**MVC Concepts**

Multiple Models on a View in MVC

## Introduction

In MVC we cannot use multiple model tag on a view. But Many times we need to pass multiple models from controller to view or we want to show data from multiple model on a view. For example we want to show a Blog and List of Comments of this Blog on a Single View or suppose if we want to show a Category and List Of Products related to this category on a View. So in such situations we need to pass multiple models to a Single View . We have many solutions of this problem. So in this article we will see different approaches to Bind Multiple Models on a View.

## Problem Statement

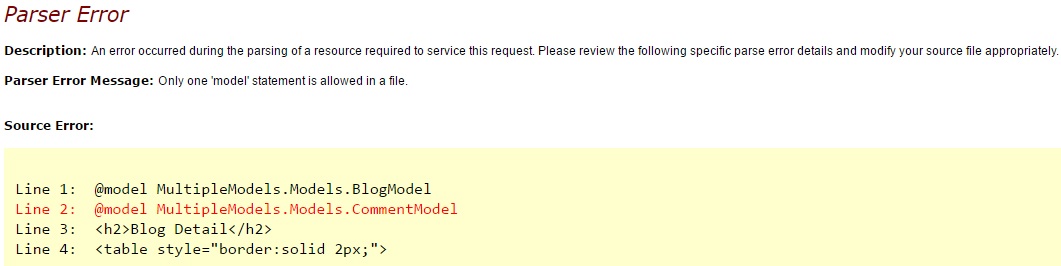
We have two models named Blog and Comment and we want to create a view where we will show Blog Title and Blog Description and List of Comments related to the Blog.

## Problem while binding two Models

If we will try to Bind Blog and Comment Model on a View as below then at run time we will get an exception that we can not Bind Multiple Models on a View.

https://www.codeproject.com/KB/aspnet/1108855/MM1.jpg

And the exception



So In MVC we need some approach through which we can achieve this and luckily i am giving 10 approaches to achieve the same.  I am not an expert, I'm sure I'll make mistakes or go against best practices several times . I will not discuss the pros and cons of different approaches. My intent is to show different approaches through which we can solve this problem.

## Different ways to achieve

we will see below 10 ways to bind multiple models on a single view  
  
1. View Model  
  
2. View Bag  
  
3. View Data  
  
4. Temp Data  
  
5. Session  
  
6. Dynamic  
  
7. Tuples  
  
8. Render Action  
  
9. JSON  
  
10. Navigation Properties

## ****Dynamic (ExpandoObject)****

The ExpandoObject class enables us to add and delete members at run time.So If we want to build up an object to work with on the fly at runtime then we can use Expando Object.  
  
I am not going in detail about ExpandoObject. let's see how can we use Expando with Multiple Models.   
  
Change the action method as below 

Hide   Copy Code

public ActionResult GetBlogComment()

{

dynamic BCVM = new ExpandoObject();

BCVM.Blog = GetBlogModel();

BCVM.Comments = GetCommentModel();

return View(BCVM);

}

**Note :**To use ExpandoObject Add System.Dynamic Namespace.   
  
Change your view as below



Run the application and we will get the expected result.

## ****Tuples****

Tuple is an ordered sequence, immutable, fixed-size and of heterogeneous objects. each object in tuple is being of a specific type.  
  
Tuples are not new in programming. They are already used in F#, Python and databases. They are new to C#. The tuples were introduced in C# 4.0 with dynamic programming.  
  
we can use Tuples also to pass multiple models to view as below 

Hide   Copy Code

public ActionResult GetBlogComment()

{

var BCVM = new Tuple<BlogModel, List<CommentModel>>(GetBlogModel(), GetCommentModel());

return View(BCVM);

}

And now change your view as below



Run the application to see the result.

## ****8. Render Action****

Render actions are special controller methods defined only to be called from view. We create Render Action Method same as we create regular Action Method.  
  
A Render action is a public method on the controller class. Render Action is called from view not from URL so we should decorate RenderAction with the [ChildActionOnly] attribute.  
  
I am creating two Action Methods from where i will return partial view Results and will render those results on view using RenderAction Methods.

Hide   Copy Code

public ActionResult GetBlogComment()

{

return View();

}

public PartialViewResult RenderBlog()

{

return PartialView(GetBlogModel());

}

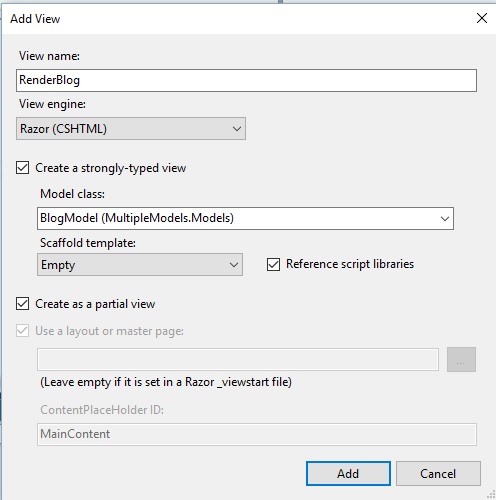
public PartialViewResult RenderComments()

{

return PartialView(GetCommentModel());

}

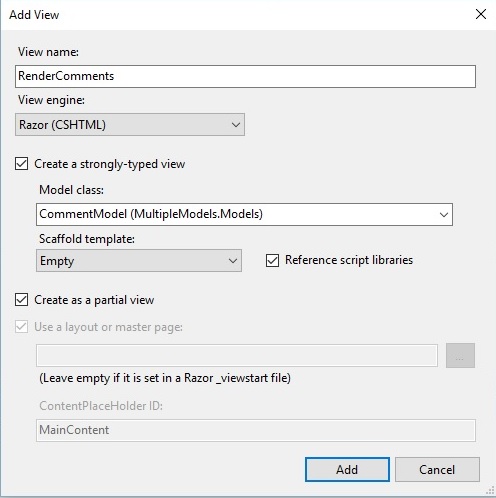
Right click on RenderBlog() and add a view as below (Make sure we need to create a partial view so check that option)



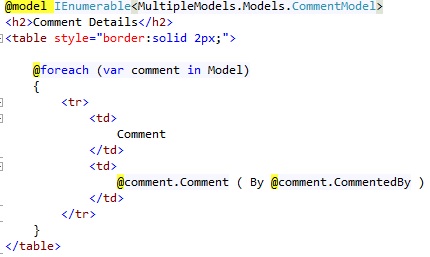
Now change the code of your created partial(RenderBlog.cshtml) view as below



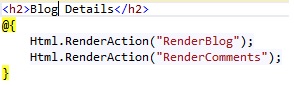
Same way right click on RenderComments and create another partial view as below



change this partial view(RenderComments.cshtml) as below



Its time to render these two partial view on our main view named GetBlogComment.cshtml.



Now run the application and we will get the required result.

## ****9. JSON****

We can Bind Multiple Models with the help of Json as well. We will retun JsonResult from action Method and on View through JQuery we can pasrse the JSON data and Bind on View. Here is the code   
  
Change your action method so that Action Method can return the JSON 

Hide   Copy Code

public ActionResult GetBlogComment()

{

return View();

}

public JsonResult RenderBlog()

{

return Json(GetBlogModel());

}

public JsonResult RenderComments()

{

return Json(GetCommentModel());

}

Change your view as below



## ****10. Navigation Properties****

If we have two related models then we can bind a model into another model as a property and can pass to a View. For example we have a Blog Model and Comment Model and A blog can contains multiple comments so we can declare a Navigation Property in Blog Model as below

Hide   Copy Code

public class BlogModel

{

public int BlogID { get; set; }

public string BlogTitle { get; set; }

public string BlogDescription { get; set; }

public DateTime CreatedDate { get; set; }

public List<CommentModel> Comments { get; set; } *//Navigation Property*

}

Now change your GetBlogModel() function and GetBlogComment() Action as below 

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

public ActionResult GetBlogComment()

{

BlogModel BM = GetBlogModel();

return View(BM);

}

public BlogModel GetBlogModel()

{

BlogModel bModel = new BlogModel()

{

BlogID = 1,

BlogTitle = "MVC Blog",

BlogDescription = "This is MVC Blog",

CreatedDate = DateTime.Now,

Comments = GetCommentModel() *//Add Comments here*

};

return bModel;

}

public List<CommentModel> GetCommentModel()

{

List<CommentModel> cModel = new List<CommentModel>();

cModel.Add(new CommentModel() { BlogID = 1, CommentID = 1, Comment = "Good One", CommentedBy = "Vijay" });

cModel.Add(new CommentModel() { BlogID = 1, CommentID = 2, Comment = "Nice", CommentedBy = "Nishant" });

cModel.Add(new CommentModel() { BlogID = 1, CommentID = 2, Comment = "Perfect", CommentedBy = "Saurabh" });

return cModel;

}

Now create your view as below



**At Last - If this is helpful and you liked this please vote above.**

[**https://www.codeproject.com/Articles/1108855/ways-to-Bind-Multiple-Models-on-a-View-in-MVC**](https://www.codeproject.com/Articles/1108855/ways-to-Bind-Multiple-Models-on-a-View-in-MVC)

Multiple Callback Functions Allowed  
We can pass more than one callback functions into the parameter of a function, just like we can pass more than one variable. Here is a classic example with jQuery’s AJAX function:

|  |  |
| --- | --- |
|  | function successCallback() { |
|  | // Do stuff before send​ |
|  | } |
|  | ​ |
|  | ​function successCallback() { |
|  | // Do stuff if success message received​ |
|  | } |
|  | ​ |
|  | ​function completeCallback() { |
|  | // Do stuff upon completion​ |
|  | } |
|  | ​ |
|  | ​function errorCallback() { |
|  | // Do stuff if error received​ |
|  | } |
|  | ​ |
|  | $.ajax({ |
|  | url:"http://fiddle.jshell.net/favicon.png", |
|  | success:successCallback, |
|  | complete:completeCallback, |
|  | error:errorCallback |
|  | ​ |
|  | }); |

# Introduction to OAuth in ASP.NET MVC

[**ashish\_\_shukla**](https://www.codeproject.com/script/Membership/View.aspx?mid=7869564), 14 Apr 2013 [CPOL](http://www.codeproject.com/info/cpol10.aspx)

|  |  |
| --- | --- |
| https://codeproject.global.ssl.fastly.net/script/Ratings/Images/stars-fill-md.png  https://codeproject.global.ssl.fastly.net/script/Ratings/Images/stars-empty-md.png | 4.86 (32 votes) |
|  | |  |  |  | | --- | --- | --- | | Rate this: |  |  | |

OAuth authorization is an open standard for authorization using third party applications

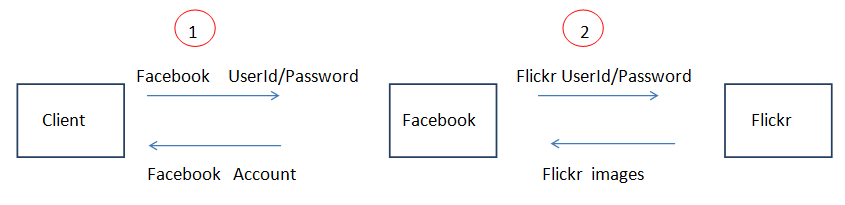
## Introduction

OAuth or Open standard for Auhtorization has become a standard which is used nowdays in most of the applications. Here we will discuss what is OAuth and how we can implement it using ASP.NET MVC.

**Background**

**What is OAuth?**

Before discussing anything else let us first understand what is OAuth. Suppose user has some resources stored on the server and there is a third party application which wants’ to access these user resources. This is a scenario which we might have encountered many times in our lives. Let’s say we have got some really cool pictures in Flickr that we want to share with our Facebook friends. We go to Facebook which redirects us to Flickr where we provide our authentication details and we are done. The great thing in all this is that we never need to share our Flickr details with Facebook.This is an example of OAuth authorization.



In the step 1 above the Client logins to his Facebook account by providing a userid and a password. If user wants to share his Flickr images with his Facebook account he selects the appropriate option in Facebook. Facebook redirects him to Flickr where he provides his credentials (step 2 above). Once the user is logged in to his Flickr account he can chose to share his Flickr images with his Facebook account.

So unlike normal authentication process as in a typical web application there is two step authentication involved here. So we can formally define OAuth as: OAuth is a protocol that allows end users to give access to third party applications to access their resources stored on a server.

We can retrieve user account information from Facebook so that we can use it in our application.We can use this information for different purposes like creating customized experience for every user depending on his personal preferences .OAuth 2.0 is the latest version of OAuth and it is not backward compatible with OAuth 1.0. Different providers use different versions of OAuth.For example Facebook and Twitter uses OAuth version 2.0 .

**Advantages of using OAuth**

Giving the third party application access to the users resources on a website has an advantage for the end user since he can easily share his already existing resources with another application instead of duplicating the resources in a new website.

Today most of the internet users have multiple accounts with different sites like Google,Microsoft, Facebook etc. .Imagine the situation when the poor user is asked to register again on another website. I am sure you might have had this feeling since registering with a new site consumes time.

Using OAuth the application can allow the user to login using his existing credentials(on another website).So user does not have to create and remember another credentials on a new web site . It has an advantage for you as a developer since you can delegate the authorization to another trusted website. These trusted websites that authorize users on other applications behalf are called **Identity providers**.

Using Open Authentication the user can give limited access to the third party applications to their resources stored on some other website. And the user never need to share his credentials with these third party  
applications.Instead of userid and password  the applications use the **access token** to fetch the users data.

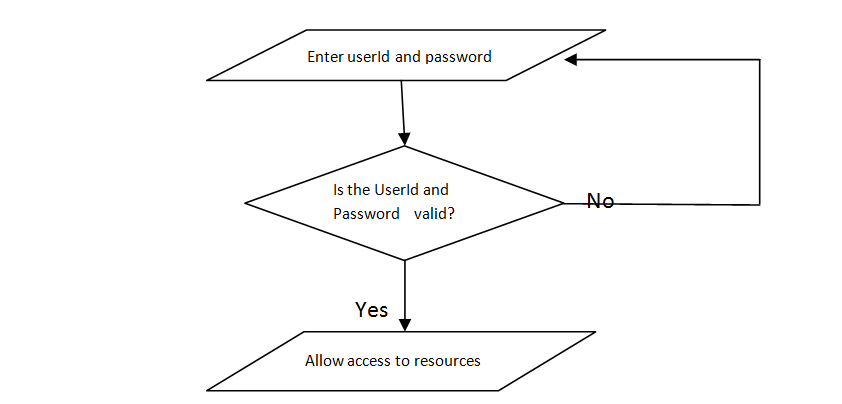
**How OAuth works**

In a normal scenario user has some resources stored on the server that he can access using his userid and password. User provides the credentials namely userid and password and is granted access to his resources. This is mandatory so that the user’s resources ,which could be images or any other documents are safe.

Here the main entities involved in this transaction are

* **Client**
* **Server**
* **User** accessing his resources stored on the server
* **Resources** of the user stored on the server that he is trying to access.

This can be depicted by the following flowchart.



## In the above flowchart user is allowed access to his resources using authorization which is a step that depends upon authentication.But the main point here is that all of the above process is mostly performed by a single application.

As the userid and password validation occurs in the same application the user is accessing, so the password of the user is stored in the database of the application most likely in an encrypted format. Since the password is encrypted user can be sure that his credentials are known only to him(ignoring the case that his account is hacked ) so it is very less likely that his credentials are misused.

The above scenario represents typical user authentication process performed by an application. In the case of applications using OAuth authentication the process works a bit differently. Instead of the user directly signing in to an application the user is rather redirected to another web site where he needs to enter his credentials.

Following steps are common no matter which provider we are using.

* Register our application with the provider and receive a key and a secret
* Once the user shows his intention to authenticate using the provider our application sends a request to the provider for a**request token**(which is just another set of credentials)
* In the final step our application asks the provider for the **access token**. Once our application receives the **access token** it has access to the users data.

Following diagram illustrates what we have discussed above.

## https://www.codeproject.com/KB/aspnet/577384/oAuth_process1.png

## In the last step the authentication provider sends back an access token to the application.It is the access token using which our application can access users data.

## Using the code

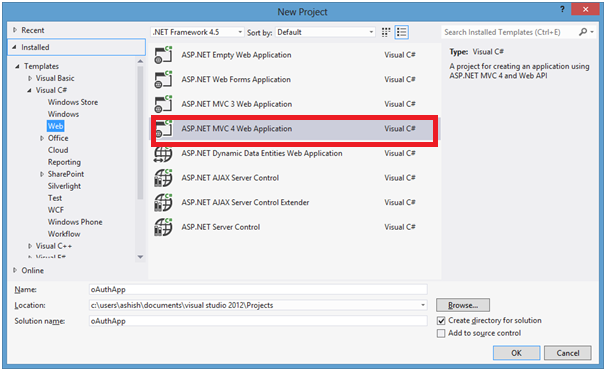
Now let us build a simple MVC application that allows the user to login using his facebook credentials. We will be implementing the application that uses Facebook as an identity provider but since the basic steps for all the providers are the same we can use any other provider as well such as Twitter with very minor changes in our application.

The above steps will be clear once we implement a simple application using OAuth.

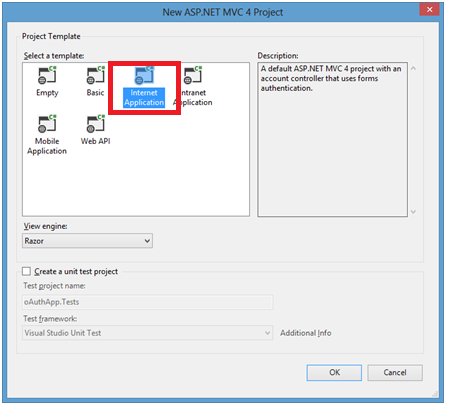
Visual studio 2012 provides OAuth support out of the box for different types of ASP.NET applications such as Web forms and MVC.

Following are the steps to create an MVC application that uses OAuth to authenticate the user using his Facebook account.

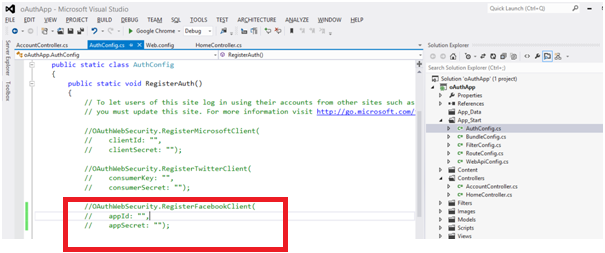
In the new project dialog select the ASP.NET MVC 4 application in the templates list.



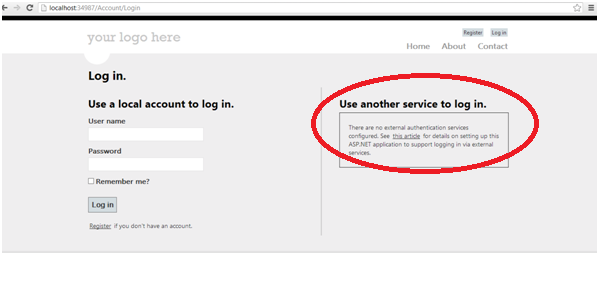
Select the Internet Application as the project template



Once you click on the create button a new project should be created and you could find an **AuthConfig.cs**file in the **App\_Start**folder which contains some commented code.



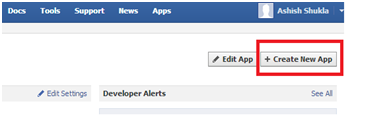
If we run the application now we will see that Under use another service to login no providers are displayed.This is because we have not yet enabled external providers yet. We will see how to do that next.



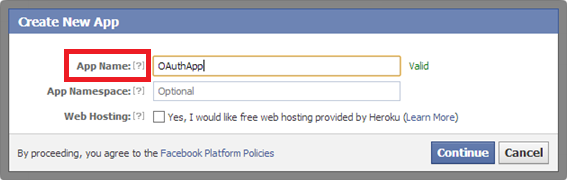
**Registering website with facebook**

Since we will be using Facebook as an identity provider we need to register our application with Facebook before we can do anything else. When we register our site with Facebook we receive id and a secret. To register our application with Facebook we browse to the url <https://developers.facebook.com/apps> and login.

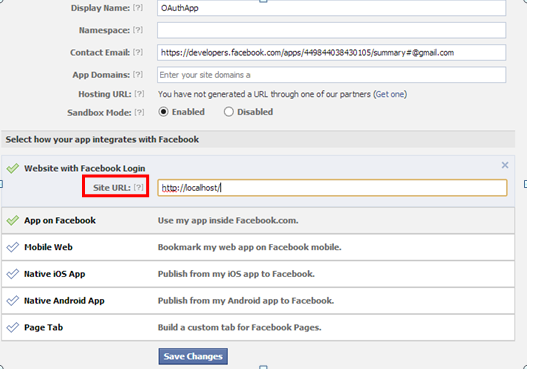
Once we login we click on Create New App in the upper right side.



In the Create New App dialog enter the name of the Application you want to create.Here we enter OAuthApp and Click on continue.



In the next window enter the application URL. You can give the URL as http://localhost/ if you are using your development URL. Also you will be able to see the AppId and App Secret(not visible below).Note the AppId and AppSecret we will need it in our application.



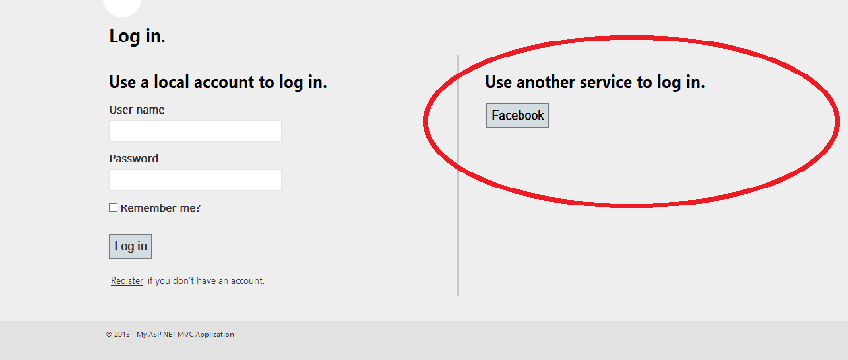
Click on save changes.

Go back to the visual studio and open **AuthConfig.cs** in the App\_Start Folder. Locate the below line in **AuthConfig.cs.**

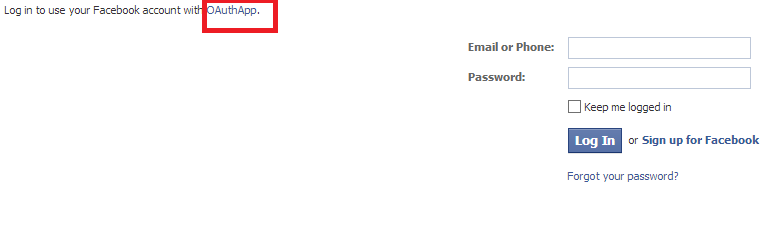
OAuthWebSecurity.RegisterFacebookClient( appId: "",appSecret: "");

Pass the appID and appSecret values to the RegisterFacebookClient method as noted above.

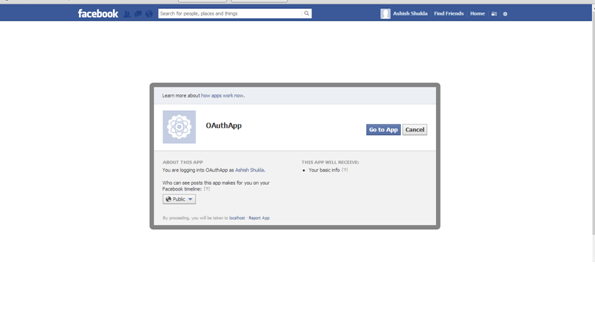
Now when we run the application and click on the login button we will see the below screen.When we put the appID and AppSecret in the **AuthConfig.cs ,**the application recognizes that we want to enable Facebook authentication and places a button for the same.



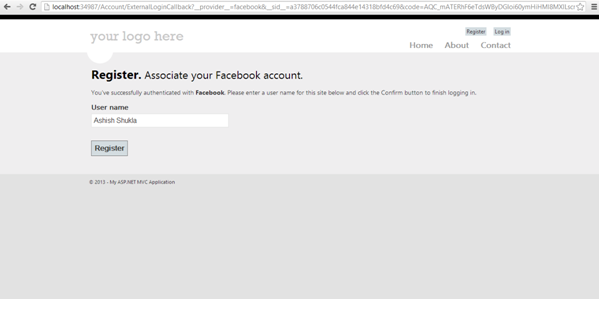
This is all we need to do for authenticating the user using facebook. Now we will go ahead and click on the Facebook button.We will be redirected to the Facebook login page.



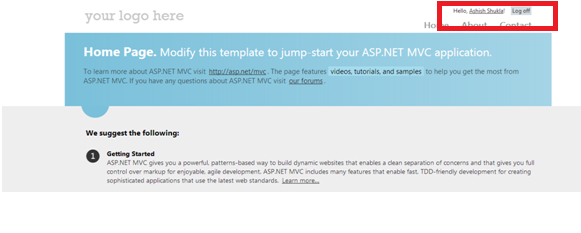
Once we login we will get an alert whether we want to login to Facebook through our Application . We click on Go to App button. This will redirect us back to our application.



We are redirected back to our application where we are asked to associate our Facebook account with a username in our application.Enter the username you want to associate with your Facebook account.



Once we click on register we are able to see our user name ,so we are logged-in now using our Facebook credentials.



Now that our Facebook authentication is working fine let’s see how we can retrieve some of the facebook details in our application. Open the AccountController.cs file in the controllers directory and go to the ExternalLoginCallback method. ExternalLogin method is called when we login to the application using external provider. ExternalLoginCallback is a callback method that is called once the provider authenticates.

The first line in the ExternalLoginCallback method is

AuthenticationResult AuthenticationResult result

= OAuthWebSecurity.VerifyAuthentication(Url.Action("ExternalLoginCallback", new { ReturnUrl =

returnUrl }));

The result variable contains a dictionary called ExtraData which contains the following keys.

* id
* name
* link
* gender
* accesstoken

Note that we need **accesstoken**to access the user account.So if we want any further details about the user we will need to use the accesstoken.

These values are Facebook specific and a different provider may return slightly different values.

**Retrieving more information from the facebook account**

Now if we want to retrieve extra data apart from the data retrieved above we can use the facebook client. Since it is available as a NuGet package we can install it as:

PM>Install-Package Facebook

Once it is installed we can very easily retrieve the detailed information about the user using the below code:

var client = new FacebookClient(HttpContext.Session["accesstoken"].ToString());

dynamic fbresult = client.Get("me");

var data = fbresult["data"].ToString();

The HttpContext.Session["accesstoken"] is the access token that is assigned to the user.Using Facebook client we are able to retrieve detailed information about the user that we can use in our application. We are able to retrieve the following details about the user.

* id
* name
* first\_name
* last\_name
* link
* username
* gender
* timezone
* locale
* verified
* updated\_time

**Points of Interest**

OAuth allows client applications to access user resource in another application.

Visual Studio 2012 ships with DotNetOpenAuth for OAuth authorization which is available in ASP.NET applications

DotNetOpenAuth is a open source library to add OpenID and OAuth capabilities to the Microsoft .NET Framework.

[**https://www.codeproject.com/Articles/577384/Introduction-to-OAuth-in-ASP-NET-MVC**](https://www.codeproject.com/Articles/577384/Introduction-to-OAuth-in-ASP-NET-MVC)

# Prevent MVC Application from Cross Site Request First step towards securing MVC applications

## Introduction

All web application platforms are potentially vulnerable to **CSRF (Cross-Site Request Forgery)** attacks. The best way to prevent this attack in MVC application is to use **Anti-Forgery token**.

Consider a banking website "www.bank.com" contains an action method DeleteUser in User Controller. When a web request comes from a client, the controller fetches the user id from session and deletes the user from database. Consider one hacker created a site "www.songs.com" and it contain one button 'Latest songs'. The button click event calls the "www.bank.com/User/DeleteAccount". A user is logged in "www.bank.com" and he is visiting "www.songs.com" using the same browser with another tab. When he clicking the 'Latest songs' button, his account will delete from the bank database. To avoid these type of unwanted requests from other sites, MVC application developers use **Anti-Forgery Token**.

Anti-Forgery Token is mainly used in form POST actions to verify the source of the POST data. In this method, for each page request, the web server sends a cookie to the client browser. While posting the data or next request time, the web server uses this cookie for client authentication. If the request is coming from an unauthorized site, the cookie will be null or invalid. By adding [ValidateAntiForgeryToken] above the controller and@Html.AntiForgeryToken() in the view page, we can prevent cross site requests forgery.

## Using the Code

The below code illustrates how **Anti-Forgery Token**Cross Site Request Forgery:

### Without Anti-Forgery Token

#### 1. Controller (Controller for deleting the user account)

Hide   Copy Code

public class UserController : Controller

{

public ActionResult DeleteUser()

{

var userId = (int)Session["userId"];

DeleteUserFromDb(userId); *//Function for deleting the user from Database*

return View();

}

}

#### 2. View (Button for deleting the user account in Bank page)

Hide   Copy Code

@using (Html.BeginForm("DeleteUser", "User"))

{

<input type="submit" value="Delete My Account" />

}

### With Anti-Forgery Token

#### 1. Controller

Hide   Copy Code

[\_\_strong\_\_][ValidateAntiForgeryToken]

public class UserController : Controller

{

public ActionResult DeleteUser()

{

var userId = (int)Session["userId"];

DeleteUserFromDb(userId);*//Function for deleting the user from Database*

return View();

}

}

#### 2. View

Hide   Copy Code

@using (Html.BeginForm("DeleteUser", "User"))

{

**@Html.AntiForgeryToken()**

<input type="submit" value="Delete My Account" />

}

### Cross Site Request Error

Hide   Copy Code

[\_\_strong\_\_]Server Error in '/' Application.

**The required anti-forgery cookie "\_\_RequestVerificationToken" is not present.**

Description: An unhandled exception occurred during the execution of the current web request.

Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.Web.Mvc.HttpAntiForgeryException:

The required anti-forgery cookie "\_\_RequestVerificationToken" is not present.

## Points of Interest

You would be wondering that I have mentioned the error and have not mentioned how to resolve that. This is a simple error that arises if you are not using the Antiforgery token attributes at appropriate places. As in, if we specify the [ValidateAntiForgeryToken] in the controller and miss out specifying in the View page posting the form, this gives rise to this exception and also prevents posting the data to the server.

## History

I came through this exception after using this valuable asset that MVC provides us with. This is a real handy attribute that is the first stage of security which an MVC developer should have in mind.

[**https://www.codeproject.com/Tips/819077/Prevent-MVC-Application-from-Cross-Site-Request-Fo**](https://www.codeproject.com/Tips/819077/Prevent-MVC-Application-from-Cross-Site-Request-Fo)

# Few Ways to Prevent Instantiation of Class

Let's explore some very interesting facts about OOP. There are a few ways prevent instantiation of a class.  
  
Let's proceed to dig into this so that my geeks can have an idea of this.  
  
Those ways are:

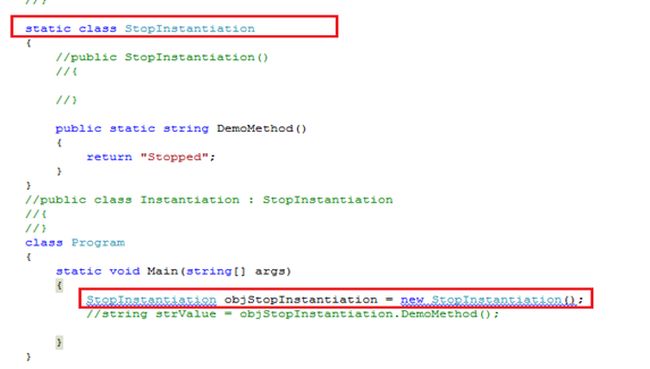
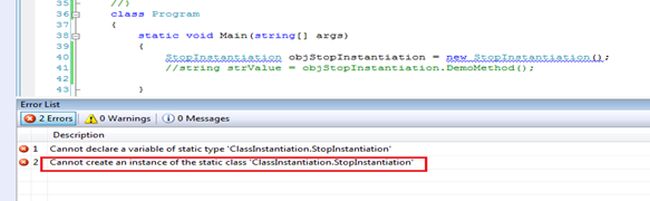
1. Abstract
2. Static Class
3. Private and protected constructor

Now jump to Abstract classes first.  
  
**1. Abstract**  
An abstract class is the one that is not used to create objects. An abstract class is designed to act as a base class (to be inherited by other classes). Some important facts about abstract are the following:

* An abstract class cannot be a sealed class or static.
* Declarations of abstract methods are only allowed in abstract classes.
* An abstract method cannot be private.
* The access modifier of the abstract method should be the same in both the abstract class and its derived class. If you declare an abstract method as protected, it should be protected in its derived class. Otherwise, the compiler will raise an error.

We declare a class as abstract to prevent us from creating an object of the class such as in the following example:  
  
  
  
It states that "Error 1 Cannot create an instance of the abstract class or interface "ClassInstantiation.StopInstantiation".  
  
Now for static classes.  
  
**2. Static Class**  
  
A class can be declared static, indicating that it contains only static members. It is not possible to create an instances of a static class using the new keyword. Static classes are loaded automatically by the .NET Framework common language runtime (CLR) when the program or namespace containing the class is loaded.  
  
Use a static class to contain methods that are not associated with a particular object. For example, it is a common requirement to create a set of methods that do not act on instance data and are not associated with a specific object in your code. You could use a static class to hold those methods.  
  
The main features of a static class are:

* They only contain static members.
* They cannot be instantiated.
* They are sealed.
* They cannot contain Instance Constructors

If you try to create an instance of a static class then it also prompts you with an error after implementation of the code as in the following:  
  
  
  
  
Whenever you try to create an instance, it prompts you with the following error:  
  
  
  
**3. Private or protected constructor**  
  
If we declare a private or protected constructor then it also prevents us from creating an instance of the class as the following code shows:  
  
  
  
These are the easiest ways to prevent class instantiation, although for some techies it may be very easy but it's also a very frequent question and most desirable question to be asked in interviews.  
  
So I thought to share in a more practical approach. It's very easy to understand, especially for those unfamiliar with this. I hope you enjoyed this demonstration.  
  
A sample application has been attached with all combinations. Kindly comment and uncomment code as needed to understand the functionality.

[**http://www.c-sharpcorner.com/UploadFile/97fc7a/few-ways-to-prevent-instantiation-of-class/**](http://www.c-sharpcorner.com/UploadFile/97fc7a/few-ways-to-prevent-instantiation-of-class/)

**Types of action results**

|  |  |  |
| --- | --- | --- |
| **ActionResult** | **Helper Method** | **Description** |
| ViewResult | View | ViewResult Renders a view as a web page. |
| PartialViewResult | PartialView | As the name describe PartialViewResult renders the partial view. |
| RedirectResult | Redirect | When you want to redirect to another action method we will use RedirectResult |
| RedirectToRouteResult | RedirectToRoute | Redirect to another action method |
| ContentResult | Content | Returns a user-defined content type |
| JsonResult | Json | When you want to return a serialized JSON object |
| JavaScriptResult | JavaScript | Returns a script that can be executed on the client |
| FileResult | File | Returns a binary output to write to the response |
| EmptyResult | (None) | returns a null result |

[**https://www.eduonix.com/blog/web-programming-tutorials/learn-different-types-of-action-results-in-mvc/**](https://www.eduonix.com/blog/web-programming-tutorials/learn-different-types-of-action-results-in-mvc/)

# Dependency Injection Techniques Explained - Using Unity Container

In the [previous article](http://www.c-sharpcorner.com/UploadFile/akkiraju/dependency-injection-techniques-explained-implementing-str/), we discussed how to use StructureMap to inject dependencies at runtime.   
  
**A brief introduction to Unity Container**  
  
The Unity Application Block (Unity) is a lightweight extensible dependency injection container with support for constructor, property, and method call injection.  
  
**Implementing the Dependency Injection with Unity Container**  
  
**According to MSDN**  
  
The Unity Container (Unity) is a lightweight, extensible dependency injection container. It facilitates building loosely coupled applications and provides developers with the following advantages:

* Simplified object creation, especially for hierarchical object structures and dependencies
* Abstraction of requirements; this allows developers to specify dependencies at run time or in configuration and simplify

**Management of crosscutting concerns**

* Increased flexibility by deferring component configuration to the container
* Service location capability; this allows clients to store or cache the container
* Instance and type interception

You need to add the reference to the Microsoft.Practices.Unity to work with Unity container.  
  
Note: Here are we are dealing with a simple implementation of Unity Container. You can use Unity Container for maintaining the object's lifetime by specifying that the object will work as a singleton that serves all the requests or per call and many other things. You are advised to go through MSDN for more details.  
  
**Setter Injector using Unity**  
  
So, here is my code to implement the Unity Container:  
  
static void Main(string[] args)  
{  
       IClassA classA = new ClassA();//creating instance of class A   
       IUnityContainer unityContainer = new UnityContainer();  
       unityContainer.RegisterType<IClassB, ClassB>();   
       var classB = unityContainer.Resolve<IClassB>();  
   
       //assign the instance to the classB property of class A   
       classA.ClassB = classB;   
  
      //Call the method  
      classA.DoSomethingFromClassB();   
      Console.Read();   
}

**First we are creating an instance of Unity Container**

IUnityContainer unityContainer = new UnityContainer();  
  
**Then we are registering the interfaces and their concrete classes**

unityContainer.RegisterType<IClassB, ClassB>();

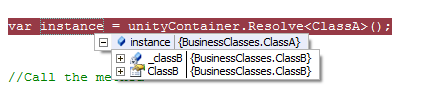
**Now asking the Container to resolve to get the actual instance of the concrete class**

var classB = unityContainer.Resolve<IClassB>();

But the Unity Container can intelligently identify what dependency object should be created if you specify the attribute [Dependency] in the consuming class setter methods.  
  
**You can see it in practice as below**

I am modifying the code to test it.  
  
    public class ClassA : IClassA  
    {  
        private IClassB \_classB;  
        [Dependency]        public IClassB ClassB  
        {  
            get { return \_classB; }              
            set  
            {  
                if (value == null) throw new ArgumentNullException("value");  
                \_classB = value;  
            }  
        }   
        public void DoSomethingFromClassB()  
        {  
            \_classB.DoSomething();  
        }  
    }

Now my Program class looks as in the following:  
  
static void Main(string[] args)  
{  
      IUnityContainer unityContainer = new UnityContainer();  
      unityContainer.RegisterType<IClassB, ClassB>();   
      IClassA classA = new ClassA();//creating instance of class A   
      var instance = unityContainer.Resolve<ClassA>();            
     //Call the method  
     instance.DoSomethingFromClassB();    
     Console.Read();   
}

If you observe in the preceding code I specified the attribute [Dependency] above the public propery of the class A and in the Main method, I have created the instance of Class A and it is asking Unity to resolve all its dependencies so that I do not need to explicitly create the dependent objects.  
  
var instance = unityContainer.Resolve<ClassA>();  
  
  
  
**Constructor Injector using Unity**  
To implement this, I have changed my interface as below:  
  
    public interface IClassA  
    {  
        //IClassB ClassB { get; set; }  
        void DoSomethingFromClassB();  
    }  
  
Now the classA looks as in the following:  
  
    public class ClassA : IClassA  
    {  
        private readonly IClassB \_classB;   
        [InjectionConstructor]  
        public ClassA(IClassB classB)  
        {  
            \_classB = classB;  
        }   
        public void DoSomethingFromClassB()  
        {  
            \_classB.DoSomething();  
        }  
    }

Observe the attribute [InjectionConstructor]. This tells the Unity that the ClassA is dependent on an object which is to be injected in a constructor.  
  
**Now the Main program looks as in the following**  
static void Main(string[] args)  
{  
     IUnityContainer unityContainer = new UnityContainer();  
     unityContainer.RegisterType<IClassB, ClassB>();  
     var instance = unityContainer.Resolve<ClassA>();  
     instance.DoSomethingFromClassB();  
     Console.Read();  
}

Here I am just specifying that we need to create an instance of ClassA by resolving all its dependencies i.e. dependent objects should be resolved by the Unity Container.

var instance = unityContainer.Resolve<ClassA>();  
 **Method Injector using Unity Container**  
  
In this, we are going to inject the dependency using a method in a class. In our example ClassA has a method that takes IClassB type as a parameter. Using a Unity Container we are going to inject the dependency into the class ClassA. In the classA, we are going to decorate the method with the attribute [InjectionMethod].  
  
    public interface IClassA  
    {  
        void DoSomethingFromClassB();  
        void SetClassB(IClassB classB);  
    }   
    public class ClassA : IClassA  
    {  
        private  IClassB \_classB;   
        [InjectionMethod]  
        public void SetClassB(IClassB classB)  
        {  
            \_classB = classB;  
        }   
        public void DoSomethingFromClassB()  
        {  
            \_classB.DoSomething();  
        }  
    }  
    static void Main(string[] args)  
    {   
          IUnityContainer unityContainer = new UnityContainer();  
          unityContainer.RegisterType<IClassB, ClassB>();  
          var instance = unityContainer.Resolve<ClassA>();  
          instance.DoSomethingFromClassB();  
          Console.Read();    
    }

In the preceding example, we are registering the classes that Unity is supposed to resolve dynamically by reading the attributes of the class. Here, the unity container without explicitly specifying the details of ClassA reads all the attributes in ClassA and resolves all the dependencies in the ClassA by instantiating the dependent objects and injecting them.  
  
In the next article, we will be discussing how to implement it using the Ninject Container.

[**http://www.c-sharpcorner.com/uploadfile/akkiraju/dependency-injection-techniques-explained-using-structurem/**](http://www.c-sharpcorner.com/uploadfile/akkiraju/dependency-injection-techniques-explained-using-structurem/)