**===TechBlock**

# [How to keep a default page with Forms Authentication?](https://stackoverflow.com/questions/7136961/how-to-keep-a-default-page-with-forms-authentication)

Think of it this way...

**POST** - create

**PUT** - replace

**PATCH** - update

**GET** - read

**DELETE** – delete

* **POST** is always for creating a resource ( does not matter if it was duplicated )
* **PUT** is for checking if resource is exists then update , else create new resource
* **PATCH** is always for update a resource

Patch Vs. Put in Web API.

Example between PUT and PATCH

PUT

if i had to change my firstname then send put request for Update

{ "first": "Nazmul", "last": "hasan" } So, here in order to update the first name we need to send all the parameters of the data again

Example: Put Method in Web API Controller

public class StudentController : ApiController

{

public StudentController()

{

}

public IHttpActionResult Put(StudentViewModel student)

{

if (!ModelState.IsValid)

return BadRequest("Not a valid model");

using (var ctx = new SchoolDBEntities())

{

var existingStudent = ctx.Students.Where(s => s.StudentID == student.Id)

.FirstOrDefault<Student>();

if (existingStudent != null)

{

existingStudent.FirstName = student.FirstName;

existingStudent.LastName = student.LastName;

ctx.SaveChanges();

}

else

{

return NotFound();

}

}

return Ok();

}

}

PATCH:

patch request says that we would only send the data that we need to modify without modifying or effecting other parts of the data. Ex: if we need to update only the first name, we pass only the first name.

For Example.

<http://techbrij.com/http-patch-request-asp-net-webapi>

[HttpPatch("update/{id}")]

public Person Patch(int id, [FromBody]JsonPatchDocument<PersonDTO> personPatch)

{

PersonDatabase personDatabase = \_personRepository.GetById(id); // Get our original person object from the database.

PersonDTO personDTO = \_mapper.Map<PersonDTO>(personDatabase); //Use Automapper to map that to our DTO object.

personPatch.ApplyTo(personDTO); //Apply the patch to that DTO.

\_mapper.Map(personDTO, personDatabase); //Use automapper to map the DTO back ontop of the database object.

\_personRepository.Update(personDatabase); //Update our person in the database.

return personDTO;

}

[HttpPatch]

public IHttpActionResult Patch(AddressPatchRequest request)

{

var address = db.Addresses.Find(request.Id);

if (address == null)

return NotFound();

request.Patch(address);

db.SaveChanges();

return Ok(new AddressUpdateResponse(address));

}

## Data contract serializer types how it works

## NetDataContractSerializer

NetDataContractSerializer is analogous to .Net Remoting Formatters. It implements IFormatter and it is compatible with [Serializable] types. It is not recommended for service oriented design.

**Usage**

* It is used when we have WCF services at both sides.
* It is easier when we want to migrate .Net remoting applications.

## DataContractSerializer

DataContractSerializer is a default serializer for WCF. We need not mention DataContractSerializer attribute above the service contract.

**Usage**

Used by default unless we specify otherwise.

It is the best choice for code-first designs.

**Can I replace IEnuramable with IQuerable?**

Yes, it can replace IEnumerable anywhere

## What to use when?

## IEnumerable

* If data is available in the same process; i.e., in memory collection, Array, ArrayList etc.
* LINQ to Object and LINQ to XML.
* It is not suitable for paging scenarios.

**IQueryable**

* When querying data from out of process; i.e., SQL, Oracle or LDAP etc.
* Good for LINQ to SQL.
* By the virtue of the support of lazy loading, it is best for the query in paging scenarios.
* Last and most important, it can replace IEnumerable anywhere.

**Mydiv find in jquery**

<div id="triger1">some elements inside</div>

<div id="triger2">some elements inside</div>

<div id="triger3">some elements inside</div>

<div id="triger4">some elements inside</div>

$("div[id^='triger']")

This will return all <div> with id starting (^=) with triger.

$('div[class="shadow"]')

**JSON** stands for JavaScript Object Notation. JSON is a lightweight format for storing and transporting data. JSON is often used when data is sent from a server to a web page. JSON is "self-describing" and easy to understand

**Skip Specific domain hit with web.config.**

<ipSecurity allowUnlisted="true">

<clear/> <!-- removes all upstream restrictions -->

<add ipAddress="83.116.19.53"/>

</ipSecurity>

</security>

**Full outer join working Linq**

A full outer join is a logical union of a left outer join and a right outer join. LINQ does not support full outer joins directly, the same as right outer joins.

1. **using** (JoinEntities Context = **new** JoinEntities())
2. {
3. var leftOuterJoin = from e **in** Context.EmployeeMasters
4. join d **in** Context.DepartmentMasters on e.DepartmentId equals d.DepartmentId into dept
5. from department **in** dept.DefaultIfEmpty()
6. select **new**
7. {
8. EmployeeCode = e.Code,
9. EmployeeName = e.Name,
10. DepartmentName = department.Name
11. };
12. var rightOuterJoin = from d **in** Context.DepartmentMasters
13. join e **in** Context.EmployeeMasters on d.DepartmentId equals e.DepartmentId into emp
14. from employee **in** emp.DefaultIfEmpty()
15. select **new**
16. {
17. EmployeeCode = employee.Code,
18. EmployeeName = employee.Name,
19. DepartmentName = d.Name
20. };
21. leftOuterJoin = leftOuterJoin.Union(rightOuterJoin);
22. Console.WriteLine("Employee Code\tEmployee Name\tDepartment Name");
23. **foreach** (var data **in** leftOuterJoin)
24. {
25. **if**(!**string**.IsNullOrEmpty(data.EmployeeCode))
26. Console.WriteLine(data.EmployeeCode + "\t\t" + data.EmployeeName + "\t" + data.DepartmentName);
27. **else**
28. Console.WriteLine(data.EmployeeCode + "\t\t" + data.EmployeeName + "\t\t" + data.DepartmentName);
29. }
30. }

**===Syntel**

**What is Satellite assembly?**

A satellite assembly is a .NET Framework assembly containing resources specific to a given language. Using satellite assemblies, you can place resources for different languages in different assemblies, and the correct assembly is loaded into memory only if the user selects to view the application in that language.

**Calling Store Procedure in EF.**

obj\_pr\_getGraphDataCountryWise = await \_context.pr\_getGraphDataCountryWise.FromSql(

"exec pr\_getGraphDataCountryWise {0}", journalID).ToListAsync();

**Truncate with transaction**

In SQL Server, you can rollback a TRUNCATE from a transaction. It does write page deallocation to the log, as you mentioned.

If you use TRANSACTIONS in your code, TRUNCATE can be rolled back. If there is no transaction is used and TRUNCATE operation is committed, it can not be retrieved from log file. TRUNCATE is DDL operation and it is not logged in log file.

**Difference between Eager Loading and Lazy Loading?**

Which is good - Eager Loading or Lazy Loading?

Without looking at the Application architecture and what we are trying to achieve, we cannot say one is better over the other. Both have their own advantages and disadvantages. There are clear performance trade-offs between Eager and Lazy Loading objects from a database.

Eager Loading When you are sure that want to get multiple entities at a time, for example you have to show user, and user details at the same page, then you should go with eager loading. Eager loading makes single hit on database and load the related entities.

Lazy loading When you have to show users only at the page, and by clicking on users you need to show user details then you need to go with lazy loading. Lazy loading make multiple hits, to get load the related entities when you bind/iterate related entities.

You can turn off the lazy loading feature by setting LazyLoadingEnabled property of the ContextOptions on context to false. Now you can fetch the related objects with the parent object in one query itself.

**context.ContextOptions.LazyLoadingEnabled = false;**

Eager loading is achieved using the **Include()** method.

In the following example, it gets all the students from the database along with its standards using the Include() method.

LINQ Query Syntax:

using (var context = new SchoolDBEntities())

{

var stud1 = (from s in context.Students.Include("Standard")

where s.StudentName == "Bill"

select s).FirstOrDefault<Student>();

}

LINQ Method Syntax:

using (var ctx = new SchoolDBEntities())

{

var stud1 = ctx.Students

.Include("Standard")

.Where(s => s.StudentName == "Bill")

.FirstOrDefault<Student>();

}

The above LINQ queries will result in following SQL query:

SELECT TOP (1)

[Extent1].[StudentID] AS [StudentID],

[Extent1].[StudentName] AS [StudentName],

[Extent2].[StandardId] AS [StandardId],

[Extent2].[StandardName] AS [StandardName],

[Extent2].[Description] AS [Description]

FROM [dbo].[Student] AS [Extent1]

LEFT OUTER JOIN [dbo].[Standard] AS [Extent2] ON [Extent1].[StandardId] = [Extent2].[StandardId]

WHERE 'Bill' = [Extent1].[StudentName]

**Can I have Constructor in static class & can we pass parameter to static class**

As MSDN says, A static constructor is called automatically to initialize the class before the first instance is created. Therefore you can't send it any parameters.If the CLR must call a static constructor how will it know which parameters to pass it?

In C#, static constructors are parameterless, and there're few approaches to overcome this limitation. One is what I've suggested you.

public class A

{

private static string ParamA { get; set; }

public static void Init(string paramA)

{

ParamA = paramA;

}

}

**Jquery unobtrusive**

Basically, it is simply Javascript validation that doesn't pollute your source code with its own validation code. This is done by making use of data- attributes in HTML.

Which interface does web service implements

**How to Make Service RestFul in WCF.**

1. <system.serviceModel>
2. <services>
3. <service name="RestFullServiceSample.Service1">
4. <endpoint address="" binding="webHttpBinding" bindingConfiguration="" behaviorConfiguration="RFEndPointBehavior"
5. contract="RestFullServiceSample.IService1" />
6. </service>
7. </services>
8. <behaviors>
9. <endpointBehaviors>
10. <behavior name="RFEndPointBehavior"  >
11. <webHttp helpEnabled="true"/>
12. </behavior>
13. </endpointBehaviors>
14. </behaviors>

**ChannelFactory Class in WCF.**

A Channel Factory enables you to create a communication channel to the service without a proxy. A Channel Factory that creates and manages the various types of channels which are used by a client to send a message to various configured service endpoints.

A Channel Factory is implemented by the IChannelFactory Interface and their associated channels are used by the initiators of a communication pattern. The Channel Factory class is useful when you want to share a common service contract DLL between the client and the server.

Channel Factory is useful when you do not share more than just the service contract with the client. And also your service entity will not change frequently. You can also cache your channel using the static property.

**Thread Safe Concurrent Collection in C#**

The .NET framework offers some collection classes specifically used in multithreading. These collections are internally used synchronization hence we can call them thread safe collections. These collections can be accessed by multiple threads at a time hence they are called concurrent collections.

Here is the list of concurrent collections in C# 4.0

ConcurrentStack<T>

ConcurrentQueue<T>

BlockingCollection<T>

ConcurrentBag<T>

ConcurrentDictionary<TKey,T>