# ASP.NET Page Life Cycle Events

At each stage of the page life cycle, the page raises some events, which could be coded. An event handler is basically a function or subroutine, bound to the event, using declarative attributes such as Onclick or handle.

Following are the page life cycle events:

* **PreInit** - PreInit is the first event in page life cycle. It checks the IsPostBack property and determines whether the page is a postback. It sets the themes and master pages, creates dynamic controls, and gets and sets profile property values. This event can be handled by overloading the OnPreInit method or creating a Page\_PreInit handler.
* **Init** - Init event initializes the control property and the control tree is built. This event can be handled by overloading the OnInit method or creating a Page\_Init handler.
* **InitComplete** - InitComplete event allows tracking of view state. All the controls turn on view-state tracking.
* **LoadViewState** - LoadViewState event allows loading view state information into the controls.
* **LoadPostData** - During this phase, the contents of all the input fields are defined with the <form> tag are processed.
* **PreLoad** - PreLoad occurs before the post back data is loaded in the controls. This event can be handled by overloading the OnPreLoad method or creating a Page\_PreLoad handler.
* **Load** - The Load event is raised for the page first and then recursively for all child controls. The controls in the control tree are created. This event can be handled by overloading the OnLoad method or creating a Page\_Load handler.
* **LoadComplete** - The loading process is completed, control event handlers are run, and page validation takes place. This event can be handled by overloading the OnLoadComplete method or creating a Page\_LoadComplete handler
* **PreRender** - The PreRender event occurs just before the output is rendered. By handling this event, pages and controls can perform any updates before the output is rendered.
* **PreRenderComplete** - As the PreRender event is recursively fired for all child controls, this event ensures the completion of the pre-rendering phase.
* **SaveStateComplete** - State of control on the page is saved. Personalization, control state and view state information is saved. The HTML markup is generated. This stage can be handled by overriding the Render method or creating a Page\_Render handler.
* **UnLoad** - The UnLoad phase is the last phase of the page life cycle. It raises the UnLoad event for all controls recursively and lastly for the page itself. Final cleanup is done and all resources and references, such as database connections, are freed. This event can be handled by modifying the OnUnLoad method or creating a Page\_UnLoad handler.

## In which event of page cycle is the ViewState available?

After the Init() and before the Page\_Load().

## What is ASP.Net?

It is a framework developed by Microsoft on which we can develop new generation web sites using web forms(aspx), MVC, HTML, Javascript, CSS etc. Its successor of Microsoft Active Server Pages(ASP). Currently there is ASP.NET 4.0, which is used to develop web sites. There are various page extensions provided by Microsoft that are being used for web site development. Eg: aspx, asmx, ascx, ashx, cs, vb, html, XML etc.

**Advantages of ASP.NET,**

* Separation of Code from HTML
* Support for compiled languages
* Use services provided by the .NET Framework
* Graphical Development Environment
* Update files while the server is running
* XML-Based Configuration Files

# State Management

State Management can be defined as the technique or the way by which we can maintain / store the state of the page or application until the User's Session ends

## **Http is stateless, What does this mean?**

Stateless protocol is a communications protocol that treats each request as an independent transaction that is unrelated to any previous request so that the communication consists of independent pairs of requests and responses.

When a sender sends a request to the server, the server processes it and sends back a response to the sender. Then, the cycle is over and the state is closed. The next request, even from the same sender, is considered a new request, or a new state. In short, every request is new to the server.

## Session

 Session provide us the way of storing data in server memory. So we can store our page data into server memory and retrieve it back during page postbacks.What are the Advantage and disadvantage of Session?

**Advantages:**  
Session provide us the way of maintain user state/data.  
It is very easy to implement.  
One big advantage of session is that we can store any kind of object in it. :eg, datatabe, dataset.. etc  
By using session we don't need to worry about data collesp, because it store every client data separately.  
Session is secure and transparent from the user.

**Disadvantages:**Performance overhead in case of large volumes of data/user, because session data is stored in server memory.  
Overhead involved in serializing and de-serializing session data, because in the case of StateServer and SQLServer session modes, we need to serialize the objects before storing them.

What is Session ID in Asp.net?  
Ans: Asp.Net use 120 bit identifier to track each session. This is secure enough and can't be reverse engineered. When client communicate with server, only session id is transmitted, between them. When client request for data, ASP.NET looks on to session ID and retrieves corresponding data.

By default, the unique identifier for a session is stored in a non-expiring session cookie in the browser. You can specify that session identifiers not be stored in a cookie by setting the cookieless attribute to true in the sessionState configuration element.

## Difference between session and application

Session state and application variable are part of Asp.net server side state management concepts. If you want to save the user specific data use session state.

Sesions are is used to save the user specific data like UserID, User Role, etc. And can be accessible only by that user.

* Data stored in the SesionState will be cleared when the session expired(usually after 20min of inactivity when using session inproc).

**Application variable**

If you want to save the application level data then use application variable.

* Application variable are used to save the application level data. And can be accessible by all the users.
* Data stored in the Application variable will be cleared when the application restart.

## Session state management

1. In-Process
2. Out-of-Process.

**In-Process** stores the session in memory on the web server.

**Out-of-Process** Session state management stores data in an external server. The external server may be either a SQL Server or a State Server. All objects stored in session are required to be serializable for Out-of-Process state management.

## Session-State modes are 5 type:

* **InProc mode:** which stores session state in memory on the Web server. This is the default.
* **StateServer mode:** which stores session state in a separate process called the ASP.NET state service. This ensures that session state is preserved if the Web application is restarted and also makes session state available to multiple Web servers in a Web farm.
* **SQLServer** mode stores session state in a SQL Server database. This ensures that session state is preserved if the Web application is restarted and also makes session state available to multiple Web servers in a Web farm.
* **Custom mode:** which enables you to specify a custom storage provider.
* **Off mode:** which disables session state.

## Difference between session and caching?

**Answer:**The first main difference between session and caching is: a session is per-user based but caching is not per-user based, So what does that mean? Session data is stored at the user level but caching data is stored at the application level and shared by all the users. It means that it is simply session data that will be different for the various users for all the various users, session memory will be allocated differently on the server but for the caching only one memory will be allocated on the server and if one user modifies the data of the cache for all, the user data will be modified.  
  
For further info click on the link,

## What is caching?

Caching is a technique used to increase performance by keeping frequently accessed data or files in memory. The request for a cached file/data will be accessed from cache instead of actual location of that file.

**What are the different types of caching?**

ASP.NET has 3 kinds of caching :

1. Output Caching,
2. Fragment Caching,
3. Data Caching.

**Which type if caching will be used if we want to cache the portion of a page instead of whole page?**

**Fragment Caching:** It caches the portion of the page generated by the request. For that, we can create user controls with the below code:

<%@ OutputCache Duration="120" VaryByParam="CategoryID;SelectedID"%>

**How can we prevent browser from caching an ASPX page?**

We can SetNoStore on HttpCachePolicy object exposed by the Response object's Cache property:

Response.Cache.SetNoStore ();

Response.Write (DateTime.Now.ToLongTimeString ());

**What is Data Cache in ASP.NET and how to use?**

**Answer:**Data Cache is used to store frequently used data in the Cache memory. It's much more efficient to retrieve data from the data cache instead of database or other sources. We need to use System.Web.Caching namespace. The scope of the data caching is within the application domain unlike "session". Every user is able to access this object.  
  
When client requests to the server, server executes the stored procedure or function or select statements on the Sql Server database then it returns the response to the browser. If we run again same process will happen on the web server with sql server.  
  
**How to create data cache?**  
*Cache ["Employee"] = "DataSet Name"*  
  
We can create data caching use Cache Keyword. It's located in the System.Web.Caching namespace. It's just like assigning value to the variable.  
  
**How to remove a Data Cache?**  
We can remove Data Cache manually.

1. //We need to specify the cache name
2. Cache.Remove(String key);

**What are the different types of cookies in ASP.NET?**

**Session Cookie -** Resides on the client machine for a single session until the user does not log out.

**Persistent Cookie -** Resides on a user's machine for a period specified for its expiry, such as 10 days, one month, and never.

## Cookies

Cookies is a small piece of information stored on the client machine. This file is located on client machines "C:\Document and Settings\Currently\_Login user\Cookie" path.  Its is used to store user preference information like Username, Password,City and PhoneNo etc on client machines. We need to import namespace called  Systen.Web.HttpCookie before we use cookie.  
   
List of properties containing the HttpCookies Class,

1. Domain: Using these properties we can set the domain of the cookie.
2. Expires: This property sets the Expiration time of the cookies.
3. HasKeys: If the cookies have a subkey then it returns True.
4. Name: Contains the name of the Key.
5. Path: Contains the Virtual Path to be submitted with the Cookies.
6. Secured: If the cookies are to be passed in a secure connection then it only returns True.
7. Value: Contains the value of the cookies.

## Type of Cookies

1. Persist Cookie - A cookie has not have expired time Which is called as Persist Cookie
2. Non-Persist Cookie - A cookie has expired time Which is called as Non-Persist Cookie

 Explain Cookie-less Session in ASP.NET.

**Answer:** By default a session uses a cookie in the background. To enable a cookie-less session, we need to change some configuration in the Web.Config file. Follow these steps,

1. Open Web.Config file.
2. Add a <sessionState> tag under <system.web> tag.
3. Add an attribute "cookieless" in the <sessionState> tag and set its value to "AutoDetect" like below:
   1. <sessionState cookieless="AutoDetect" regenerateExpiredSessionId="true"/>

The possible values for "cookieless" attribute are,

* AutoDetect: Session uses background cookie if cookies are enabled. If cookies are disabled, then the URL is used to store session information.
* UseCookie: Session always use background cookie. This is default.
* UseDeviceProfile: Session uses background cookie if browser supports cookies else URL is used.
* UseUri: Session always use URL.

"regenerateExpiredSessionId" is used to ensure that if a cookieless url is expired a new new url is created with a new session. And if the same cookieless url is being used by multiple users an the same time, they all get a new regenerated session url.

## ****ASP.NET QueryString****

A query string is one of the techniques in Web applications to send data from one webform to another through the URL. A query string consists of two parts, field and value, and each of pair separated by ampersand (&).

The ?(question mark in a query string indicates the beginning of a query string and it's value.

There is a limit on the Query string length. Hence, Query strings cannot be used to send very long data.

Query strings are visible to the user, hence should not be used to send sensitive information suck as a username and password, unless encrypted

**Advantages**

* Simple to Implement
* Easy to get information from Query string.
* Used to send or read cross domain (from different domain).

**Disadvantages**

* Human Readable
* Client browser limit on URL length
* Cross paging functionality makes it redundant
* Easily modified by end user

## encrypted QueryString

The QueryString Parameter values will be first encrypted using AES Symmetric key (Same key) algorithm,

encoded (as the encrypted output might contain some special characters) and then will be sent to next page.

On the destination page the QueryString Parameter values will be first decoded and then decrypted using

AES Algorithm using the same key that was used for encryption.

## What is ViewState?

ViewState is used to retain the state of server-side objects between page post backs.

**Where the viewstate is stored after the page postback?**

ViewState is stored in a hidden field on the page at client side. ViewState is transported to the client and back to the server, and is not stored on the server or any other external source.

**How long the items in ViewState exists?**

They exist for the life of the current page.

## ****What is the difference between web config and machine config?****

1. Web config file is specific to a web application where as machine config is specific to a machine or server. There can be multiple web config files into an application where as we can have only one machine config file on a server

**What are the differences between ASP.NET HttpHandler and HttpModule?**

**Answer:** The user requests for a resource on web server. The web server examines the file name extension of the requested file, and determines which ISAPI extension should handle the request. Then the request is passed to the appropriate ISAPI extension. For example when an .aspx page is requested it is passed to ASP.NET page handler. Then Application domain is created and after that different ASP.NET objects like Httpcontext, HttpRequest, HttpResponse are created. Then instance of HttpApplication is created and also instance of any configured modules. One can register different events of HttpApplication class like BeginRequest, AuthenticateRequest, AuthorizeRequest, ProcessRequest etc.  
  
**HTTP Handler**  
HTTP Handler is the process which runs in response to a HTTP request. So whenever user requests a file it is processed by the handler based on the extension. So, custom http handlers are created when you need to special handling based on the file name extension. Let's consider an example to create RSS for a site. So, create a handler that generates RSS-formatted XML. Now bind the .rss extension to the custom handler.  
  
**HTTP Modules**  
HTTP Modules are plugged into the life cycle of a request. So when a request is processed it is passed through all the modules in the pipeline of the request. So generally http modules are used for,

* Security: For authenticating a request before the request is handled.
* Statistics and Logging: Since modules are called for every request they can be used for gathering statistics and for logging information.
* Custom header: Since response can be modified, one can add custom header information to the response.

**Navigations techniques in ASP.NET?**

Answer: Navigation can cause data loss if it not properly handled. We do have many techniques to transfer data from one page to another but every technique has its own importance and benefits.  
  
We will discuss the following techniques in this article.

* Response.Redirect
* Server.Transfer
* Server.Exceute
* Cross page posting

# Cross Page Posting & Redirect Permananent & auto wireUp

## ****What is Cross Page Posting?****

When we click submit button on a web page, the page post the data to the same page. The technique in which we post the data to different pages is called Cross Page posting. This can be achieved by setting POSTBACKURL property of the button that causes the postback. Findcontrol method of PreviousPage can be used to get the posted values on the page to which the page has been posted.

## What are Web Services in ASP.NET?

**Answer:** A Web Service is a software program that uses XML to exchange information with other software via common internet protocols. In a simple sense, Web Services are a way of interacting with objects over the Internet.  
  
A web service is,

* Language Independent.
* Protocol Independent.
* Platform Independent.
* It assumes a stateless service architecture.
* Scalable (e.g. multiplying two numbers together to an entire customer-relationship management system).
* Programmable (encapsulates a task).
* Based on XML (open, text-based standard).
* Self-describing (metadata for access and use).
* Discoverable (search and locate in registries)- ability of applications and developers to search for and locate desired Web services through registries. This is based on UDDI.

## ****What is RedirectPermanent in ASP.Net?****

RedirectPermanent Performs a permanent redirection from the requested URL to the specified URL. Once the redirection is done, it also returns 301 Moved Permanently responses

## What is use of the AutoEventWireup attribute in the Page directive ?

The AutoEventWireUp is a boolean attribute that allows automatic wireup of page events when this attribute is set to true on the page. It is set to True by default for a C# web form whereas it is set as False for VB.NET forms.

## ****Differentiate strong typing and weak typing****

In strong typing, the data types of variable are checked at compile time. On the other hand, in case of weak typing the variable data types are checked at runtime. In case of strong typing, there is no chance of compilation error. Scripts use weak typing and hence issues arises at runtime.

## How do you secure your connection string information?

By using the Protected Configuration feature.

## How can I configure ASP.NET applications that are running on a remote machine?

You can use the Web Site Administration Tool to configure remote websites.

## How does Application Pools work in IIS 6.0?

Every application within an application pool shares the same worker process. Because each worker process operates as a separate instance of the worker process executable, W3wp.exe, the worker process that services one application pool is separated from the worker process that services another. Each separate worker process provides a process boundary so that when an application is assigned to one application pool, problems in other application pools do not affect the application. This ensures that if a worker process fails, it does not affect the applications running in other application pools.

## What are the different validators in ASP.NET?

1. Required field Validator
2. Range Validator
3. Compare Validator
4. Custom Validator
5. Regular expression Validator
6. Summary Validator

## How you can add an event handler?

Using the Attributes property of server side contro

**Can we add code files of different languages in App\_Code folder?**

No. The code files must be in same language to be kept in App\_code folder.

**What is Protected Configuration?**

It is a feature used to secure connection string information.

**What is the good practice to implement validations in aspx page?**

Client-side validation is the best way to validate data of a web page. It reduces the network traffic and saves server resources

## What are the event handlers that we can have in Global.asax file?

**Application Events:** Application\_Start , Application\_End, Application\_AcquireRequestState, Application\_AuthenticateRequest, Application\_AuthorizeRequest, Application\_BeginRequest, Application\_Disposed, Application\_EndRequest, Application\_Error, Application\_PostRequestHandlerExecute, Application\_PreRequestHandlerExecute,Application\_PreSendRequestContent, Application\_PreSendRequestHeaders, Application\_ReleaseRequestState, Application\_ResolveRequestCache, Application\_UpdateRequestCache

## What is the Web.config file in ASP?

**Answer:** Configuration file is used to manage various settings that define a website. The settings are stored in XML files that are separate from your application code. In this way you can configure settings independently from your code. Generally a website contains a single Web.config file stored inside the application root directory. However there can be many configuration files that manage settings at various levels within an application.  
  
Usage of configuration file  
  
ASP.NET Configuration system is used to describe the properties and behaviors of various aspects of ASP.NET applications. Configuration files help you to manage the settings related to your website. Each file is an XML file (with the extension .config) that contains a set of configuration elements. Configuration information is stored in XML-based text files.  
  
Benefits of XML-based Configuration files

* ASP.NET Configuration system is extensible and application specific information can be stored and retrieved easily. It is human-readable.
* You need not restart the web server when the settings are changed in configuration files. ASP.NET automatically detects the changes and applies them to the running ASP.NET application.
* You can use any standard text editor or XML parser to create and edit ASP.NET configuration files.

## Explain GridView control in ASP.NET?

**Answer:**The GridView control displays the values of a data source in a table. Each column represents a field, while each row represents a record. The GridView control supports the following features,

# Security & ADO.Net

## Explain role based security ?

Role Based Security used to implement security based on roles assigned to user groups in the organization.

Then we can allow or deny users based on their role in the organization. Windows defines several built-in groups, including Administrators, Users, and Guests.

<AUTHORIZATION>< authorization >

< allow roles="Domain\_Name\Administrators" / > < !-- Allow Administrators in domain. -- >

< deny users="\*" / > < !-- Deny anyone else. -- >

< /authorization >

## Explain the working of passport authentication.

First of all it checks passport authentication cookie. If the cookie is not available then the application redirects the user to Passport Sign on page. Passport service authenticates the user details on sign on page and if valid then stores the authenticated cookie on client machine and then redirect the user to requested page

## What are the advantages of Passport authentication?

All the websites can be accessed using single login credentials. So no need to remember login credentials for each web site.

Users can maintain his/ her information in a single location.

## What are the asp.net Security Controls?

* <asp:Login>: Provides a standard login capability that allows the users to enter their credentials
* <asp:LoginName>: Allows you to display the name of the logged-in user
* <asp:LoginStatus>: Displays whether the user is authenticated or not
* <asp:LoginView>: Provides various login views depending on the selected template
* <asp:PasswordRecovery>: email the users their lost password

## How do you register JavaScript for webcontrols ?

We can register javascript for controls using <CONTROL -name>Attribtues.Add(scriptname,scripttext) method.

## List all templates of the Repeater control.

* ItemTemplate
* AlternatingltemTemplate
* SeparatorTemplate
* HeaderTemplate
* FooterTemplate

## List the major built-in objects in ASP.NET?

* Application
* Request
* Response
* Server
* Session
* Context
* Trace

## What is the appSettings Section in the web.config file?

The appSettings block in web config file sets the user-defined values for the whole application.

## What are the components of ADO.NET?

The components of ADO.Net are Dataset, Data Reader, Data Adaptor, Command, connection.

## What is the difference between ExecuteScalar and ExecuteNonQuery?

ExecuteScalar returns output value where as ExecuteNonQuery does not return any value but the number of rows affected by the query. ExecuteScalar used for fetching a single value and ExecuteNonQuery used to execute Insert and Update statements.

What is tracing in .NET?

**Answer:** Tracing helps to see the information of issues at the runtime of the application. By default Tracing is disabled.  
  
Tracing has the following important features:

1. We can see the execution path of the page and application using the debug statement.
2. We can access and manipulate trace messages programmatically.
3. We can see the most recent tracing of the data.

Tracing can be done with the following 2 types.

***Page Level***When the trace output is displayed on the page and for the page-level tracing we need to set the property of tracing at the page level.  
  
*<%@ Page Trace="true" Language="C#"*

***Application Level***In Application-Level tracing the information is stored for each request of the application. The default number of requests to store is 10. But if you want to increase the number of requests and discard the older request and display a recent request then you need to set the property in the web.config file.  
  
*<trace enabled="true"/>*

Global Application Class(Global.asax)  
  
It is a Class which consists of event handlers which executes the code implicitly whenever a relevant task has been performed on the web server.Design:

# **Authentication and Autherization & code behind&Inline**

## What is the code behind and Inline Code?

**Answer**  
**Code Behind**  
Code Behind refers to the code for an ASP.NET Web page that is written in a separate class file that can have the extension of .aspx.cs or .aspx.vb depending on the language used. Here the code is compiled into a separate class from which the .aspx file derives. You can write the code in a separate .cs or .vb code file for each .aspx page. One major point of Code Behind is that the code for all the Web pages is compiled into a DLL file that allows the web pages to be hosted free from any Inline Server Code.  
  
**Inline Code**  
Inline Code refers to the code that is written inside an ASP.NET Web Page that has an extension of .aspx. It allows the code to be written along with the HTML source code using a <Script> tag. It's major point is that since it's physically in the .aspx file it's deployed with the Web Form page whenever the Web Page is deployed.

## Authentication providers

* **The windows Authentication** provider lets you authenticates users based on their windows accounts. This provider uses IIS to perform the authentication and then passes the authenticated identity to your code. This is the default provided for ASP.net.
* **The passport authentication** provider uses Microsoft's passport service to authenticate users.
* **The forms authentication** provider uses custom HTML forms to collect authentication information and lets you use your own logic to authenticate users. The user's credentials are stored in a cookie for use during the session.

Selecting an authentication provider is as simple as making an entry in the web.config file for the application. You can use one of these entries to select the corresponding built in authentication provider:

1. <authentication mode="windows">
2. authentication mode="passport">
3. <authentication mode="forms">

## Windows authentication and IIS

If you select windows authentication for your ASP.NET application, you also have to configure authentication within IIS. This is because IIS provides Windows authentication. IIS gives you a choice for four different authentication methods:

**Anonymous, basic digest, and windows integrated**

If you select anonymous authentication, IIS doesn't perform any authentication, Any one is allowed to access the ASP.NET application.

If you select basic authentication, users must provide a windows username and password to connect. How ever this information is sent over the network in clear text, which makes basic authentication very much insecure over the internet.

If you select digest authentication, users must still provide a windows user name and password to connect. However the password is hashed before it is sent across the network. Digest authentication requires that all users be running Internet Explorer 5 or later and that windows accounts to stored in active directory.

If you select windows integrated authentication, passwords never cross the network. Users must still have a username and password, but the application uses either the Kerberos or challenge/response protocols authenticate the user

## Configuring Authorization

After your application has authenticated users, you can proceed to authorize their access to resources. But there is a question to answer first: Just who is the user to whom your are grating access? It turns out that there are different answers to that question, depending on whether you implement impersonation. **Impersonation** is a technique that allows the ASP.NET process to act as the authenticated user, or as an arbitrary specified user.

ASP.NET impersonation is controlled by entries in the applications web.config file. The default setting is "no impersonation". You can explicitly specify that ASP.NET shouldn't use impersonation by including the following code in the file

<identity impersonate="false"/>

**Forms authentication**

Forms authentication provides you with a way to handle authentication using your own custom logic with in an ASP.NET application. The following applies if you choose forms authentication.

1. When a user requests a page for the application, ASP.NET checks for the presence of a special session cookie. If the cookie is present, ASP.NET assumes the user is authenticated and processes the request.
2. If the cookie isn't present, ASP.NET redirects the user to a web form you provide
3. You can carry out whatever authentication, checks you like in your form. When the user is authenticated, you indicate this to ASP.NET by setting a property, which creates the special cookie to handle subsequent requests.
4. <configuration>
5. <appSettings>
6. <add key="ValidationSettings:UnobtrusiveValidationMode" value="None" />
7. </appSettings>
8. <system.web>
9. <compilation debug="true" targetFramework="4.5" />
10. <authentication mode="Forms">
11. <forms loginUrl="login.aspx" defaultUrl="welcome.aspx">
12. <credentials passwordFormat="Clear">
13. <user name="abhishek" password="abhi@123"/>
14. <user name="Kantesh" password="sinha@123" />
15. </credentials>
16. </forms>
17. </authentication>
18. <authorization>
19. <deny users="?"/>
20. </authorization>
21. <httpRuntime targetFramework="4.5" />
22. </system.web>
23. </configuration>

Here we write the code for Login button in*login.aspx.cs* page.

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Web;
5. **using** System.Web.UI;
6. **using** System.Web.UI.WebControls;
7. **using** System.Web.Security;
9. **namespace** FormAuth {
10. **public** partial **class** Login: System.Web.UI.Page {
11. **protected** **void** Page\_Load(**object** sender, EventArgs e) {
13. }
15. **protected** **void** Login\_Click(**object** sender, EventArgs e) {
16. **if** (FormsAuthentication.Authenticate(UserName.Text, UserPass.Text)) {
17. FormsAuthentication.RedirectFromLoginPage(UserName.Text, chkboxPersist.Checked);
18. } **else** {
19. Msg.Text = "Invalid User Name and/or Password";
20. }
21. }
22. }

## Form Authentication in MVC

1. **namespace** MvcFormAuthentication\_Demo.Controllers
2. {
4. **public** **class** HomeController : Controller
5. {
6. **private** **readonly** EmployeeContext \_dbContext = **new** EmployeeContext();
7. **public** ActionResult Index()
8. {
9. **return** View();
10. }
11. **public** ActionResult Login()
12. {
13. **return** View();
14. }
15. [HttpPost]
16. [ValidateAntiForgeryToken]
17. **public** ActionResult Login(UserModel user)
18. {
19. **if** (ModelState.IsValid)
20. {
21. **bool** IsValidUser = \_dbContext.Users
22. .Any(u => u.Username.ToLower() == user
23. .Username.ToLower() && user
24. .Password == user.Password);
26. **if** (IsValidUser)
27. {
28. FormsAuthentication.SetAuthCookie(user.Username, **false**);
29. **return** RedirectToAction("Index", "Employee");
30. }
31. }
32. ModelState.AddModelError("", "invalid Username or Password");
33. **return** View();
34. }
35. **public** ActionResult Register()
36. {
37. **return** View();
38. }
39. [HttpPost]
40. [ValidateAntiForgeryToken]
41. **public** ActionResult Register(User registerUser)
42. {
43. **if** (ModelState.IsValid)
44. {
45. \_dbContext.Users.Add(registerUser);
46. \_dbContext.SaveChanges();
47. **return** RedirectToAction("Login");
49. }
50. **return** View();
51. }
52. **public** ActionResult Logout()
53. {
54. FormsAuthentication.SignOut();
55. **return** RedirectToAction("Login");
56. }
57. }
58. }

# **What Is a JWT Token?**

**What Is Stateless?**

When a sender sends a request to the server, the server processes it and sends back a response to the sender. Then, the cycle is over and the state is closed. The next request, even from the same sender, is considered a new request, or a new state. In short, every request is new to the server.

In this situation, if a sender requests a secured resource, then the sender has to be authenticated and verified of its genuineness, and they should have true permission to access the resource. This is done through the authentication by credentials.

Now, as every request is stateless, is it like we need to send the username/password every time. Yes, it is quite possible but not at all feasible. To deal with this problem, there are many mechanisms that can be used, and one of them is token-based authentication.

## ****What Is Token-Based Authentication?****

In this mechanism, when the server first receives a request with the username and password, it validates it with the values stored in its database (collected through registration) and allows the request to access the secured resource, but additionally, in the response, the server sends an authentication token and this token is stored in the database as well. When the subsequent request is sent by the same user, instead of the sending the username and password, this authentication token is being sent in the request packet. This token is validated against the token stored in the database for its validity and authenticity, and based on that, the user is allowed access.

There are many tokens based authentication available, a JSON web token (JWT) is one of them. OAuth is also another well-known mechanism. JWT follows a different technique for making it more secure because it is signed. This helps in detecting any modifications.

## ****What Is a JWT Token?****

A JSON web token (JWT) is an open standard (RFC 7519) that is a compact and self-contained way for securely transmitting information between systems as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA

### **Secure**

JWT digitally signs the content, and since only the server knows the secret to decrypt the information, it is vastly more secure. If anything is changed between the transmission, then the system can detect it and will provide an invalid signature error.

### **Compact**

The JWT is simply a JSON object that is transmitted between systems with all its information, which is encoded with an algorithm. Because of this size, it can be sent through URL, POST Parameter, or inside HTTP Headers.

### **Faster**

As this is compact, and being just an object, the transmission speed is very low. Also, it contains much-required information that eliminates the DB calls to fetch information, thus reducing overhead time.

### **Self-Contained**

Because the JSON token contains sufficient information about the user, it does not need to query the database more than once.

### **Easy**

As JSON is less verbose than XML, it is compact and smaller in size. We have great debugging tools for seeing the information of JWT. (Note: While this is a great feature, we must keep secret information from being leaked or made public — so, restrain from putting secure data in a token.)

## **When Should We Use JWT Token?**

### **Authentication and Authorization**

Once the user is logged in and authenticated by the server, then the JWT token is generated and passed in response, and in each subsequent request, the token is passed to the server. This JWT token contains the information for the user's access and permission, which is part of the authorization.

### **Information Exchange**

This can also be used for transmitting information between two systems, as JWT tokens can be signed with public-private key pairs.

That is enough information, for now, about JWTs, now let's understand the structure of the JWT.

## ****JWT Structure****

Basically, JWT Token is a pure JSON object. So, it has its own structure where we put the information.

### **Header**

This contains information about the algorithm used for signing and the type of the token.

This is kind of metadata of the token is generated by the server. This contains an algorithm used in signing the token and the type of token used. Once this object is created, it is encoded with Base64Url to form the first part of the JWT token

### **Payload**

This usually contains the information about a dataset, which the server wants to send to the client so it can use it when the token is passed back to the server. This data can typically count as user data. Also, this should not contain any sensitive information about the user, e.g. password, email, etc.

**claims**

The second part of the token is the payload, which contains the claims. There are three types of claims: registered, public, and private claims.

**Registered claims**: These are a set of predefined claims, which are not mandatory but recommended, to provide a set of useful, interoperable claims. Some of them are:

* iss (issuer),
* exp (expiration time),
* sub (subject),
* aud (audience),
* and more.

claim names are only three characters long, as JWT is meant to be compact.

**Public claims**: These can be defined at will by those using JWTs.

**Private claims**: These are the custom claims created to share information between systems.

Once this object is created, it is encoded with Base64Url to form the second part of the JWT token.

### **Signature**

It is created by encoding the HEADER and the PAYLOAD with a secret in the algorithm specified in the header and sign the same with the secret. The secret is only known to the server.

The signature is formed by concatenating the encoded headers and the encoded payload, and then, we sign it using the secret that is only known to the server with the algorithm specified in the header. If there is any alteration in the content of the JWT token, then it is marked as "Invalid." If the secret is also not correct, then also it is marked as "Invalid" and cannot be trusted.

## ****How Does the JWT Token Work?****

1. The user logs in by providing the username and password for the first time.

2. The server authenticates the information based on the username and password provided by the user and retrieves the user information from the database.

3. Invokes the JWT libraries to create a JSON web token (JWT) with this information, along with access permission and authority levels. sensitive information should not be sent in the token as the token can be decoded. Before sending the token, the token is stored in the database associated with the user.

4. With the subsequent calls to the server, the user sends the JWT as a header to the server.

5. The server then encodes the information and validates the information before granting access to the user for secure data in the server.

6. Once the user logged out, the JWT token is cleared out.

# JWT Authentication Steps

The first step is to configure JWT based authentication in our project. To do this, we need to register a JWT authentication schema by using "AddAuthentication" method and specifying JwtBearerDefaults.AuthenticationScheme. Here, we configure the authentication schema with JWT bearer options.

1. **public** **void** ConfigureServices(IServiceCollection services)
2. {
3. services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
4. .AddJwtBearer(options =>
5. {
6. options.TokenValidationParameters = **new** TokenValidationParameters
7. {
8. ValidateIssuer = **true**,
9. ValidateAudience = **true**,
10. ValidateLifetime = **true**,
11. ValidateIssuerSigningKey = **true**,
12. ValidIssuer = Configuration["Jwt:Issuer"],
13. ValidAudience = Configuration["Jwt:Issuer"],
14. IssuerSigningKey = **new** SymmetricSecurityKey(Encoding.UTF8.GetBytes(Configuration["Jwt:Key"]))
15. };
16. });
17. services.AddMvc();
18. }

* Validate the server (ValidateIssuer = true) that generates the token.
* Validate the recipient of the token is authorized to receive (ValidateAudience = true)
* Check if the token is not expired and the signing key of the issuer is valid (ValidateLifetime = true)
* Validate signature of the token (ValidateIssuerSigningKey = true)
* Additionally, we specify the values for the issuer, audience, signing key. In this example, I have stored these values in appsettings.json file.

## AppSetting.Json

1. {
2. "Jwt": {
3. "Key": "ThisismySecretKey",
4. "Issuer": "Test.com"
5. }
6. }

configure a JWT based authentication service. The next step is to make the authentication service is available to the application. To do this, we need to call app.UseAuthentication() method in the Configure method of startup class. The UseAuthentication method is called before UseMvc method.

## Generate JSON Web Token

we have to create a LoginController and Login method within this controller, which is responsible to generate the JWT. we have to marke this method with the AllowAnonymous attribute to bypass the authentication. This method expects the Usermodel object for Username and Password.

we have to create the "AuthenticateUser" method, which is responsible to validate the user credential and returns to the UserModel. If the "AuthenticateUser" method returns the user model, API generates the new token by using the "GenerateJSONWebToken" method.

Finally, JwtSecurityTokenHandler.WriteToken method is used to generate the JWT. This method expects an object of the JwtSecurityToken class.

1. **using** Microsoft.AspNetCore.Authorization;
2. **using** Microsoft.AspNetCore.Mvc;
3. **using** Microsoft.Extensions.Configuration;
4. **using** Microsoft.IdentityModel.Tokens;
5. **using** System;
6. **using** System.IdentityModel.Tokens.Jwt;
7. **using** System.Security.Claims;
8. **using** System.Text;
10. **namespace** JWTAuthentication.Controllers
11. {
12. [Route("api/[controller]")]
13. [ApiController]
14. **public** **class** LoginController : Controller
15. {
16. **private** IConfiguration \_config;
18. **public** LoginController(IConfiguration config)
19. {
20. \_config = config;
21. }
22. [AllowAnonymous]
23. [HttpPost]
24. **public** IActionResult Login([FromBody]UserModel login)
25. {
26. IActionResult response = Unauthorized();
27. var user = AuthenticateUser(login);
29. **if** (user != **null**)
30. {
31. var tokenString = GenerateJSONWebToken(user);
32. response = Ok(**new** { token = tokenString });
33. }
35. **return** response;
36. }
38. **private** **string** GenerateJSONWebToken(UserModel userInfo)
39. {
40. var securityKey = **new** SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_config["Jwt:Key"]));
41. var credentials = **new** SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);
43. var token = **new** JwtSecurityToken(\_config["Jwt:Issuer"],
44. \_config["Jwt:Issuer"],
45. **null**,
46. expires: DateTime.Now.AddMinutes(120),
47. signingCredentials: credentials);
49. **return** **new** JwtSecurityTokenHandler().WriteToken(token);
50. }
52. **private** UserModel AuthenticateUser(UserModel login)
53. {
54. UserModel user = **null**;
56. //Validate the User Credentials
57. //Demo Purpose, I have Passed HardCoded User Information
58. **if** (login.Username == "Jignesh")
59. {
60. user = **new** UserModel { Username = "Jignesh Trivedi", EmailAddress = "test.btest@gmail.com" };
61. }
62. **return** user;
63. }
64. }
65. }
66. Once, we have enabled the JWT based authentication, I have created a simple Web API method that returns a list of value strings when invoked with an HTTP GET request. Here, I have marked this method with the authorize attribute, so that this endpoint will trigger the validation check of the token passed with an HTTP request.
68. If we call this method without a token, we will get 401 (UnAuthorizedAccess) HTTP status code as a response. If we want to bypass the authentication for any method, we can mark that method with the AllowAnonymous attribute.

# **Microservices**

Microservice is an approach to create small services each running in their own space and can communicate via messaging. These are independent services directly calling their own database.

## Advantages and disadvantages of Microservice

### **Easier to Build and Maintain Apps**

The key principle of microservices is simplicity. Applications become easier to build and maintain when they’re split into a set of smaller, composable fragments. Managing the code also becomes less painful because each microservice is, in fact, a separate chunk of code. Services can be implemented using different programming languages, databases and software environments. This allows each service to be deployed, rebuilt, re-deployed and managed independently.

* Services can be written in different programming language and can be accessed by using any framework.
* Independently develop, deploy, redeploy, version and scale component services in seconds without compromising the integrity of an application
* Better fault isolation keeps other services to work even though on got failed.
* Zero downtime upgrades.
* Services can be of from different servers or even different datacenters.
* Interaction with other services in a well-defined protocol
* Monitor, capture, and report health diagnostics
* Reliable and self-healing
* Supports continuous integration and delivery
* Easy to transfer knowledge to the new team member
* Easy to integrate with third parties

## Disadvantages of Microservices

* The additional complexity for implementation of an inter-process communication mechanism between services.
* Writing automated tests involving multiple services is challenging and It can be difficult to create consistent testing environments.
* Requires high level of automation to manage multiple instances of different types of services in production.
* Everyone has to manage eventual consistency as maintaining string consistency becomes extremely difficult.
* Managing multiple databases and their transactions are difficult.
* Inter-process calls are slow.
* Debugging will become difficult.
* Complexity in DevOps.
* Production monitoring cost is higher.
* Formal documentation overhead.
* Lack of governance.

**Message communication**

Unlike HTTP communication, the services involved do not directly communicate with each other. Instead, the services push messages to a message broker that other services subscribe to. This eliminates a lot of complexity associated with HTTP communication.

It doesn’t require services to know how to talk to one another; it removes the need for services to call each other directly. Instead, all services know of a message broker, and they push messages to that broker. Other services can choose to subscribe to the messages in the broker that they care about.

If our application is in Amazon Web Services, we can use Simple Notification Service (SNS) as our message broker. Now ServiceA can push messages to an SNS topic that ServiceB listens on.

* function asyncProcessMessage(name: string): Promise<string> {
* /\*\* do some ServiceA business logic
* ....
* ....
* \*/
* /\*\*
* \* send message to SNS that ServiceB is listening on
* \*/
* let snsClient = new AWS.SNS()
* let params = {
* Message: JSON.stringify({
* 'data': 'our message data'
* }),
* TopicArn: 'our-sns-topic-message-broker'
* }
* return snsClient.publish(params)
* .then((response) => {
* return response.MessageId
* })
* }
* ServiceB listens for messages on the SNS topic. When it receives one it cares about, it executes its business logic.

This introduces its own complexities. Notice that ServiceA no longer receives a status URL to check on progress. This is because we only know that the message has been sent, not that ServiceB has received it.

This could be solved in many different ways. One way is to return the MessageId to the caller. It can use that to query ServiceB, which will store the MessageId of the messages it has received.

Take note that there is still some coupling between the two services using this pattern. For instance, ServiceB and ServiceA must agree on what the message structure is and what it contains.

# What is REST architecture?

REST stands for REpresentational State Transfer. REST is web standards based architecture and uses HTTP Protocol. It revolves around resource where every component is a resource and a resource is accessed by a common interface using HTTP standard methods. In REST architecture, a REST Server simply provides access to resources and REST client accesses and modifies the resources. Here each resource is identified by URIs/ global IDs. REST uses various representation to represent a resource like text, JSON, XML. JSON is the most popular one.

### **HTTP methods**

Following four HTTP methods are commonly used in REST based architecture.

* **GET** − Provides a read only access to a resource.
* **POST** − Used to create a new resource.
* **DELETE** − Used to remove a resource.
* **PUT** − Used to update a existing resource or create a new resource.

## 

# Assembly in Dot Net

### **What is an Assembly?**

* Assemblies are the basic building blocks required for any application to work in .net realm.
* Assemblies provides collection of types and resources that work together to form a logical unit of functionality.
* Assembly is a set of one or more modules and classes compiled in MSIL and Metadata.

### **What all are the components of Assembly?**

* Assembly Manifest
* MSIL source code
* Type Metadata
* Resources

### **What all are the different type of Assemblies?**

* Private Assembly
  + It is used by Single Application
  + Private Assemblies are kept in a local folder in which client application is installed
* Public Assembly
  + Public Assembly is shared by multiple application
  + Shared Assembly resides in GAC

### **What is Assembly Manifest?**

Assemblies has a special logical unit called Manifest. It contains:

* Assembly name
* Version number
* Culture
* Strong Name information
* Type Reference Information
* List of all files in Assembly
* Information on referenced assemblies

### **What is GAC?**

* GAC is a central repository in a system in which assemblies are registered to shared between application.
* GACUtil.exe is used to view and change the content of GAC in system
* GAC can contain multiple versions on .net assemblies
* Thegautil.exe/I<assembly\_name> is used to install assembly in GAC

### **What is Metadata?**

* Assembly metadata describes every data type and member defined in the code.
* it stores name, version, culture, public key of an Assembly
* Also it stores the description of types like name, visibility, base class, interfaces, implemented etc.
* Metadata of an assembly is sharable among applications that execute on different platforms

### **What is Native Image Generator?**

The Native Image Generator is a tool that creates a native image from an Assembly and stores that image to native image cache on the Whenever an assembly runs, this native image is automatically used to compile original assembly. This improves the performance of the application by loading and executing an assembly faster.

### **What is Satellite Assembly?**

Satellite assemblies are assemblies that is used to deploy culture and language for an application. A separate product id is assigned to each language and a satellite assembly is installed in language specific sub directory.

### **How do you add/remove assembly from GAC?**

you can add assembly by using below syntax:

**gacutil /i [assemblyName | assemblyPath]**

you can remove assembly by using below syntax:

**gacutil /u [assemblyName | assemblyPath]**

### **What is the difference between EXE and DLL?**

**EXE**:

* Executable file, can run independently
* It runs in a separate process
* It can’t be reused in application
* it has a main function

**DLL**:

* Dynamic link library is used as part of EXE or other DLL’s
* It runs in application process memory,
* It can be reused in application
* It does not have a main function

## A strong name

A strong name consists of the assembly's identity—its simple text name, version number, and culture information (if provided)—plus a public key and a digital signature. It is generated from an assembly file using the corresponding private key. (The assembly file contains the assembly manifest, which contains the names and hashes of all the files that make up the assembly.)

# What is a Web Service?

A web service is a web-based functionality accessed using the protocols of the web to be used by the web applications and uses standard XML Messaging for communication.

XML is used to encode all communications to a Web service. For example, a client invokes a Web service by sending an XML message, then waits for a corresponding XML response. Because all communication is in XML, Web services are not tied to any one operating system or programming.

## What is the Serialization used for WebService?

Web Services uses XML Serialization.

## What is WSDL?

WSDL is an acronym for Web Services Description Language. It is a format to describe what a Web Service is going to offer, generally Operations, Definition and Service bindings.

## What is UDDI?

UDDI is an acronym for Universal Description, Discovery, and Integration. UDDI is used for locating the web service.

## What is the Web service protocol stack?

The Web service protocol stack is an evolving set of protocols used to define, discover, and implement Web services. The core protocol stack consists of four layers:

**Service Transport:** This layer is responsible for transporting messages between applications. Currently, this includes HTTP, SMTP, FTP, and newer protocols, such as Blocks Extensible Exchange Protocol (BEEP).

**XML Messaging:** This layer is responsible for encoding messages in a common XML format so that messages can be understood at either end. Currently, this includes XML-RPC and SOAP.

**Service Description:** This layer is responsible for describing the public interface to a specific Web service. Currently, service description is handled via the WSDL.

**Service Discovery:**This layer is responsible for centralizing services into a common registry, and providing easy publish/find functionality. Currently, service discovery is handled via the UDDI.

Beyond the essentials of XML-RPC, SOAP, WSDL, and UDDI, the Web service protocol stack includes a whole zoo of newer, evolving protocols. These include WSFL (Web Services Flow Language), SOAP-DSIG (SOAP Security Extensions: Digital Signature), and USML (UDDI Search Markup Language).

## What is the extension for Web Service?

.asmx

## What is the namespace for Web Service?

System.Web.Services