**Which are the different ASP.NET authentication modes**

ASP.NET supports the following Authentication Providers

* Windows : Is used in conjunction with IIS authentication. Authentication is performed by IIS in one of three ways: basic, digest, or Integrated Windows Authentication. When IIS authentication is complete, ASP.NET uses the authenticated identity to authorize access
* Forms : The user provides credentials and submits the form.
* Passport : Centralized authentication service provided by Microsoft that offers a single logon and core profile services for member sites.
* None : No Authentication provided. This is default Authentication mode

<authentication

   mode= " [ Windows | Forms | Passport | None | Federated ] ">

</authentication>

## How to use mode "Windows"?

Change the authentication mode to **Windows**.

Windows Authentication mode provides the developer to authenticate a user based on Windows user accounts. This is the default authentication mode provided by ASP.Net. You can easily get the Identity of the user by using User.Identity.Name. This will return the computer name along with the user name. Windows authentication also provides IsInRole method to find the role of the user and than you can give permissions to the user depending on the role.

<authentication mode="Windows">

  <forms name=" AuthenticationDemo" loginUrl="logon.aspx" protection="All" path="/" timeout="30" />

</authentication>

Deny access to the anonymous user in the <authorization> section as follows:

<authorization>

     <deny users ="?" />

    <allow users = "\*" />

</authorization>

Other you can make a special client to access you project with windows authentication. Code like this (this case you can get value using 'User.Identity.Name', then you can use it to do other work you like.):

<authorization>

     <deny users ="?" />

</authorization>

**How to use mode "Forms"?**

Change the authentication mode to **Forms**.

Insert the <Forms> tag, and fill the appropriate attributes. (For more information about these attributes, refer to the MSDN documentation)

First you should specify a page and make sure all clients can found it. Code like this

<authentication mode="Forms">

    <forms name=" AuthenticationDemo" loginUrl="logon.aspx" protection="All" path="/" timeout="30"/>

</authentication>

Deny access to the anonymous user in the <authorization> section as follows:

<authorization>

    <deny users ="?" />

</authorization>

Second in that page you to validate the user's Id and Password. Code like this:

You can use one of two methods to generate the forms authentication cookie and redirect the user to an appropriate page in the **cmdLogin\_ServerClick**event. Sample code is provided for both scenarios. Use either of them according to your requirement.

**(1).** Call the **RedirectFromLoginPage** method to automatically generate the forms authentication cookie and redirect the user to an appropriate page in the **cmdLogin\_ServerClick** event:

private void cmdLogin\_ServerClick(object sender, System.EventArgs e)

{

     If (ValidateUser(txtUserName.Value,txtUserPass.Value) )

     {

          FormsAuthentication.RedirectFromLoginPage(txtUserName.Value, false);

     }

     else

     {

          Response.Redirect("logon.aspx", true);

     }

}

**(2).** Generate the authentication ticket, encrypt it, create a cookie, add it to the response, and redirect the user. This gives you more control in how you create the cookie. You can also include custom data along with the **FormsAuthenticationTicket** in this case.

Private void cmdLogin\_ServerClick(object sender, System.EventArgs e)

{

    if (ValidateUser(txtUserName.Value,txtUserPass.Value) )

    {

        FormsAuthenticationTicket tkt;

        string cookiestr;

        HttpCookie ck;

        tkt = new FormsAuthenticationTicket(1, txtUserName.Value, DateTime.Now,

DateTime.Now.AddMinutes(30), chkPersistCookie.Checked, "your custom data");

        cookiestr = FormsAuthentication.Encrypt(tkt);

        ck = new HttpCookie(FormsAuthentication.FormsCookieName, cookiestr);

        if (chkPersistCookie.Checked)

        ck.Expires=tkt.Expiration;

        ck.Path = FormsAuthentication.FormsCookiePath;

        Response.Cookies.Add(ck);

        string strRedirect;

        strRedirect = Request["ReturnUrl"];

        if (strRedirect==null)

        strRedirect = "default.aspx";

        Response.Redirect(strRedirect, true);

    }

    else

    Response.Redirect("logon.aspx", true);

}

## HTTP Methods

* **GET**
* **POST**
* **PUT**
* **HEAD**
* **DELETE**
* **PATCH**
* **OPTIONS**

## The GET Method

**GET is used to request data from a specified resource.**

**GET is one of the most common HTTP methods.**

Note that the query string (name/value pairs) is sent in the URL of a GET request:

/test/demo\_form.php?name1=value1&name2=value2

**Some other notes on GET requests:**

* GET requests can be cached
* GET requests remain in the browser history
* GET requests can be bookmarked
* GET requests should never be used when dealing with sensitive data
* GET requests have length restrictions
* GET requests is only used to request data (not modify)

## The POST Method

**POST is used to send data to a server to create/update a resource.**

The data sent to the server with POST is stored in the request body of the HTTP request:

POST /test/demo\_form.php HTTP/1.1  
Host: w3schools.com  
name1=value1&name2=value2

**POST is one of the most common HTTP methods.**

**Some other notes on POST requests:**

* POST requests are never cached
* POST requests do not remain in the browser history
* POST requests cannot be bookmarked
* POST requests have no restrictions on data length

## The PUT Method

**PUT is used to send data to a server to create/update a resource.**

The difference between POST and PUT is that PUT requests are idempotent. That is, calling the same PUT request multiple times will always produce the same result. In contrast, calling a POST request repeatedly have side effects of creating the same resource multiple times.

## The HEAD Method

**HEAD is almost identical to GET, but without the response body.**

In other words, if GET /users returns a list of users, then HEAD /users will make the same request but will not return the list of users.

HEAD requests are useful for checking what a GET request will return before actually making a GET request - like before downloading a large file or response body.

## The DELETE Method

**The DELETE method deletes the specified resource.**

## The OPTIONS Method

**The OPTIONS method describes the communication options for the target resource.**

## Compare GET vs. POST

The following table compares the two HTTP methods: GET and POST.

|  |  |  |
| --- | --- | --- |
|  | **GET** | **POST** |
| BACK button/Reload | Harmless | Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted) |
| Bookmarked | Can be bookmarked | Cannot be bookmarked |
| Cached | Can be cached | Not cached |
| Encoding type | application/x-www-form-urlencoded | application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data |
| History | Parameters remain in browser history | Parameters are not saved in browser history |
| Restrictions on data length | Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters) | No restrictions |
| Restrictions on data type | Only ASCII characters allowed | No restrictions. Binary data is also allowed |
| Security | GET is less secure compared to POST because data sent is part of the URL  Never use GET when sending passwords or other sensitive information! | POST is a little safer than GET because the parameters are not stored in browser history or in web server logs |
| Visibility | Data is visible to everyone in the URL | Data is not displayed in the URL |

Content Negotiation

<https://www.infoworld.com/article/3083273/how-to-work-with-content-negotiation-in-web-api.html>

<https://docs.microsoft.com/en-us/aspnet/web-api/overview/formats-and-model-binding/content-negotiation>