**P4 – Session Management & vulnerabilities**

# Overview

Task :

1. Managing ASP.Net Core Session
2. Learn how to implement Session Token (Anti-forgery)
3. Learn how to avoid the Session fixation vulnerability in ASP.NET.

**Part 1 ASP.NET Core Session Management**

## Session State

### Session state is a method in ASP.NET Core to store user data while the user browses a web app.

### Session state uses a store maintained by the app to persist data across requests from a client.

### The session data is backed by a cache and considered temporary.

ASP.NET Core maintains session state by providing a cookie to the client that contains a session ID. The cookie session ID:

* Is sent to the app with each request.
* Is used by the app to fetch the session data.

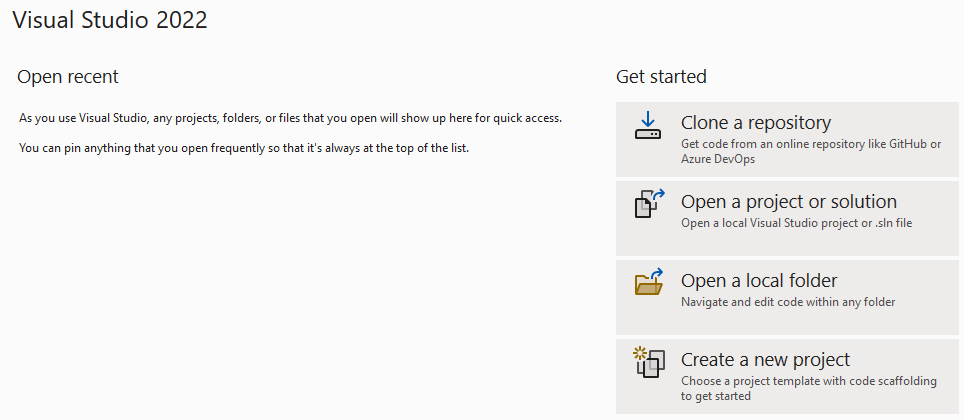
Session state exhibits the following behaviors:

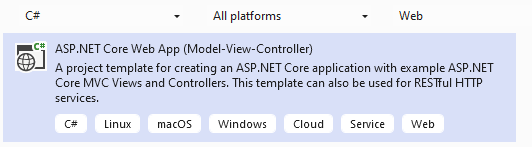
* The session cookie is specific to the browser. Sessions aren't shared across browsers.
* Session cookies are deleted when the browser session ends.
* If a cookie is received for an expired session, a new session is created that uses the same session cookie.
* Empty sessions aren't retained. The session must have at least one value set to persist the session across requests. When a session isn't retained, a new session ID is generated for each new request.
* The app retains a session for a limited time after the last request. The app either sets the session timeout or uses the default value of 20 minutes. Session state is ideal for storing user data:
  + That's specific to a particular session.
  + Where the data doesn't require permanent storage across sessions.
* Session data is deleted either when the **ISession.Clear** implementation is called or when the session expires. (This will be covered in next week practical)
* There's no default mechanism to inform app code that a client browser has been closed or when the session cookie is deleted or expired on the client.
* Session state cookies aren't marked essential by default. Session state isn't functional unless tracking is permitted by the site visitor. For more information, see General Data Protection Regulation (GDPR) support in ASP.NET Core.

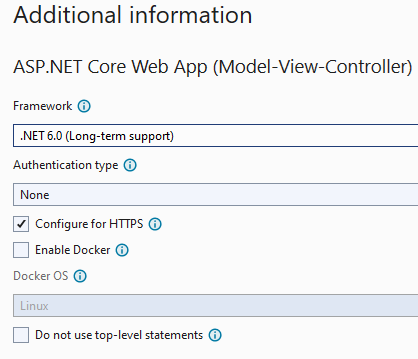
**Task 1 : How to create session and set timeout in ASP.NET Core**

Create a ASP.NET Core MVC  
  
The following steps show how to create a web project:

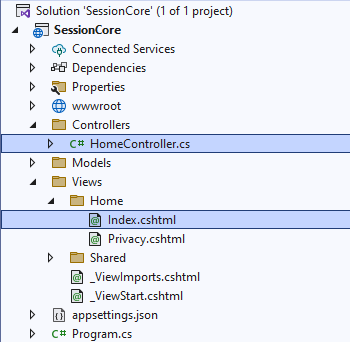
1. Create a new project 🡪 “ASP.NET Core Web App (Model-View-Controller)





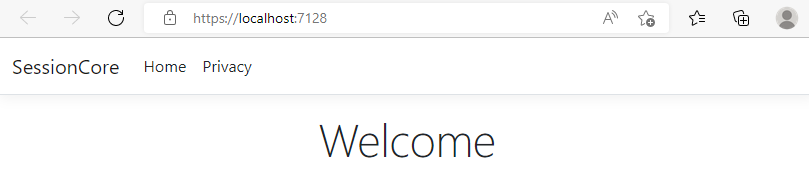


1. Check the files in the solution folder.



For this lab, we will be mostly using HomeController.cs and Index.cshtml.

Run the app to check if the default page is working fine. The content below was displayed from Index.cshtml.



1. Make changes to Program.cs file by adding 3 important services.

* AddSingleton 🡪 IHttpContextAccessor
* AddDistributedMemoryCache
* AddSession 🡪 Set timeout value

**HTTPContext** helps to extract session info within the app. HttpContext encapsulates all information about an individual HTTP request and response. An HttpContext instance is initialized when an HTTP request is received. The HttpContext instance is accessible by middleware and app frameworks such as Web API controllers, Razor Pages, etc.

**AddDistributedMemoryCache** (Distributed Memory Cache () ) is a framework that provides an implementation of IDistributedCache that stores items in memory. The Distributed Memory Cache isn't an actual distributed cache. Cached items are stored by the app instance on the server where the app is running.

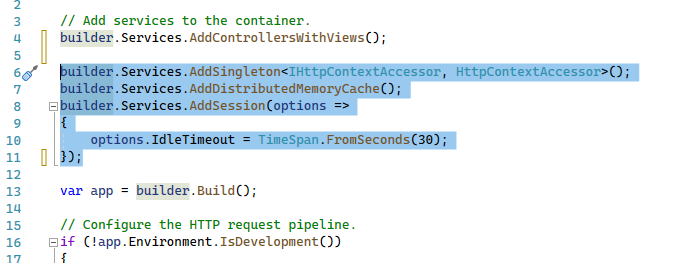
**AddSingleton()** creates a single instance of the service when it is first requested and reuses that same instance in all the places where that service is needed. The **IHttpContextAccessor** is an interface for . Net Core for accessing HttpContext property. This interface needs to be injected as dependency in the Controller and then later used throughout the Controller.

To enable the session middleware, Program.cs must contain:

Any of the IDistributedCache memory caches. The IDistributedCache implementation is used as a backing store for session. For more information, see Distributed caching in ASP.NET Core.

* A call to AddSession
* A call to UseSession

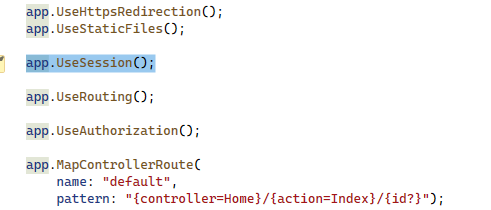
Add the codes into Program.cs



builder.Services.AddSingleton<IHttpContextAccessor, HttpContextAccessor>();  
builder.Services.AddDistributedMemoryCache(); //save session in memory

builder.Services.AddSession(options=>

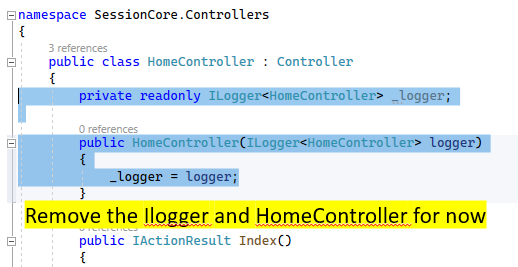
{  
 options.IdleTimeout = TimeSpan.FromSeconds(30);   
} );



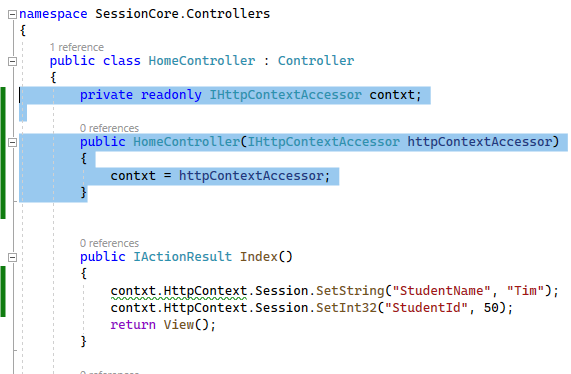
app.UseSession();

1. In HomeController.cs

Remove the following 2 lines of codes. We will add our own constructor later on.



Add the following codes into HomeController class :

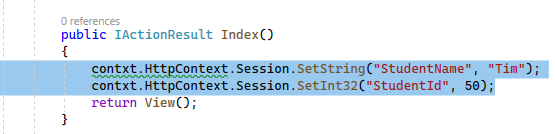


private readonly IHttpContextAccessor contxt;  
public HomeController(IHttpContextAccessor httpContextAccessor){

contxt = httpContextAccessor;

}

**Create a session variable inside IActionResult Index()**

****

public IActionResult Index …

{

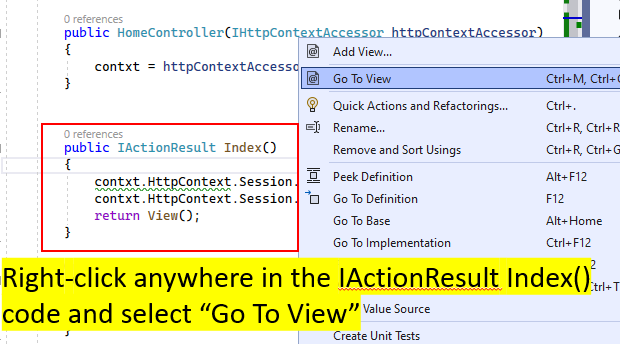
contxt.HttpContext.Session.SetString("StudentName", "Tim");  
 contxt.HttpContext.Session.SetInt32("StudentId", 50);

return View();

}

1. Make change to the HomeController View (index.cshtml)

Right click anywhere in the IActionResult Index() of HomeController.cs 🡪 select “Go To View”



1. In Index.cshtml add and remove the recommended codes.

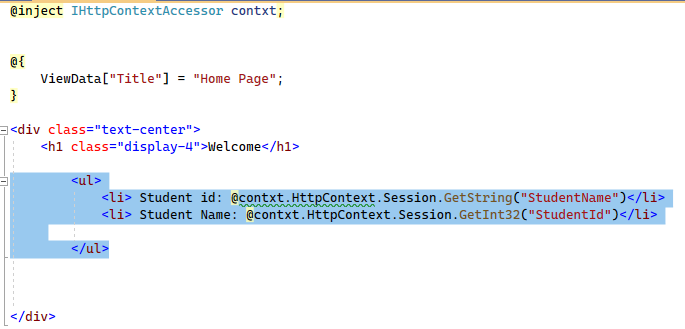
Add in the IHttpContextAccessor objects



Add :

@inject IHttpContextAccessor contxt;

Display the content of the Session.



Add :

<ul>  
 <li> Student id: @contxt.HttpContext.Session.GetString("StudentName")</li>  
 <li> Student Name: @contxt.HttpContext.Session.GetInt32("StudentId")</li>  
 </ul>

Run the app and see the Session data appear in the home page.

1. Try passing data to the privacy page

**In Homecontroller.cs** add in the following codes and report the above instruction.

public IactionResult Privacy(){

string StudentName = contxt.HttpContext.Session.GetString("StudentName");  
 return view();

}

1. Test again.

**Task 2 : Session Token (Anti-forgery) in .Net Core**

Cross-site request forgery (also known as XSRF or CSRF) is an attack against web-hosted apps whereby a malicious web app can influence the interaction between a client browser and a web app that trusts that browser. These attacks are possible because web browsers send some types of authentication tokens automatically with every request to a website. This form of exploit is also known as a one-click attack or session riding because the attack takes advantage of the user's previously authenticated session.

Using HTTPS doesn't prevent a CSRF attack. The malicious site can send an website request just as easily as it can send an insecure request.

CSRF attacks are possible against web apps that use cookies for authentication because:

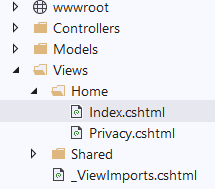
* Browsers store cookies issued by a web app.
* Stored cookies include session cookies for authenticated users.
* Browsers send all of the cookies associated with a domain to the web app every request regardless of how the request to app was generated within the browser.

The Method

When a user is authenticated, they're issued a token (not an antiforgery token). The token contains user information in the form of claims or a reference token that points the app to user state maintained in the app. When a user attempts to access a resource requiring authentication, the token is sent to the app with an additional authorization header in form of Bearer token. This makes the app stateless. In each subsequent request, the token is passed in the request for server-side validation. This token isn't encrypted; it's encoded. On the server, the token is decoded to access its information. To send the token on subsequent requests, store the token in the browser's local storage. Don't be concerned about CSRF vulnerability if the token is stored in the browser's local storage. CSRF is a concern when the token is stored in a cookie.

Lets begin with the short demo !

* Edit Index.cshtml to create a TextBox query and Submit button



<div class="text-center">

<h1 class="display-4">Welcome</h1>

<form **asp-controller**="Home" **asp-action**="Index" method="post" **asp-antiforgery**="true">

<input name="txtname" type="Text" />

<input type="Submit" value="Submit" />

</form>

</div>

* Edit HomeController to process the Index page with a textbox entry. Note that there should be 2 different sets of IActionResult Index(..)

[HttpPost]

[ValidateAntiForgeryToken]

public IActionResult Index(IFormCollection fc)

{

string res = fc["txtname"];

return View();

}

public IActionResult Index()

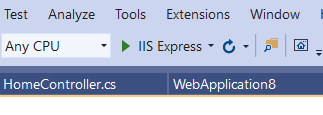
{

return View();

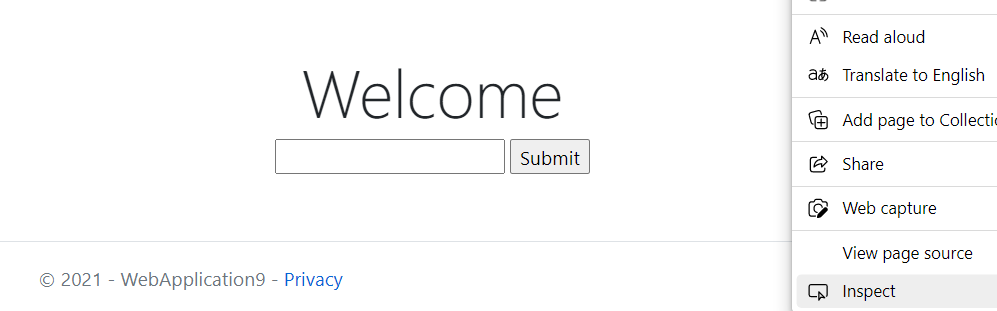
}

* Add a reference : using Microsoft.AspNetCore.Http;

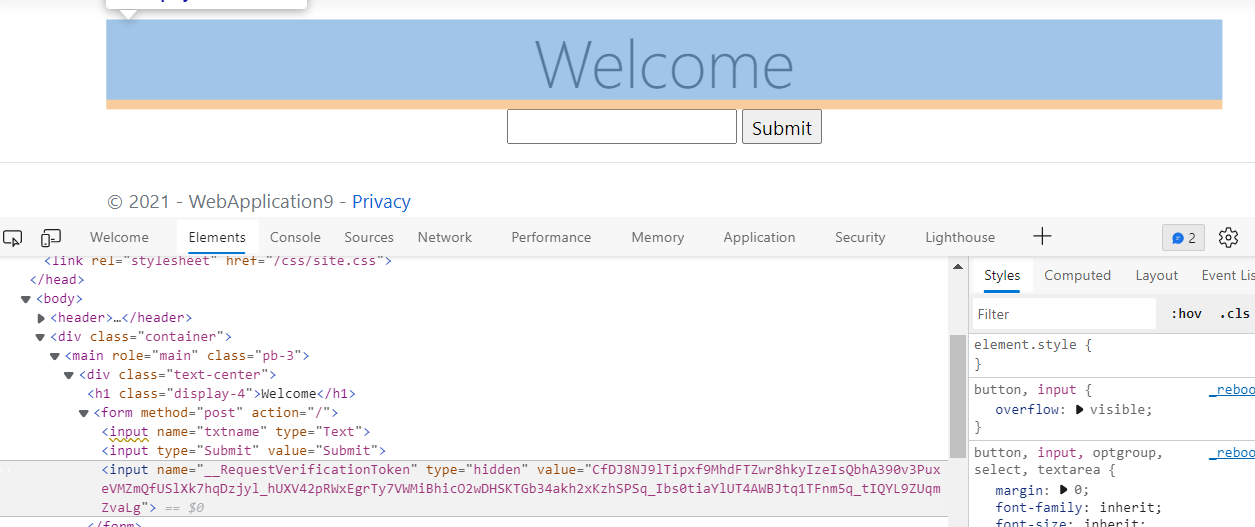
Run the app now :



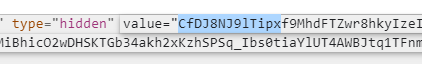
* Right click on anywhere on the webpage and choose “Inspect”



* To verify if the token and the Anti-Forgery process works, we need to delete some chars from the Anti-forgery token from the “Inspect” window.
* Search for the hidden Anti-Forgery Token and make some modification to it by deleting some chars from it.



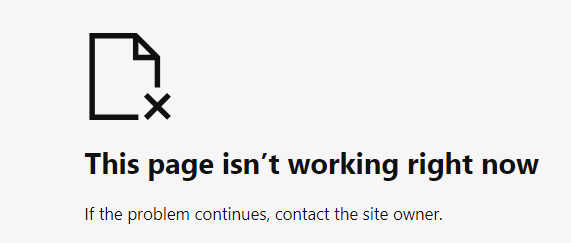
* Example just delete some chars from this string.



* After deleting some chars from the hidden token, enter a search string into the textbox.
* This will verify if your token is working or not.



* If your Anti-forgery is working properly, it should return an error since the token did not match the one from the server!



**Part 3 Session Fixation (ASP.Net Framework)**

## What is Session Fixation?

A session fixation attack lets an attacker set her session identifier for another user. The result would be a session shared between an attacker and the victim. When the user interacts with the website sensitive information is added to the session. Since the user and the attacker shares the session ID, the changes will also be reflected on the attacker's end. Information stored in session might be disclosed in a web page, or it could be used to control access to protected resources. Either way, the attacker will gain access to information that should only be accessible to the user.

### The Basics : Session

ASP.NET Session keeps track of the user by creating a cookie called **ASP.NET\_SessionId** in the user browser. This cookie value is checked for every request to ensure that the data being served is specific to that user. In many applications, a Session variable is used to track the logged in user, i.e., if a session variable exists for that user, then the user is logged in, otherwise not.

Whenever any data is saved into the Session, the ASP.NET\_SessionId cookie is created in the user’s browser. Most developers sanitize the Session object by deploying the following methods when user logout from the system - **Session.Abandon() or Session.RemoveAll() or Session.Clear()**

However, the ASP.NET\_SessionId cookie and its value is not deleted from the user browser even though the above methods were executed. This legitimate cookie value can be used by the hijacker to hijack the user session by giving a link that exploits cross site scripting vulnerability to set this pre-defined cookie. When the user clicks this link and logs in, the user will have the same ASP.NET\_SessionId cookie value that hijackers knows and he will also be able to browse the user account and will be able to access all the information pertaining to that user. This attack is called Session fixation vulnerability.

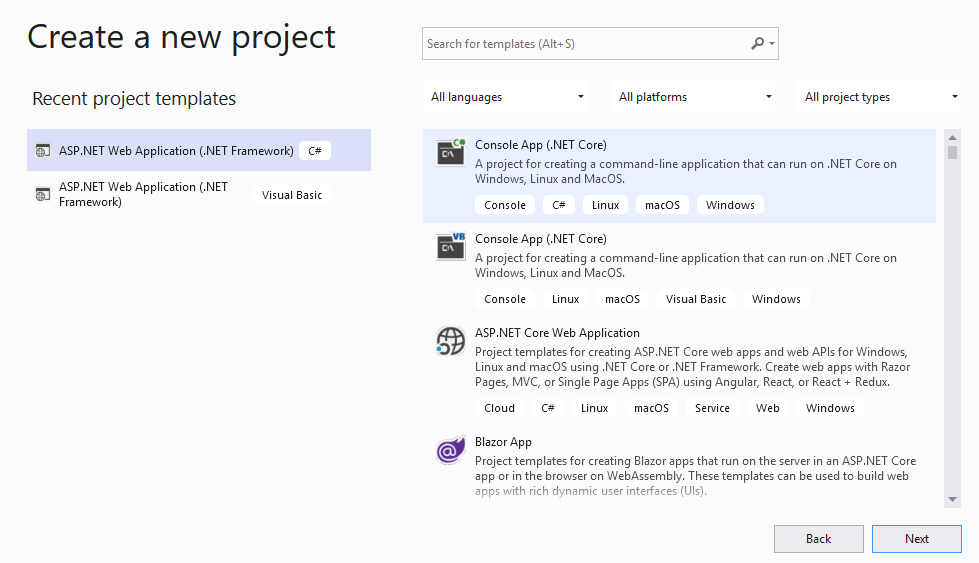
Let us create a simple web site to illustrate the above issue. We will add codes along the way to fix the fix the fixation issue.

**Task 3.1 : Session Fixation vulnerability**

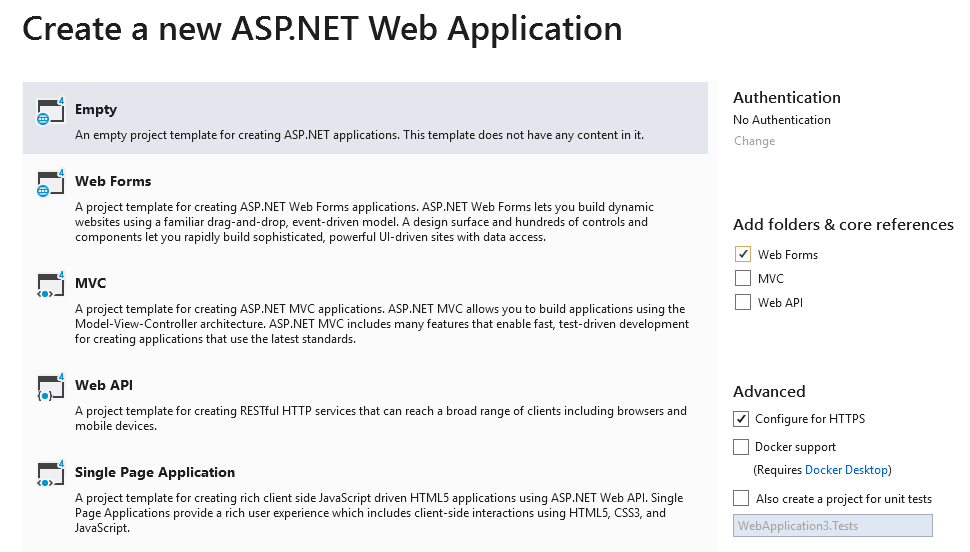
Create a New website  
  
The following steps show how to create a web project:

1. Click **New Project** in the **Start** page or in the **File** menu.
2. In the **New Project** dialog, select **ASP.NET Web Application** (c#)
3. Specify project **Name**, **Location**, and other options, and then click **OK**.

The **New ASP.NET Project** dialog appears.



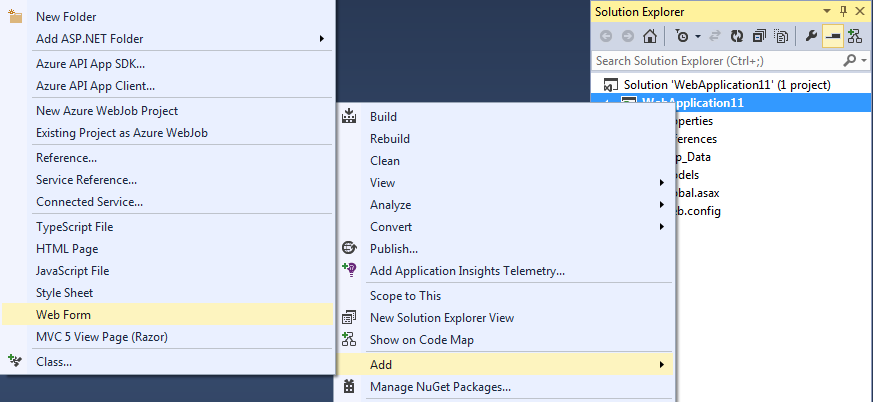
1. Click a template : Empty

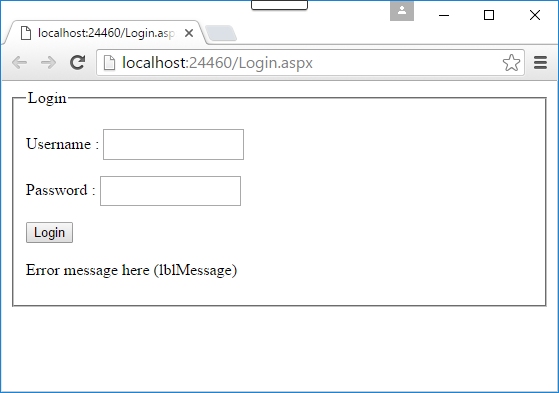


Create a Login.aspx form

Add a new Web Fom (C#) – Login.aspx

Right-click application name 🡪 choose Add 🡪 web form





Textboxes/Button/Label

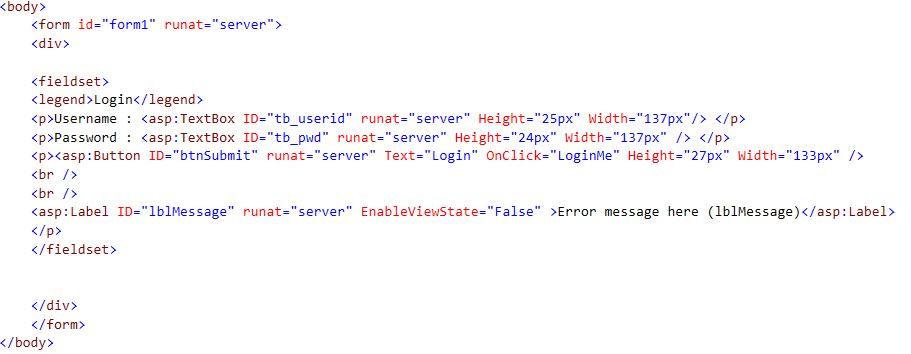
UserName – tb\_userid

Password – tb\_pwd

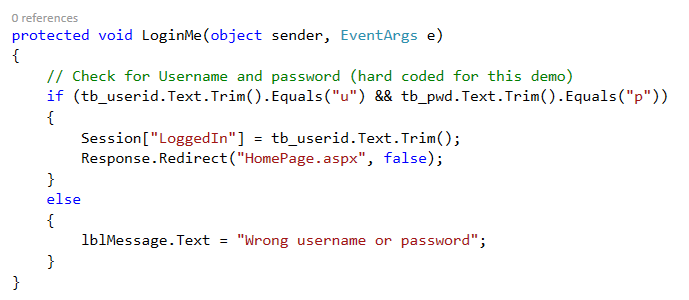
Login Button – btn\_login

Error Message label – lblMessage

Your final Login.aspx codes should look something like the one shown below.

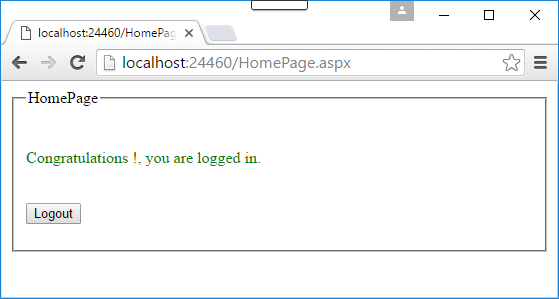


Create the Login method in Login.aspx.cs



For the purpose of this example, we will simply just hardcode the userid and password as ‘**u**’ and ‘**p**’ respectively. If the userid and password matches, LoginMe will forward the request to HomePage.aspx.

Create the HomePage.aspx – for successful login



Button/Label

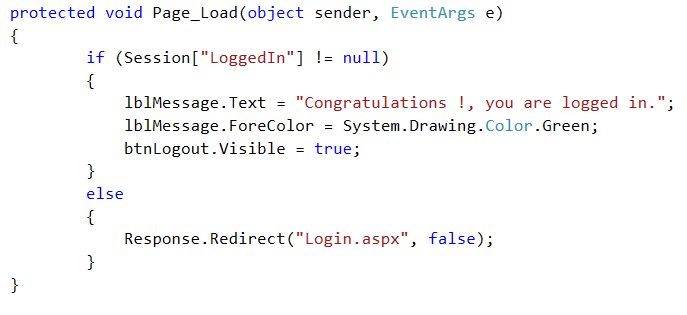
Logout Button – btn\_login

Error Message label – lblMessage

HomePage.aspx codes:

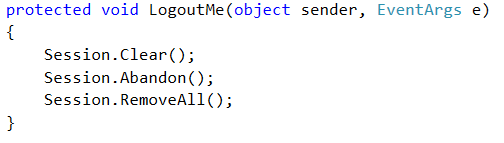


Create the Page\_Load method in HomePage.aspx.cs



Page\_Load method checks for valid session and display a successful message. If Session is invalid, the method will redirect browser back to Login.aspx. This also helps to prevent “**Insecure Direct Object Reference**” – example will be able to access HomePage.aspx without getting a valid session from Login.aspx.

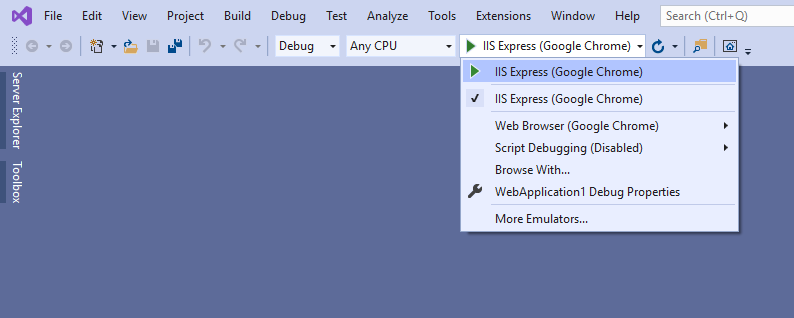
Create the LogoutMe method in HomePage.aspx.cs



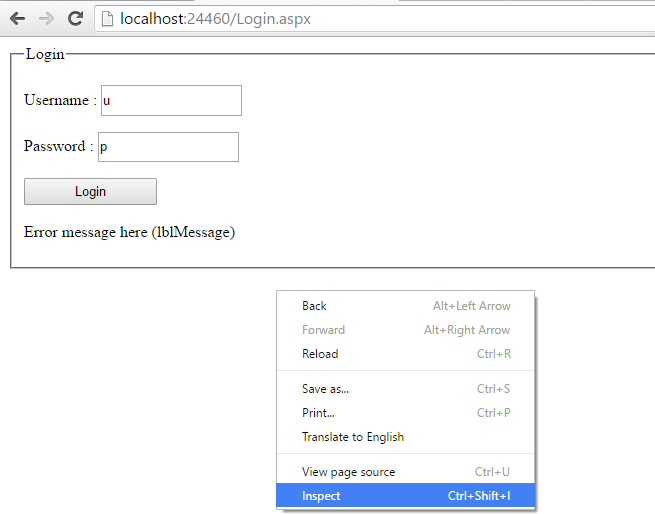
* Session.Clear removes all the variables stored in session and if user try to browse your site same sessionID will be used which was previously assigned to him.
* Session.Abandon removes all the variables stored in session, fire session\_end event and if user try to browse your site a new sessionID will be assigned to him.
* Session.Clear() is like removing books from the bookshelf whereas Session.Abandon() is like throwing the bookshelf itself.

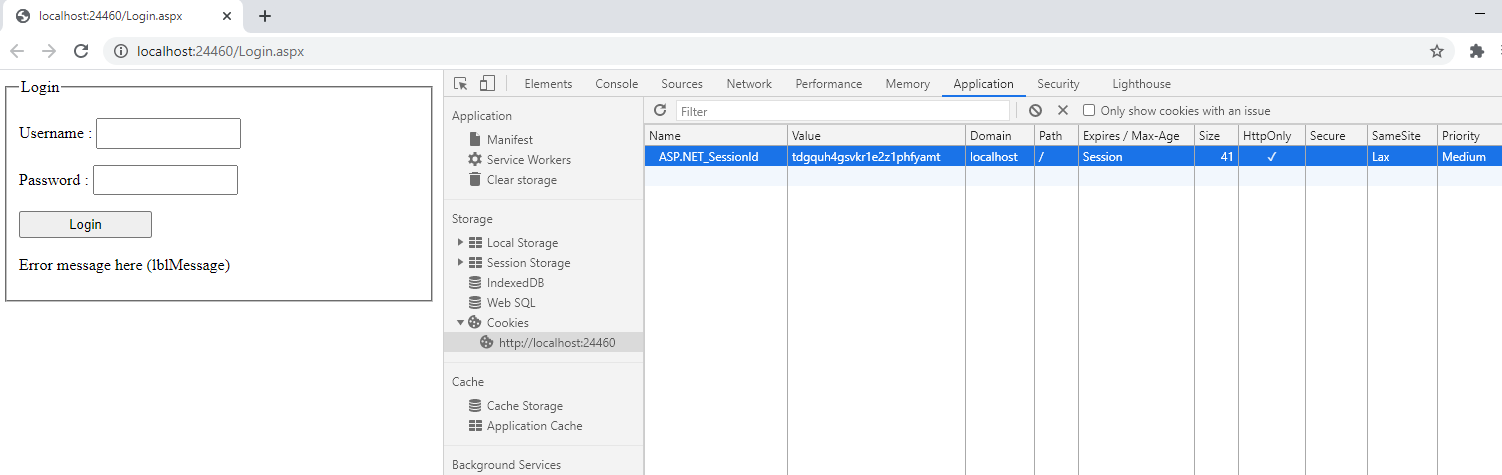
Output – Running the example

Set default browser of VS.NET to “Google Chrome’



Set Login.aspx as the ‘start page’ – Run Login.aspx. User userid ‘u’ and password ‘p’. Before you press the “Login” button, right-click on the browser page and selection “Inspect”.



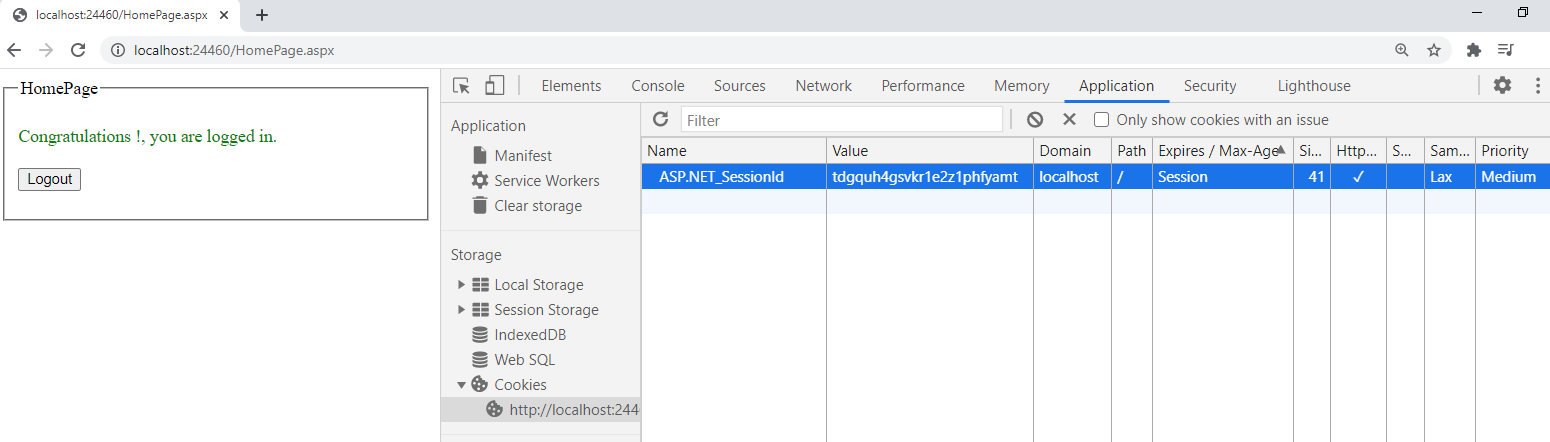


Inspect “Application” 🡪 Cookies 🡪 http://localhost..

Take note of the value of ASP.NET\_SessionId. If there no value for ASP.NET\_SessionId, click on the login button.

Notice that when the user has logged in, an ASP.NET\_SessionId cookie has been created.

ASP.NET\_SessionId BEFORE logout : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



Now click on the “Logout” button. Take note of the value of the ASP.NET\_SessionId.

ASP.NET\_SessionId AFTER logout : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Now when we click on the Logout button, even though the Session has been abandoned / removed, the ASP.NET\_SessionId cookie exists! Try to CLOSE the app and rerun the app again from VS.Net. Does the ASP.NET\_SessionId still exist and remains the same ? This is call “Session Fixation”.

How to fix the above vulnerability?

The immediate and easiest way to address the above issue is to explicitly remove the ASP.NET\_SessionId cookie in the Logout method. However, that does not solved all the problem.

To bullet proof this attack, we can create another cookie (e.g., AuthCookie) with a unique value and the same value can be stored into the Session as well. On every page load, we can match this cookie value with the Session value; if both matches, then let the use enter the application otherwise redirect to the Login page.

In the Logout function, ensure that you are removing this new Cookie “AuthCookie” as well. To remove this cookie, simply set its expiration date time to a few months earlier than the current date time.

To accomplish the above, we need to modify both the Login.aspx and HomePage.aspx file.

Fix LoginMe method in Login.aspx.cs

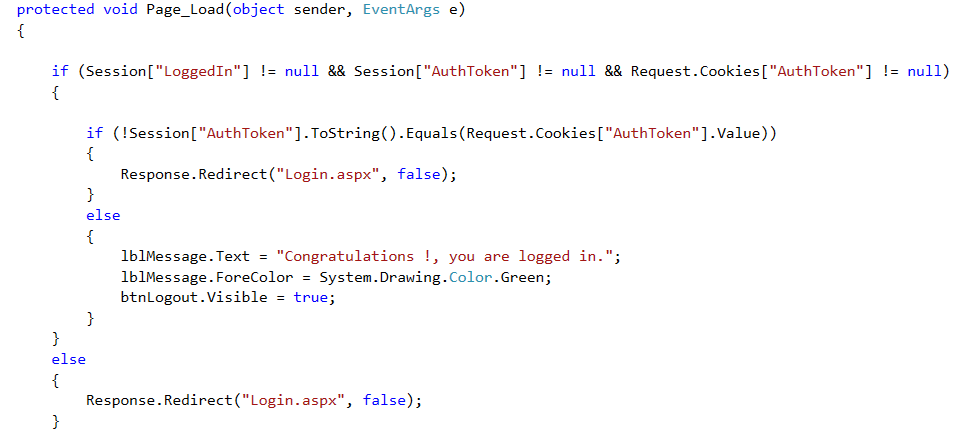
In this method, after setting the normal Session variable, we create a GUID (a unique value and almost impossible to guess) and save it as a new Session variable called AuthToken. The same GUID is then saved into a cookie named AuthToken.



Fix Page\_Load method in HomePage.aspx.cs

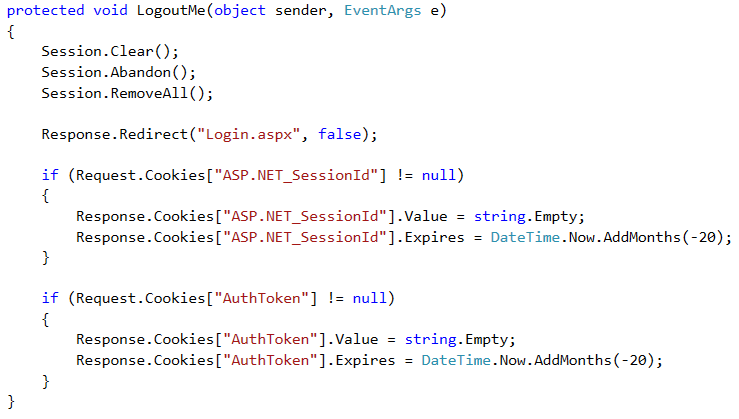
On every page load, we can match this cookie value with the Session value; if both matches, then let the use enter the application otherwise redirect to the Login page.

In the nutshell, we check for the normal LoggedIn session variable, the new Session variable called AuthToken and the new Cookie AuthToken. If all three of them are not null, then again we match the new Session variable AuthToken and the new Cookie AuthToken values. If both are **not** the same, then we redirect the user to the Login page.



Fix LogoutMe method in HomePage.aspx.cs

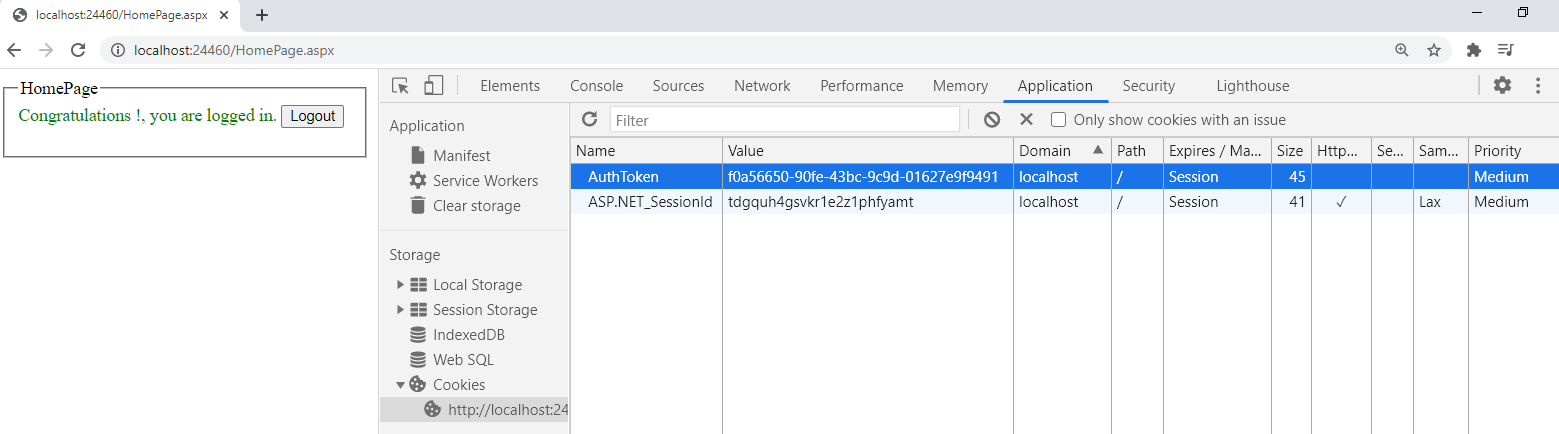
In the LogoutMe method, we first explicitly expire the ASP.NET\_SessionId cookie to make sure that this cookie is removed from the browser when the user clicks on the Logout button, and after that, we expire the AuthToken cookie as well.



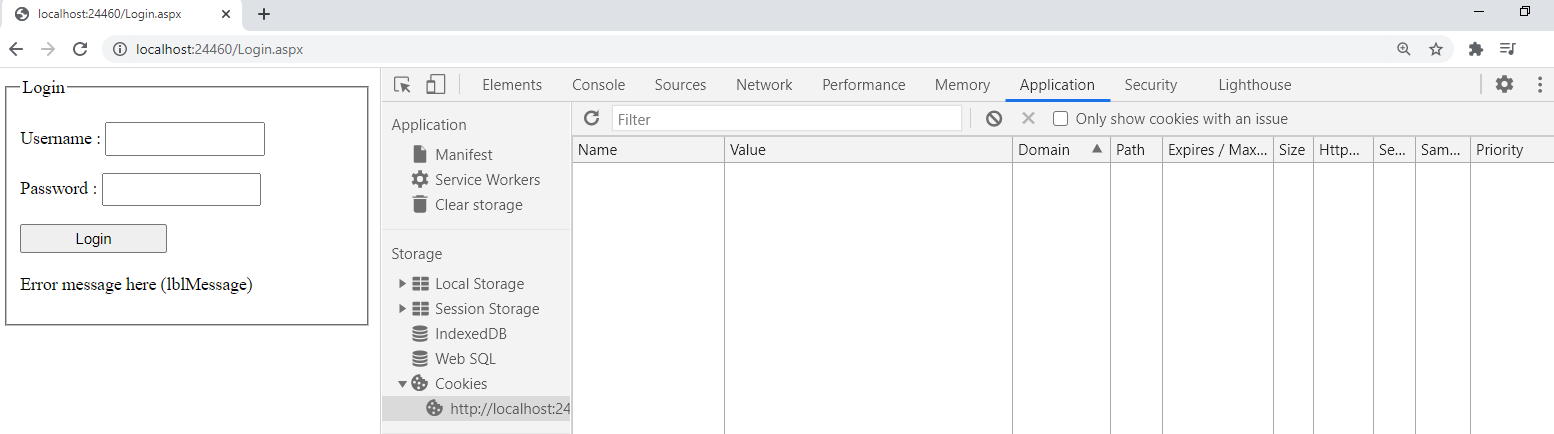
All these logic makes sure that even if the ASP.NET\_SessionId cookie value is known to the hijacker, he will not be able to login to the application as we are checking for the new Session value with the new cookie that is created by us and their GUID value is created by us. A hijacker can know the Cookie value but he can’t know the Session value that is stored in the web server level, and as this AuthToken value changes every time the user logs in, the older value will not work and the hijacker will not be able to even guess this value. Unless the new Session (AuthToken) value and the new Cookie (AuthToken) are the same, no one will be able to login to the application.

Verify the output

Verify that the issue is resolved by running the app again - login and logout from the sample web app.



After Logout



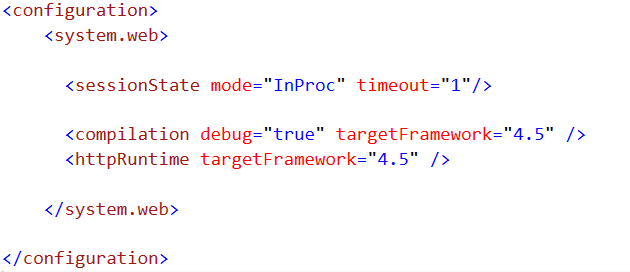
Please apply the Session Fixation to your project module as well as your upcoming practical assignment.

**Task 3.2 : Implement Session timeout**

In the web.config file add the code into the <system.web> section :

**<sessionState mode="InProc" timeout="1"/>**

Where timeout="1" is equivalent to 1 minute before the session timeout.



Login again to the system. Wait for over a minute then hit the refresh button on the browser.

The webpage will automatically redirect to login page.

*Challenge : try to implement a countdown timer for your session time out by using jQuery or javascript.*