

## Tutorial for Class number 7

This exercise intends to develop a web application to test the use of web cookies and session variables and check their operation in HTTP messages.

In a third part of the exercise will be experienced the use of web cookies in an exercise that applies session cookies in the identity management of logged-in users.

### First step – testing web cookies

- Create a new “ASP.NET Core Web App (Model-View-Controller)” project.

Let's add an action to create web cookies. As there is no concrete objective for the project, the cookies created are merely examples of possible decontextualized values.

All values used in cookies have representation in text (string), regardless of the type of data they want to represent.

- Create the **AddCookies** action on the **Home** controller.

```
public IActionResult AddCookies()
{
    // this is just to test the use of web cookies
    // we should see them in the response messages and in the subsequent requests
    HttpContext.Response.Cookies.Append("Test1", "Value1"); // session cookie
    HttpContext.Response.Cookies.Append("Test2", "Value2",
        new CookieOptions() { Expires = DateTime.Now.AddSeconds(10) }); // persistent cookie (for 10 seconds)
    HttpContext.Response.Cookies.Append("Test3", "Value3",
        new CookieOptions() { Expires = DateTime.Now.AddDays(1) }); // persistent cookie (for 1 day)

    /// we could have some application logic code...
    ///
    return RedirectToAction("Index");
}
```

To verify the result of the creation of web cookies, we will implement in the Home/Index view the following code.

- Change the code of the existing **Index.cshtml** file in the **Views/Home** folder.

```
@{
    ViewData["Title"] = "Home Page";
}

<div class="text-center">
    <h1 class="display-4">Web Engineering</h1>
    <h2>This exercise is only to play a little with cookies and session variables</h2>
    <p>Learn about <a href="https://docs.microsoft.com/aspnet/core">building Web apps with ASP.NET Core</a>.</p>
</div>

<h1>Web cookies in request</h1>
@if (Context.Request.Cookies.Count == 0)
{
    <div>No cookies were sent</div>
}
@foreach (var item in Context.Request.Cookies.Keys)
{
    <div>@item - @Context.Request.Cookies[item]</div>
}
```

To better check the management of web cookies, implement a feature that allows you to delete them. To simplify the process, the displayed code always deletes all of them.

- Create the **DeleteCookies** action on the **Home** controller.

```
public IActionResult DeleteCookies()
{
    // delete all cookies from response (and client)
    foreach (var item in HttpContext.Request.Cookies.Keys)
    {
        HttpContext.Response.Cookies.Delete(item);
    }

    return RedirectToAction("Index");
}
```

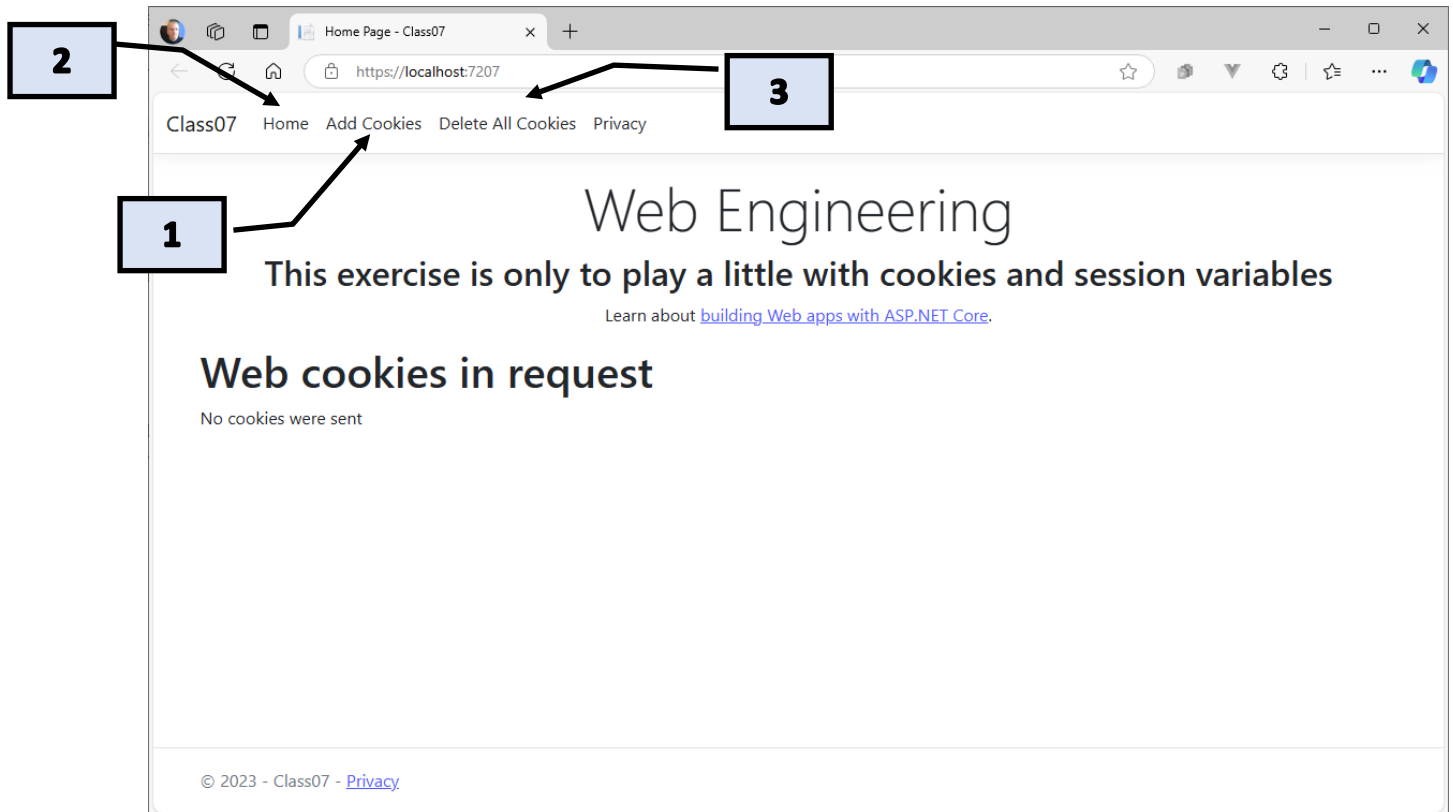
To better verify the process of creating and deleting web cookies, create menu options for direct access to the implemented features.

- Changes the existing **\_Layout.cshtml** file in the **Views/Shared** folder.

```
<div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
  <ul class="navbar-nav flex-grow-1">
    <li class="nav-item">
      <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
    </li>
    <li class="nav-item">
      <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="AddCookies">Add Cookies</a>
    </li>
    <li class="nav-item">
      <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="DeleteCookies">Delete All Cookies</a>
    </li>
    <li class="nav-item">
      <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
    </li>
  </ul>
</div>
```

Test the application using both features placed in the menu. Use the following sequence of actions to check the change in the status of web cookies:

- 1 - Add cookies to the web client. Immediately check the cookies that appear in the view;
- 2 – (several times) Renew the query request and confirm the evolution of cookies (2nd cookie disappears 10 seconds after it has been created);
- 3- Remove all cookies - the application gets his initial state;



After step 1, the application shows all cookies created.

## Web cookies in request

Test1 - Value1  
Test2 - Value2  
Test3 - Value3

The cookie information list (step 2) after 10 seconds of step 1 shows a missing (expired) cookie.

## Web cookies in request

Test1 - Value1  
Test3 - Value3

If we close the browser window, non-persistent (session) cookies are deleted and the result in the executed application is as follows (the Test2 cookie will also have expired after 10 seconds have occurred).

# Web cookies in request

Test3 - Value3

Check the influence of actions on headers of HTTP requests and responses. Using the browser developer mod (F12) we can consult the exchange of HTTP messages between client and server.

```
set-cookie: Test1=Value1; path=/  
set-cookie: Test2=Value2; expires=Sun, 22 Nov 2020 23:06:59 GMT; path=/  
set-cookie: Test3=Value3; expires=Mon, 23 Nov 2020 23:06:49 GMT; path=/
```

```
cookie: Test1=Value1; Test3=Value3; Test2=Value2
```

The removal of web cookies is done by sending the same cookie (same name) with an already expired date.

```
set-cookie: Test1=; expires=Thu, 01 Jan 1970 00:00:00 GMT; path=/  
set-cookie: Test2=; expires=Thu, 01 Jan 1970 00:00:00 GMT; path=/  
set-cookie: Test3=; expires=Thu, 01 Jan 1970 00:00:00 GMT; path=/  
HTTP/1.1 200 OK
```

## Second step – testing session variables

Now let's test the use of session scanners. We can configure the application so that we can manage in the server "memory" a set of data referring to any request from a client and that can be reused between requests. To be able to do this, a specific cookie is used to identify the session.

Configure the application to use session variables.

- Change the file **Program.cs** to contain the following code snippets.

```
// Add services to the container.
builder.Services.AddControllersWithViews();

builder.Services.AddSession(option => {
    option.IdleTimeout = TimeSpan.FromMinutes(10);
    option.Cookie.Name = ".AspNetCore.Session";
});

var app = builder.Build();
```

```
app.UseAuthorization();

app.UseSession();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
```

With this configuration, the server creates a session with a unique ID to allow the storage of information required by the application in the context of the particular client.

- Create the **AddSessionVariables** action in the **Home** controller.

```
public IActionResult AddSessionVariables()
{
    // we can create variables of type string, int ou byte array.
    HttpContext.Session.SetString("StringValue", "Text variable value");
    HttpContext.Session.SetInt32("IntegerValue", 100);
    // a session cookie '.AspNetCore.Session' will be automatically created and send to the client

    return RedirectToAction("Index");
}
```

- Create the **DeleteSessionVariables** action on the **Home** controller.

```
public IActionResult DeleteSessionVariables()
{
    // delete all variables stored in session
    // this does not ends the session. For that it is necessary to delete the cookie
    foreach (var item in HttpContext.Session.Keys)
    {
        HttpContext.Session.Remove(item);
    }

    return RedirectToAction("Index");
}
```

- Create the **DeleteSession** action on the **Home** controller.

```
public IActionResult DeleteSession()
{
    // this really delete all session variables because it ends de session itself
    HttpContext.Response.Cookies.Delete(".AspNetCore.Session");
    // this is the default name (see service configuration)

    return RedirectToAction("Index");
}
```

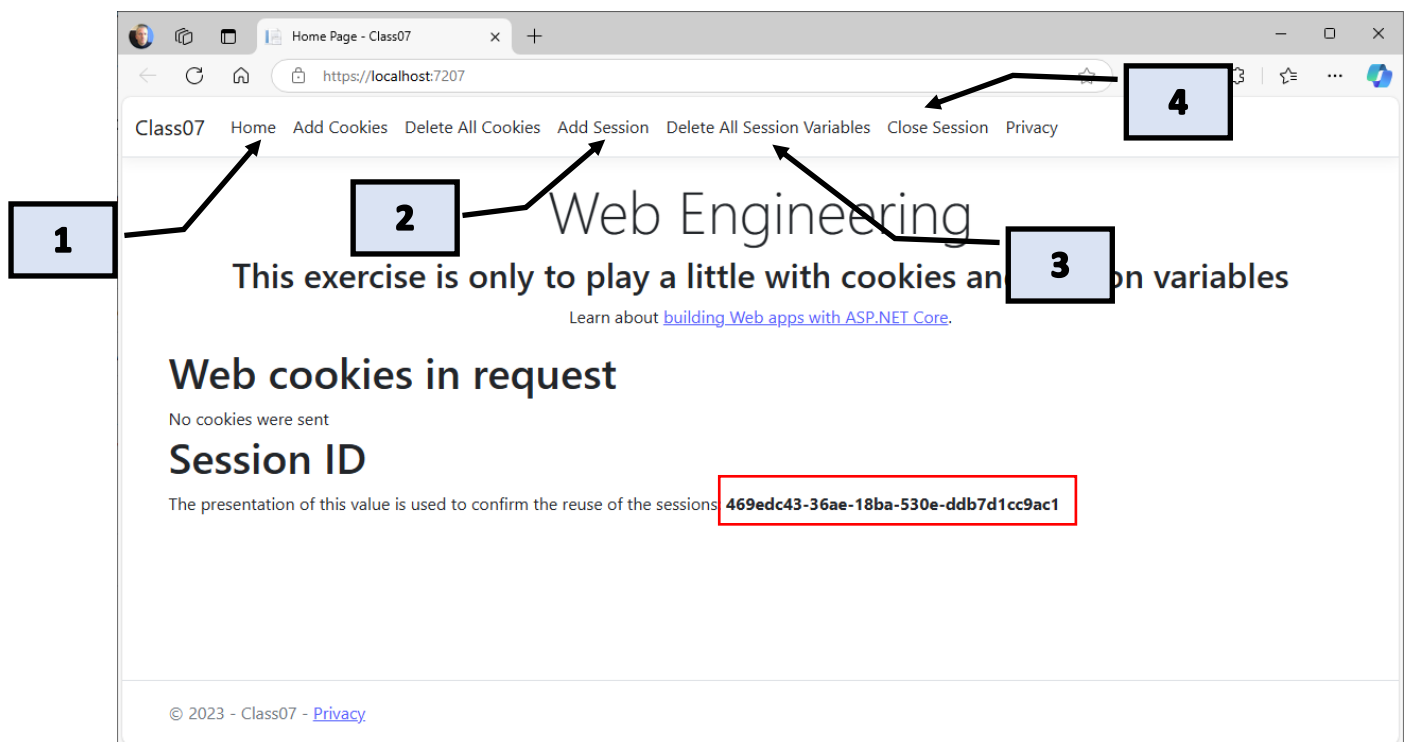
Change the code in the **Index.cshtml** file in the **Views/Home** folder by adding the following code to the end of the file.

```
<h1>Session ID</h1>
<div>The presentation of this value is used to confirm the reuse of the sessions: <strong>@Context.Session.Id</strong></div>
@foreach (var item in Context.Session.Keys)
{
    if (item == "StringValue")
    {
        <div>@item - @Context.Session.GetString(item)</div>
    }
    else
    {
        <div>@item - @Context.Session.GetInt32(item)</div>
    }
}
```

Again, to better verify the process of creating and deleting session variables, create menu options for direct access to deployed features.

- Changes the existing **\_Layout.cshtml** file in the **Views/Shared** folder.

```
<li class="nav-item">
  <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="DeleteCookies">Delete All Cookies</a>
</li>
<li class="nav-item">
  <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="AddSessionVariables">Add Session</a>
</li>
<li class="nav-item">
  <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="DeleteSessionVariables">Delete All Session Variables</a>
</li>
<li class="nav-item">
  <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="DeleteSession">Close Session</a>
</li>
<li class="nav-item">
  <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
</li>
```



Through the constant clicking of option **1 - Home**, it turns out that the Session ID is constantly changing. It means that the server always uses a different session with each request.

Option **2 - Add Session** creates the **StringValue** and **IntegerValue** session variables and simultaneously the “.AspNetCore.Session” web cookie. Here the server allows you to recover the session for reuse of the data.

## Web cookies in request

.AspNetCore.Session -

CfDJ8NUSS5mDjJROkP2dvGWITqN5IDPW4TLdOkOQRGe2RZgLMZRMvPnRhKwTf89WsN0Dd3XdVYrGsr8Rr9wAReAK/UVwzyCqBcY

## Session ID

The presentation of this value is used to confirm the reuse of the sessions: **51e2073b-fe79-2a18-578a-191867d57e24**

StringValue - Text variable value

IntegerValue - 100

At this stage, the constant click of option **1 - Home** no longer changes the value of the Session ID because the server reuses the session. Hence you can display the values of the session variables.

Option **3 - Delete** deletes all session variables that had been created but does not delete the session. This is confirmed because, although the variables are no longer displayed in the view (have been deleted) the value of the Session ID remains the same.

## Web cookies in request

.AspNetCore.Session -

CfDJ8NUSS5mDjJROkP2dvGWITqN5IDPW4TLdOkOQRGe2RZgLMZRMvPnRhKwTf89WsN0Dd3XdVYrGsr8Rr9wAReAK/UVwzyCqB

## Session ID

The presentation of this value is used to confirm the reuse of the sessions: **51e2073b-fe79-2a18-578a-191867d57e24**

Option **4 - Close Session** allows the cookie that identifies the session to be deleted from the client. In this way, the reference to the session used is lost and is equivalent to deleting all the information from the session (as result, the presented Session ID is changed again).

These changes can also be confirmed in the analysis of HTTP messages.

At option 2 (create session):

```
set-cookie: .AspNetCore.Session=CfDJ8KbztDno6xBNkPDmwlCpMKI8esOfN  
AEDN8fFWRj%2B9pWKLd4wMRRwL2iQZzeyb2oWZAwGzI5R09uPT7uNu35vea44aW  
5hKb274pbDHw2uHL67bEjoFYAuWVNpnsdUC0b5g1YTS3fULFDbMd7UNMwt5DNAs  
07b0PCnP4QK07Izifck; path=/; samesite=lax; httponly
```



Following option 2 (all requests to the server):

```
cookie: .AspNetCore.Session=CfDJ8KbztDno6xBNkPDmWwCpMKI8esOfNAED
NBffWRj%2B9pWKLd4WMRRwL2iQZzeyb2oWZAwGzI5R09uPT7uNu35vea44aW5hK
b274pbDHw2uHL67bEjofYAuwVNpnsdUC0b5g1YTS3FULFDbMd7UNMwt5DNAs07b
OPCnP4QK07Izifck
```

When option 4 (delete session):

```
set-cookie: .AspNetCore.Session=; expires=Thu, 01 Jan 1970 00:00:
00 GMT; path=/
```

For more information see the URL

<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/app-state?view=aspnetcore-8.0>

### Third step – Applying web cookies usage

Now it's time to apply this knowledge to a more realistic application. The application will be based on the authentication of a user and consequently on features that are only available when the user is well identified (with an active session).

- Create a new “ASP.NET Core Web App (Model-View-Controller)” (could be in the same solution)
- Add the **User** class to manage the identity of users:

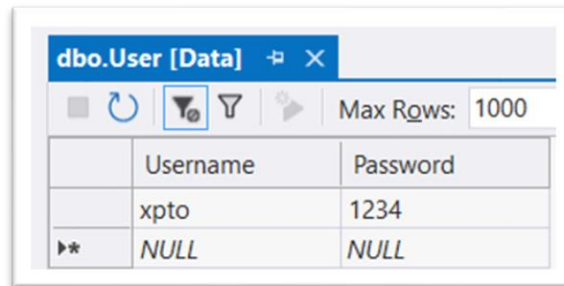
```
public class User
{
    [Key]
    [Required]
    public string Username { get; set; }

    [Required]
    [DataType(DataType.Password)]
    public string Password { get; set; }
}
```

- Create the database for the project, providing the project with a Database Context, a connection string with the name **Class07bContext\_Database** and the corresponding registration as a service in the **CreateBuilder** process (in **Program.cs**).

**TIP:** add an “MVC Controller with views, using Entity Framework” to generate these elements.

- Add a data model Migration and update the database.
- In the database (SQL Server Explorer), add a row in the **User** table with the data “xpto” and “1234” as *username* and *password*, respectively.



The screenshot shows the 'dbo.User [Data]' window in SQL Server Enterprise Manager. The window has a toolbar with icons for refreshing, filtering, and sorting. The 'Max Rows' is set to 1000. The table has two columns: 'Username' and 'Password'. The data is as follows:

	Username	Password
	xpto	1234
▶*	NULL	NULL

- In the **Users** controller, eliminate all the generated methods (actions) leaving only the class constructor. Also delete from project all views corresponding to the deleted actions.

```
public class UsersController : Controller
{
    private readonly Class07bContext _context;

    public UsersController(Class07bContext context)
    {
        _context = context;
    }
}
```

- Now dealing with authentication functionality, create a **Login** action in **Users** controller to authenticate a user:

For the GET action (retrieve the login view)...

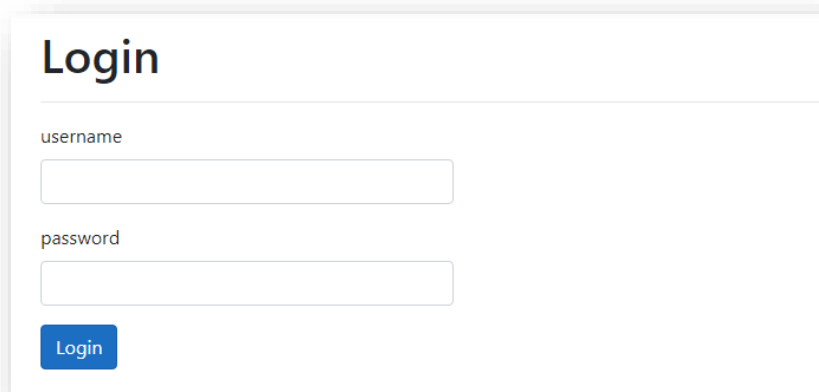
```
public IActionResult Login()
{
    return View();
}
```

...and for the POST action (process the form submission)...

```
[HttpPost]
public IActionResult Login(string username, string password)
{
    if (ModelState.IsValid)
    {
        User u = _context.User.SingleOrDefault(u => u.Username == username && u.Password == password);
        if (u == null)
            ModelState.AddModelError("username", "username or password are wrong");
        else
        {
            // the user is authenticated
            // the session variable "user" is created to recover the user identify at each request
            HttpContext.Session.SetString("user", username);

            return RedirectToAction("Index", "Home");
        }
    }
    return View();
}
```

- Add a view with a form, generated from the *Create* template and the **User** model, which looks like the one shown in the following figure.



The image shows a web form titled "Login". It has two input fields: one for "username" and one for "password". Below the "password" field is a blue button labeled "Login". The form is styled with a light gray border and a white background.

It is necessary to configure the application to use sessions.

- Add the following code to the **Program.cs** file.

```
builder.Services.AddDbContext<DbContext>(options =>
    options.UseSqlServer(builder.Configuration.GetC

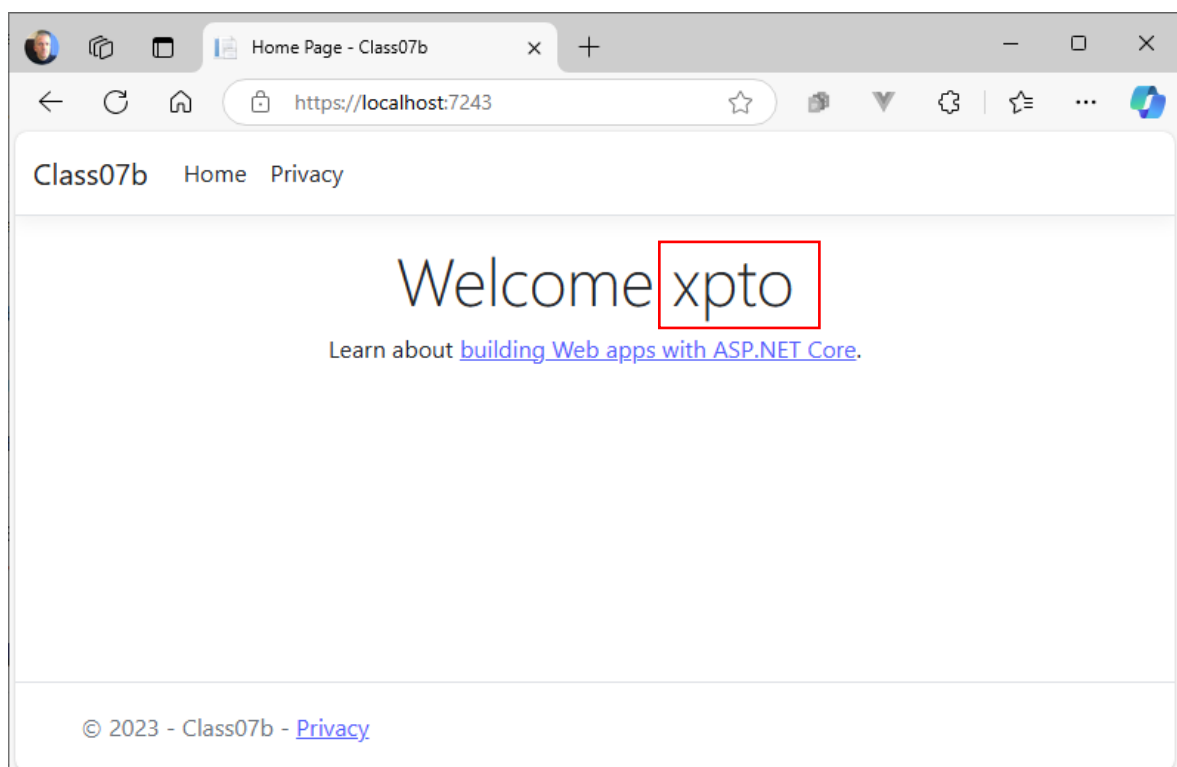
builder.Services.AddSession(option => {
    option.IdleTimeout = TimeSpan.FromMinutes(10);
    option.Cookie.Name = ".Class07b.Session";
});
```

```
app.UseAuthorization();  
  
app.UseSession();  
  
app.MapControllerRoute(  
    name: "default",  
    pattern: "{controller=Home}/{action=Index}/{id?}");
```

- Change the view of the **Home/Index** action to present the welcome to the user who has authenticated:

```
@{  
    ViewData["Title"] = "Home Page";  
}  
  
<div class="text-center">  
    <h1 class="display-4">Welcome @Context.Session.GetString("user")</h1>  
    <p>Learn about <a href="https://docs.microsoft.com/aspnet/core">building Web apps with ASP.NET Core</a>.</p>  
</div>
```

When testing this **Login** feature, we should get the following screen after a successful authentication.



We will add the **Logout** functionality and add the respective options to the application menu.

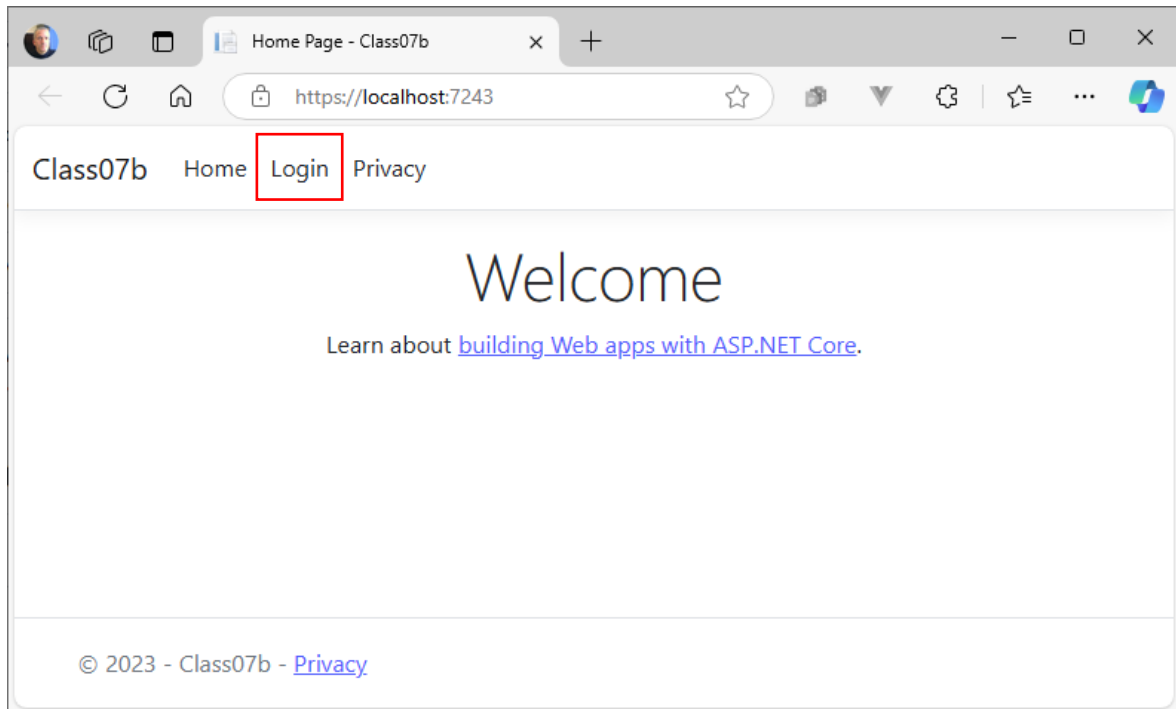
```
public IActionResult Logout()
{
    HttpContext.Response.Cookies.Delete(".Class07b.Session");

    return RedirectToAction("Index", "Home");
}
```

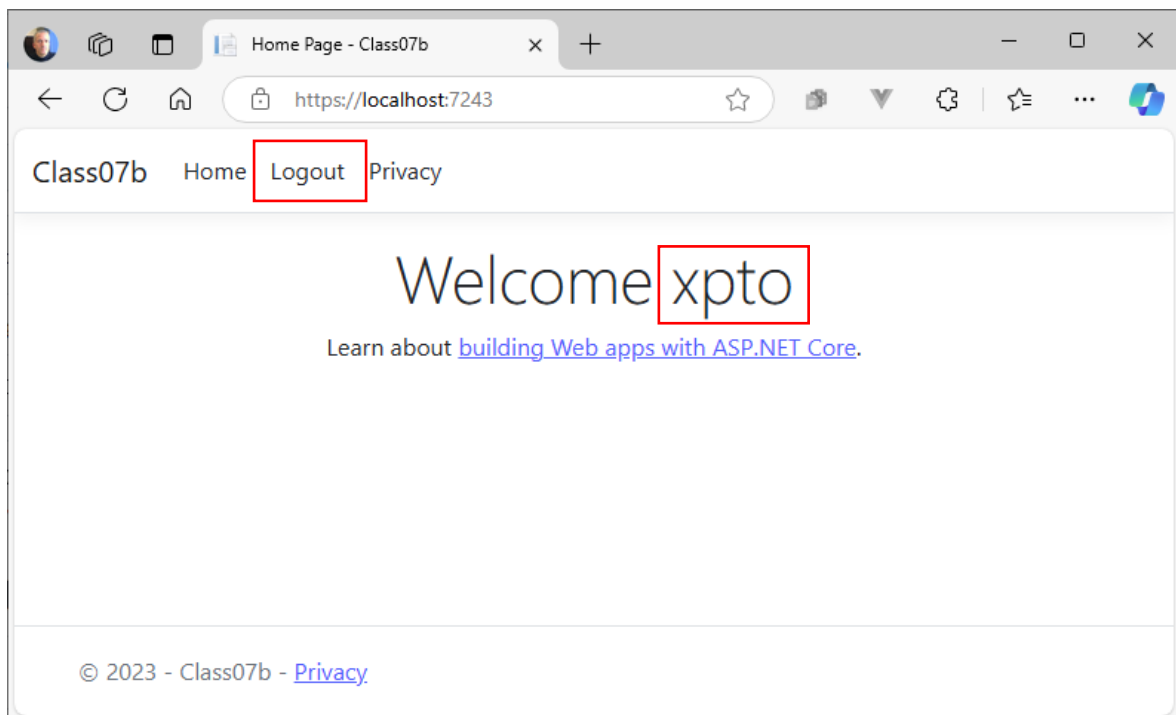
- Change the **\_Layout.cshtml** file in the **Views/Shared** folder to add the **Login** and **Logout** options to the menu.

```
<ul class="navbar-nav flex-grow-1">
  <li class="nav-item">
    <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
  </li>
  @if (Context.Session.GetString("user") == null)
  {
    <li class="nav-item">
      <a class="nav-link text-dark" asp-area="" asp-controller="Users" asp-action="Login">Login</a>
    </li>
  }
  @if (Context.Session.GetString("user") != null)
  {
    <li class="nav-item">
      <a class="nav-link text-dark" asp-area="" asp-controller="Users" asp-action="Logout">Logout</a>
    </li>
  }
  <li class="nav-item">
    <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
  </li>
</ul>
```

These menu options are configured to appear to the user only when necessary (**Login** only appears when the user is not *logged in* and **Logout** appears only when the user is already *logged in*).



After logging in, the following information should appear.



We will now add a new feature that allows us to use cookies to store user preferences locally in the browser.

This feature intends to allow the user to change the application presentation between light mode and dark mode.

- Add the **Preferences** action to the **Users** controller using the following code:

```
public IActionResult Preferences()  
{  
    ViewBag.mode = HttpContext.Request.Cookies["viewMode"] ?? "light";  
  
    return View();  
}  
[HttpPost]  
public IActionResult Preferences(string mode)  
{  
  
    HttpContext.Response.Cookies.Append("viewMode", mode, new CookieOptions { Expires = DateTime.Now.AddYears(1) });  
    return RedirectToAction("Index", "Home");  
}
```

- For the actions implemented, create the view with a form using the following code:

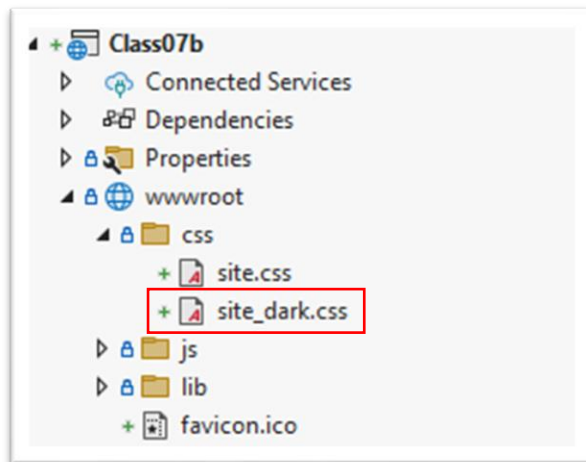
```
<h1>Preferences</h1>  
<h2>View Mode</h2>  
<div class="row">  
    <div class="col-md-4">  
        <form asp-action="Preferences">  
            <div class="form-group">  
                <input type="radio" name="mode" value="light" checked=@(ViewBag.mode=="light" )/>Light  
                <br />  
                <input type="radio" name="mode" value="dark" checked=@(ViewBag.mode=="dark" ) />Dark  
            </div>  
            <div class="form-group">  
                <input type="submit" value="Go" class="btn btn-primary" />  
            </div>  
        </form>  
    </div>  
</div>
```

- Add an option to navigate to this new feature to the application menu. The option also should be available only to *logged in* users.

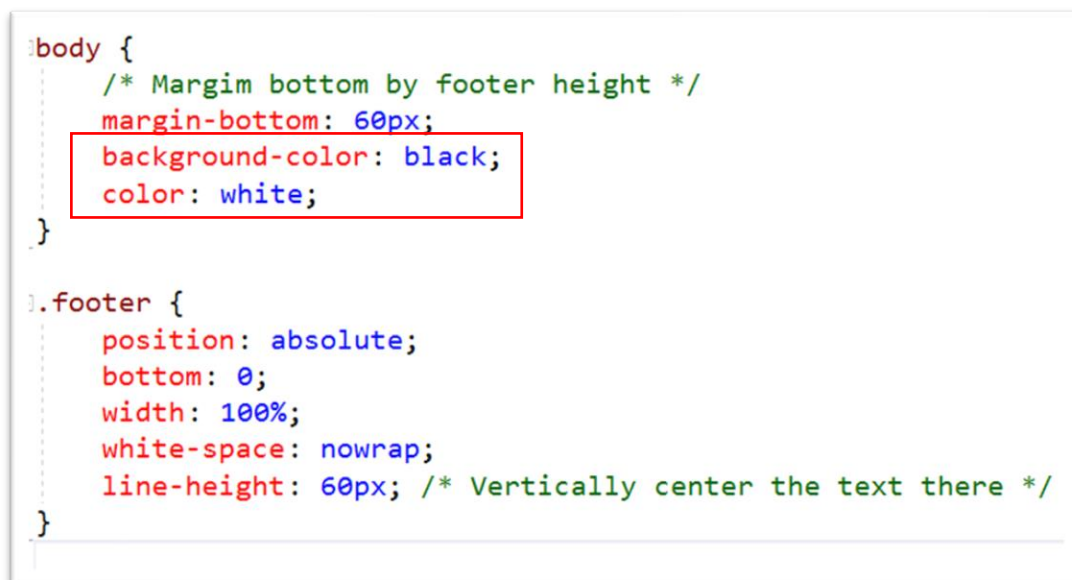
```
@if (Context.Session.GetString("user") != null)  
{  
    <li class="nav-item">  
        <a class="nav-link text-dark" asp-area="" asp-controller="Users" asp-action="Preferences">Preferences</a>  
    </li>  
    <li class="nav-item">  
        <a class="nav-link text-dark" asp-area="" asp-controller="Users" asp-action="Logout">Logout</a>  
    </li>  
}  
<li class="nav-item">  
    <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>  
</li>
```

To implement the change between light and dark mode, we will use an exchange of the CSS file applied to the layout of the application.

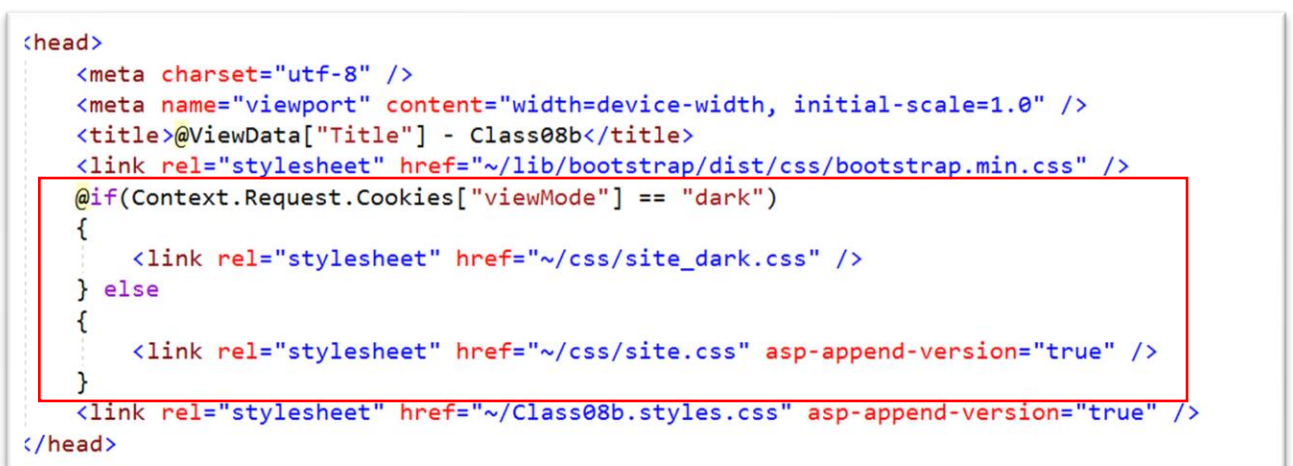
- In the **wwwroot\css** folder, create a file named **site\_dark.css** as a clone of the **site.css** file in that same folder.



- Change the created file by adding the style formatting lines according to the following figure.



- Change the **\_Layout.cshtml** file in the **Views/Shared** folder as follows.





- Test the application ...

**Homework:**

- Add the **Register** action to the register a new **User**. Guarantee that does not exist other user with same Username.
- After successful registration, automatic login is made.
- Add an option to navigate to this new feature to the application menu. The option also should be available only to **not logged in** users