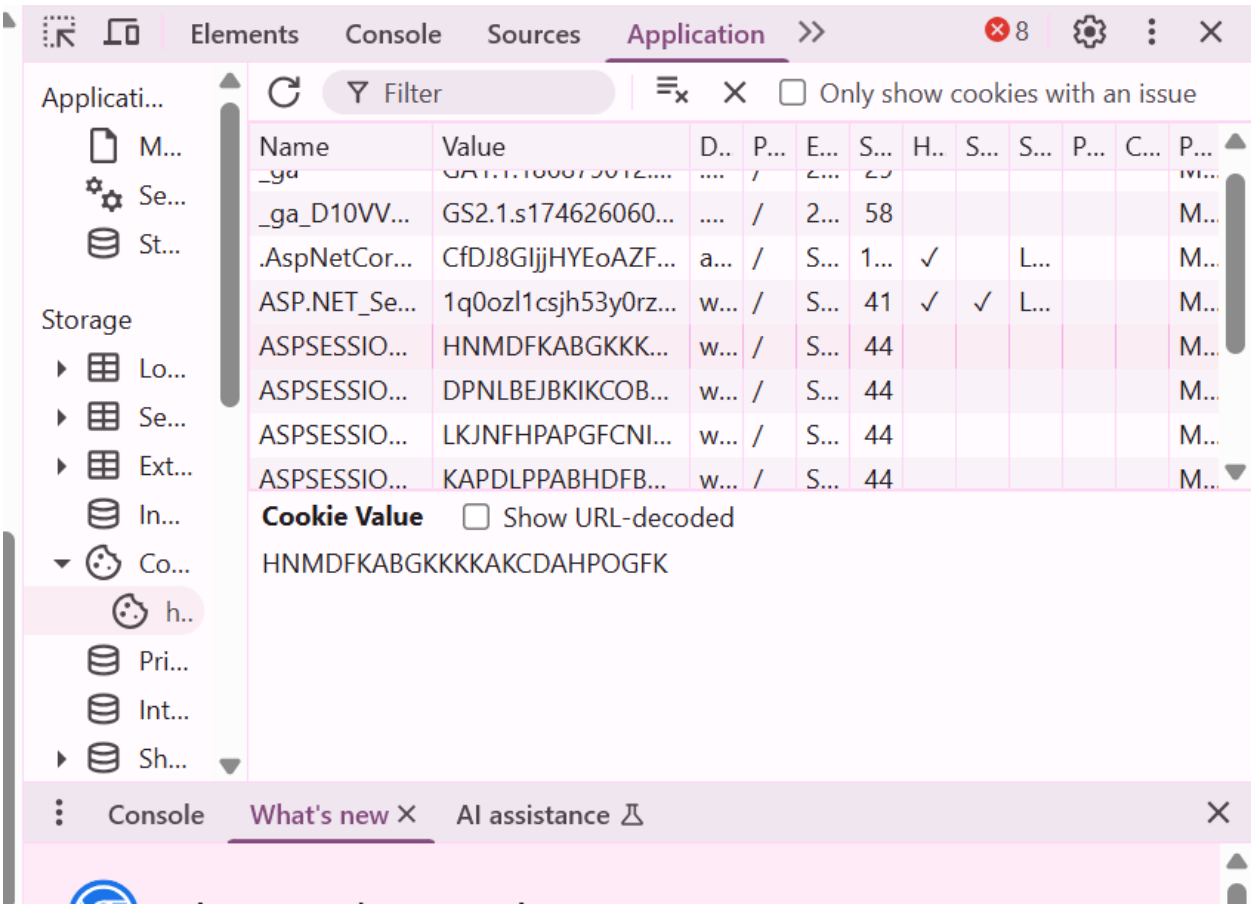
```
Session("username") = reader("customer_name").ToString()
Session("customer_id") = reader("customer_id").ToString()
```

TEST CASE FOR TESTING SESSION MANAGEMENT:

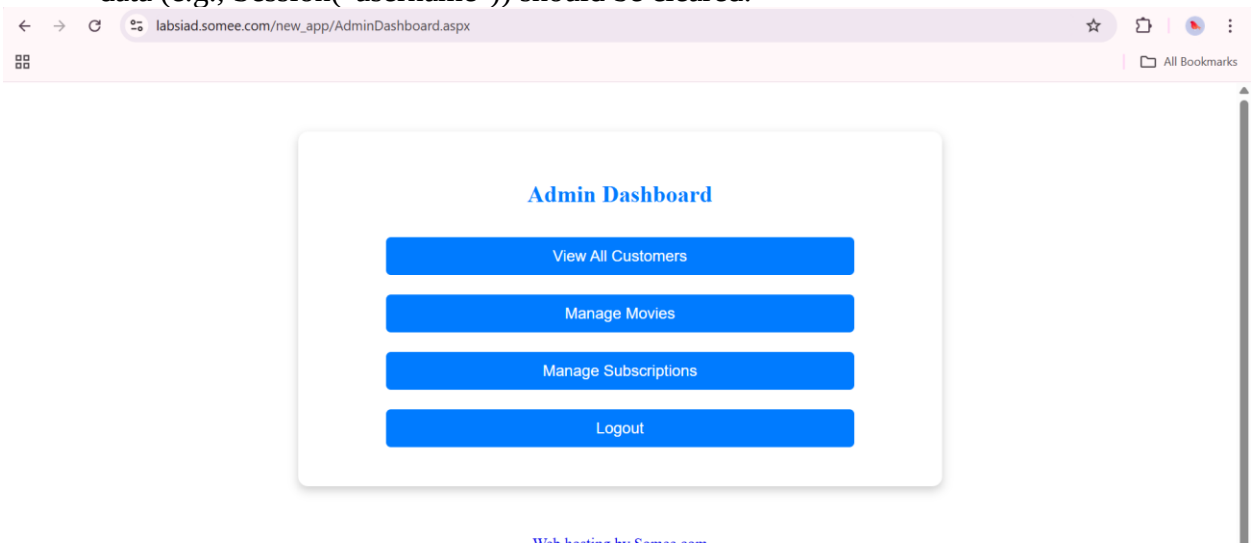
Testing Secure Cookies:

- **Step 1:** Log in to the application using valid credentials.
- **Step 2:** Open your browser's developer tools (right-click > Inspect > Application tab).
- **Step 3:** Look for the session cookie (ASP.NET_SessionId) under the "Cookies" section.
- **What happens?** The session cookie should have both the HttpOnly and Secure flags enabled, ensuring that the cookie is not accessible via JavaScript and is sent only over HTTPS.


labsiad.somee.com
Lab 4
Demo
Output
Lab 5
Problem
Code PDF
Demo
Output
Lab 6
Problem
Code PDF
Prob 1,2
Output
Prob 3
Lab 9
Problem
Code PDF
Demo A
Output
Demo B
Quiz/Teacher.
Problem
Code PDF
Hosted Windows Virtual Server. 2.5GHz CPU, 2GB RAM, 60GB SSD. Try it now for $1!
Web hosting by Somee.com
Elements
Console
Sources
Network
html
body
div.container
Styles
Computed
Layout
Event Listeners
DOM Breakpoints
Properties
Filter
element.style {
}
@media (max-width: 768px) {
.container {
padding: 15px;
}
}
(index):204
.container {
max-width: 1200px;
margin: 0 auto;
padding: 20px;
}
(index):34
div {
display: block;
user agent stylesheet
Console
What's new
AI assistance
What's new in DevTools 136
See all new features
new
See the highlights from Chrome 136
Activate Windows
Go to Settings to activate Windows.
Search
9:25 PM
5/13/2025
Application
Manifest
Service workers
Storage
Local storage
Session storage
Extension storage
IndexedDB
Cookies
Private state tokens
Interest groups
Shared storage
No cookie selected
Select a cookie to preview its value
9:26 PM
5/13/2025



- **Testing Session Invalidation on Logout:**
  - **Step 1:** Log in to the application.
  - **Step 2:** Log out by clicking the logout button.
  - **Step 3:** After logout, try to access a restricted page.
  - **Expected Behavior:** The user should be redirected to the login page, and any session data (e.g., Session("username")) should be cleared.



So, By implementing secure session management (HTTP-only and Secure cookies, session timeouts, and proper invalidation on logout), I've enhanced the security of the application and minimized the risk of session hijacking. Additionally, by hashing passwords, I have ensured that sensitive data such as user credentials is protected even if the database is compromised.

----------END----------