COMP 241 Lab #5. Web Application Development

Working with a partner, use Visual Studio to create a web application that manages state, queries a database, implements security, and uses caching to improve performance.

Relevant textbook chapters: Chapters 8, 14, 19, and 23!

Part 1: HTTP Headers, and Cookies.

Read chapter 8, paying close attention to the section on Cookies (p. 249).

- Create a new empty web application in Visual Studio.
- Add a new web form called Headers. In the Page_Load(…) method of Headers.aspx.cs, iterate through the HTTP headers and display their keys and associated values. The headers are in the Request.Header property, and are a key-value mapping from string to string. Use the Response.Write() method to generate appropriate HTML that contains the header info.
- Test your Headers.aspx. View source and notice that the HTML generated by Response.Write() happened before the rest of the ASP.NET processing.
- Add a new web form called Login. Make this the start page for your application by right-clicking on Login.aspx in the Solution Explorer and "Set As Start Page". Build a simple Login.aspx authentication web form that prompts the user for a username and password.
- Set the password textbox `TextMode` property to `TextBoxMode.Password`.
- Write a method that takes the username (maximum of 16 characters) and generates an encoding suitable to use as a difficult to spoof cookie value. Your encoding should be two-way so you can generate the username from the encoded version. Feel free to use Base64 encoding/decoding (google it for an example of how to do this in C#), or devise your own mechanism.
- Change your Login.aspx page so that it in response to the button click event your page creates a cookie. The cookie should be called "login" and the value should be the entered username run through your encoding function. Store that cookie on the browser, with an expiry date one week into the future.
- Your Login.aspx should check for this cookie - and if set – display a link to Headers.aspx in addition to the prompts for the username and password. You should also populate the username field with the unencoded username.
- Add a logout button that "deletes" the cookie to your Headers form. If this button is clicked, you should not only expire the "login" cookie but also use a Response.Redirect() to reload your Login.aspx page. This logout button should only be visible if the user is currently logged in; i.e., the "login" cookie was found.

Demonstrate in Lab: Week of November 14th

Part 2: ADO.Net.

NB: there is a lot of material in this lab, so start early!

Extend your Login.aspx from Part 1 to authenticate against data stored in a relational database.

Read Chapter 14, pp. 425-432 and learn about the built-in version of SQL Server Express in Visual Studio.

- Using the sqlcmd command-line tool, install the downloaded code sample *pubs* database (p. 431).
- Manage the *pubs* database on (localhost)\v11.0 by using the built-in Server Explorer in Visual Studio (p. 428-430).
- Extend the database instance by adding a *customers* table that stores <int customerid, varchar(16) username, varchar(16) password, varchar(50) firstname, varchar(50) lastname>. How you do this isn't entirely intuitive! Ask the instructor if you are having trouble figuring it out, but:
    - Name the table in the T-SQL tab "[dbo].[customers]"

    - Rename the default “Id” field “customerid”
    - Allow NULL data only for the firstname and lastname fields
    - Click on the “Update” button (top left of design pane) to generate the SQL script to add the new table and select “Update database” from the preview window
- Refresh your database to see the changes. Add some rows of test data to your *customers* table.
- Create a *sessions* table to store <int customerid, varchar(50) cookie> relationships.

Read pp. 440-463 to learn how to connect to your database within C#, and submit queries.

- Store the connection string in the <connectionStrings> section of the Web.Debug.config file.
- Extend your Login.aspx page so that the username and password are authenticated against data stored in the *customers* table in your database. Display an appropriate error message and re-prompt if the entered values don’t correspond to a record in the table.
- Be sure to close your connection once done with the query! Following this pattern is a good way to ensure that your database connection is always closed:

```
SqlConnection con = new SqlConnection(myConnectionString);
try
{
    con.Open();
    … your code goes here…
}
catch (Exception e)
{
    Response.Write(e.Message);
}
finally
{
   con.Close();
}
```

- When a user logs in successfully, generate a random, spoof-proof cookie value (i.e., the encoding you used in Part 1 with some additional noise characters added at the beginning of the string is fine) and (a) set it on the user’s browser and (b) insert it into the *sessions* relation.
- Create a class called `Customer` with a constructor that takes a cookie as a single parameter. This constructor should query both the *sessions* table for a matching cookie value, and the *customers* table (with a join) to initialize the automatic properties `Firstname, Lastname,` and `Customerid.`). If for some reason the cookie does not exist in the database, set a boolean property `Exists` to false.
- Use your `Customer` class within your Login.aspx Page_Load event. Instantiate a Customer object using the cookie value retrieved from the browser (if present).
- If a user is logged in, display their first name, last name, customerid and cookie value on the login page. Do not prompt for login credentials; make those elements invisible by setting their Visible property to false.
- The “logout” button on your Headers page should appear if the user is logged in. Clicking it should not only expire the cookie, but also delete the record in the *sessions* table.

Demonstrate in Lab: Week of November 21st.

Part 3(a): Security.

Extend your Login.aspx to use better security: forms authentication.

- Read all of Chapter 19 on Security.
- Implement a webpage called SecuredPage.aspx that is similar to the one on page 631.
- Change the Web.config settings for your project in order to implement forms authentication like on page 629: only authenticated users should be able to view SecuredPage.aspx.

- Make SecuredPage.aspx the default (start) page of your web application. If the user is not logged in, SecuredPage.aspx will now automatically redirect the user to the Login.aspx page.
- Add a "remember me" checkbox to your Login.aspx page. If checked by the user, their login cookie should persist for one week. If not checked, the cookie should only last until the browser is restarted.
- Your Login page should set two cookies: the one you did in Part 1, plus another one using the technique described on page 633. (Instead of using the password "secret", you should continue to authenticate against the records in your database *sessions* table.) Set the username in the FormsAuthentication.RedirectFromLoginPage method to the firstname + lastname of the user from your *customers* table.
- Add a "Logout" button to SecuredPage.aspx. This button should remove both cookies. In addition to the ADO.Net code from Part 2 of this lab, use the Signing Out code on p. 632 to respond to the Logout button click to remove the new cookie.
- Add a Default.aspx page to your project. This page should simply include links to your SecuredPage.aspx and Headers.aspx pages. (More importantly, this gives a page for your Login page to land if it doesn't know where to redirect after successful login.)

Part 3(b): More Database Querying

Read the section on *Creating More Robust Commands* (pp. 458-61).

- Update your code from Part 2 to use *parameterized commands* (p. 459) instead of wrapping strings in cumbersome single quotes. (NB: You should always use parameterized commands in web applications that query SQL databases to avoid harmful SQL injection attacks.)

Let's now add some authorization to our application. Only logged in users should be able to query the *sales* database table.

- Update the SecuredPage.aspx such that it connects to the database, selects all records from the *sales* table, and displays the results in a nicely formatted HTML table.
- Add a "Refresh" button (resubmits the query of the *sales* table) and a link to the Default.aspx page to SecuredPage.aspx.

Demonstrate in Lab: Week of November 28th

Part 4: Caching

Read Chapter 23 on Caching (pages 729 to 740 only).

- Modify your Default.aspx page to use your custom cookie to create a `Customer` object (from Part 2) and display the `Firstname` and `Lastname` of the logged in user.
- Implement data caching (p. 737) to improve the performance of your web application. In particular, change Default.aspx so that it caches `Customer` objects to avoid creating a new object (and querying the *customers* and *sessions* tables) with each page load. Use the cookie value as the key. Use an absolute expiration of 30 seconds.
- Modify Default.aspx to indicate if there is a cache hit or miss.
- Make sure you clear the Cache of any related `Customer` object when you logout the user.

Demonstrate in Lab: Week of December 5th

Hand in your source code (via Dropbox) for the completed project the week of December 5th during your scheduled lab.