Published on: 21 Mar, 2018

ASP.Net MVC Identity without Entity Framework

Posted by **andy** |

In this tutorial you will learn how to create your own custom identity authentication and authorization with ASP.Net MVC without using Entity Framework. By default, the example given in the MVC official tutorial site is using Entity Framework. So if you do not want to use Entity Framework and want to use external data source or your own database, you may want to read the following article I wrote on how to integrate your existing login details. If you are completely new to this, I would suggest you read their official MVC site and try their sample and examine their code by debugging the sample site so you can know at least the basic flow on how it works.

The following code I wrote will partially base on the sample code of the official site

Related Articles

How to enable attribute routing in C# MVC?

ASP.NET MVC 01 MAR, 2018

If you want to enable routing in C# MVC, you have to do the following steps. Note: this only applies to **MVC version 5 or above**. Open your RouteConfig.cs file under App_Start folder in your MVC root project.

How to download a file using .Net Web API MVC?

ASP.NET MVC 27 OCT, 2017

In this article, you will learn how to download a file in Web API MVC. We will use HTTP Get API method to perform this download action with one single parameter which will accept the file name.

PayPal Express Checkout using C# MVC Web API

ASP.NET MVC 28 OCT, 2016

In this tutorial you will learn how easily you can implement a simple checkout express using C# MVC Web API. We will create a really simple shopping cart where customers can

gave. Here is the quick link provided by Microsoft about the security overview of authentication and authorization.


```
https://docs.microsoft.com/en-us/aspnet/mvc/overview/security/
```

Lets get started with setup the database first. We will use SQL Server in this example. If you want to connect to the different database like MySQL or MongoDB, feel free to do so, as the concept will be pretty similar. What you need to do is to create the required tables and stored procedures or inline SQL to execute the query.

Connect to your SQL Management Studio and create a database named **CustomMVCIdentity**. Feel free to change the database name if needed. We are going to create 3 tables which will be:

- ## Users Table

  The Users table will store the login information. It will contains the following fields: Id, UserName, Email, Password, Status, CreatedOnDate. The status field will represent the status of an user account which can be Pending, Active, Banned, or Closed. We will create enum C# for this in the later section.

- ## Roles table

  The Roles table will store the available roles on your site. It will contains only two fields which are Id and Name only

add and delete their cart items before proceed to payment.

Multiple actions were found that match the request in MVC Web API C# ASP.Net

ASP.NET MVC 17 MAY, 2016

If you are completely new to MVC Web API and encounter a problem with an error message saying **Multiple actions were found that match the request**. It means you need to modify your Web API register class to provide additional action request in the code.

How to trigger a download file in c# MVC controller?

ASP.NET MVC 21 FEB, 2016

Learn how easily you can trigger a popup download file in c# MVC Controller.

two fields which are Id and Name only.

- ## UserRoles table

  The UserRoles table will store the relation data between Users and Roles table, therefore it will contains the following fields: UserRoleID, UserID, and RoleID.

I have created SQL scripts to create the database tables. Copy the code and execute SQL Query under the **CustomMVCIdentity** database.

```
CREATE TABLE [dbo].[Users](
        [Id] [nvarchar](50) NOT NULL,
        [Username] [nvarchar](50) NOT NULL,
        [Email] [nvarchar](50) NOT NULL,
        [Password] [nvarchar](50) NOT NULL,
        [Status] [int] NOT NULL DEFAULT(0),
        [CreatedOnDate] [datetime] NULL,
 CONSTRAINT [PK_Users] PRIMARY KEY CLUSTERED
(
        [Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPU
TE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LO
CKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMAR
Y]
) ON [PRIMARY]

GO

ALTER TABLE [dbo].[Users] ADD  CONSTRAINT [DF
_Users_CreatedOnDate]  DEFAULT (getdate()) FO
R [CreatedOnDate]
GO

CREATE TABLE [dbo].[Roles](
        [Id] [nvarchar](50) NOT NULL,
        [Name] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_Roles] PRIMARY KEY CLUSTERED
(
        [Id] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPU
TE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LO
CKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMAR
Y]
```

```
) ON [PRIMARY]

GO

CREATE TABLE [dbo].[UserRoles](
        [UserRoleID] [nvarchar](50) NOT NULL,
        [UserID] [nvarchar](50) NOT NULL,
        [RoleID] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_UserRoles] PRIMARY KEY CLUSTE
RED
(
        [UserRoleID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPU
TE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LO
CKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMAR
Y]
) ON [PRIMARY]

GO
```

The next part is we need to create a list of required stored procedures. There are a couple of optionally stored procedures that are not going to be covered in this example, but I will include in here anyway for future reference. Here is the list of stored procedures we need.

- ## NewUser

  This stored procedure will create a new user.

- ## DeleteUser

  This stored procedure will delete an existing user.

- ## GetUser

  This stored procedure will get an existing user by field ID.

- ## UpdateUser

- ## UpdateUser

  This stored procedure will update an existing user.

- ## GetUserByUserName

  This stored procedure will get an existing user by field UserName.

- ## NewUserRole

  This stored procedure will create a relation data between an user and a role.

- ## RemoveUserRole

  This stored procedure will delete a relation data between an user and a role.

- ## GetUserRoles

  This stored procedure will get user roles based on field User ID.

Here is the script version of SQL. Please execute this query under the **CustomMVCIdentity** database.

```
CREATE PROCEDURE [dbo].[NewUser]
        -- Add the parameters for the stored
procedure here
        @ID nvarchar(50),
        @UserName nvarchar(50),
        @Email nvarchar(50),
        @Password nvarchar(50),
        @Status int
AS
BEGIN
        -- SET NOCOUNT ON added to prevent ex
```

```
tra result sets from
        -- interfering with SELECT statement
s.
        SET NOCOUNT ON;

    -- Insert statements for procedure here
        INSERT INTO Users(
                ID,
                UserName ,
                Email ,
                Password ,
                Status
        )VALUES(
                @ID,
                @UserName ,
                @Email ,
                @Password ,
                @Status
        )
END
GO

CREATE PROCEDURE [dbo].[DeleteUser]
        -- Add the parameters for the stored
procedure here
        @ID nvarchar(50)
AS
BEGIN
        -- SET NOCOUNT ON added to prevent ex
tra result sets from
        -- interfering with SELECT statement
s.
        SET NOCOUNT ON;

    -- Insert statements for procedure here
        DELETE FROM Users
        WHERE ID = @ID
END
GO

CREATE PROCEDURE [dbo].[GetUser]
        -- Add the parameters for the stored
procedure here
        @ID nvarchar(50)
AS
BEGIN
        -- SET NOCOUNT ON added to prevent ex
tra result sets from
        -- interfering with SELECT statement
s.
        SET NOCOUNT ON;

    -- Insert statements for procedure here
```

```
    -- Insert statements for procedure here
        SELECT * FROM Users
        WHERE ID = @ID
END
GO

CREATE PROCEDURE [dbo].[GetUserByUsername]
        -- Add the parameters for the stored
procedure here
        @Username nvarchar(50)
AS
BEGIN
        -- SET NOCOUNT ON added to prevent ex
tra result sets from
        -- interfering with SELECT statement
s.
        SET NOCOUNT ON;

    -- Insert statements for procedure here
        SELECT * FROM Users
        WHERE Username = @Username
END
GO


CREATE PROCEDURE [dbo].[UpdateUser]
        -- Add the parameters for the stored
procedure here
        @UserName nvarchar(50),
        @Email nvarchar(50)
AS
BEGIN
        -- SET NOCOUNT ON added to prevent ex
tra result sets from
        -- interfering with SELECT statement
s.
        SET NOCOUNT ON;

    -- Insert statements for procedure here
        UPDATE Users
        SET Email = @Email
        WHERE UserName = @UserName
END
GO

CREATE PROCEDURE [dbo].[NewUserRole]
        @UserID nvarchar(50),
        @RoleName nvarchar(50)
AS
BEGIN
        DECLARE @UserRoleID nvarchar(50)
        DECLARE @RoleID nvarchar(50)
```

```
        SELECT @RoleID = Id
        FROM Roles
        WHERE Name = @RoleName

        IF @RoleID IS NULL
                BEGIN
                        INSERT INTO Roles(
                                Id,
                                Name
                        )VALUES(
                                NEWID(),
                                @RoleName
                        )

                        SELECT @RoleID = Id
                        FROM Roles
                        WHERE Name = @RoleNam
e
                END

        SELECT @UserRoleID = UserRoleID
        FROM UserRoles
        WHERE UserID = @UserID AND RoleID = @
RoleID

        IF @UserRoleID IS NULL
                BEGIN
                        INSERT INTO UserRoles
(
                                UserRoleID,
                                UserID,
                                RoleID
                        )VALUES(
                                NEWID(),
                                @UserID,
                                @RoleID
                        )
                END

END
GO

CREATE PROCEDURE [dbo].[RemoveUserRole]
        @UserID nvarchar(50),
        @RoleName nvarchar(50)
AS
BEGIN
        DECLARE @RoleID nvarchar(50)

        SELECT @RoleID = Id
        FROM Roles
        WHERE Name = @RoleName
```

```
        IF @RoleID IS NULL
                BEGIN
                        Delete FROM UserRoles
                        WHERE RoleID = @RoleI
D AND UserID = @UserID
                END

END
GO

CREATE PROCEDURE [dbo].[GetUserRoles]
        @UserID nvarchar(50)
AS
BEGIN

        SELECT R.Name As RoleName
        FROM UserRoles UR
        INNER JOIN Roles R
        ON UR.RoleID = R.Id
        WHERE UR.UserID = @UserID

END
GO
```

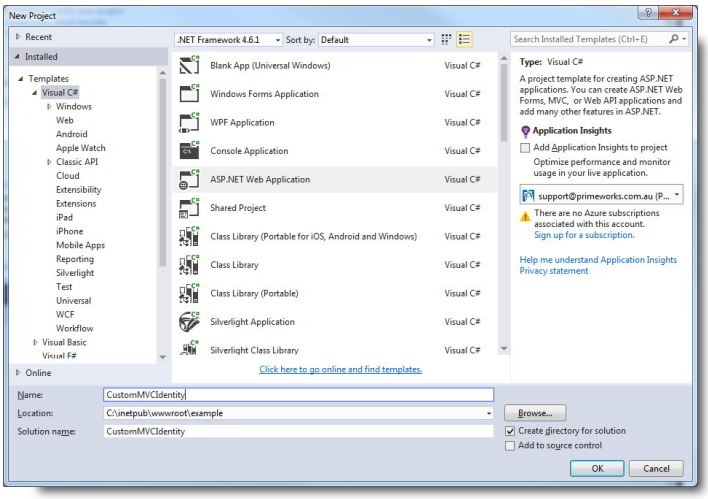
For this example, we are going to populate sample data for roles table and users table. Please run the following query under the **CustomMVCIdentity** database.

```
DECLARE @RoleIDAdmin nvarchar(50),
        @RoleIDMember nvarchar(50),
        @UserIDAdmin nvarchar(50),
        @UserIDMember nvarchar(50)

SET @RoleIDAdmin = NewID()
SET @RoleIDMember = NewID()
SET @UserIDAdmin = NewID()
SET @UserIDMember = NewID()

INSERT INTO Roles(
        ID,
        Name
)VALUES(
        @RoleIDAdmin,
        'Administrator'
)

INSERT INTO Roles(
        ID,
```

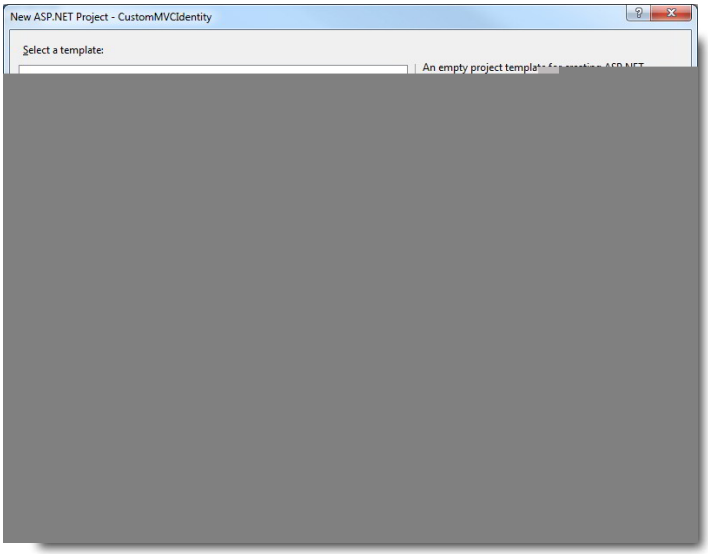
```
        Name
)VALUES(
        @RoleIDMember,
        'Member'
)

INSERT INTO Users(
        ID,
        UserName ,
        Email ,
        Password ,
        Status
)VALUES(
        @UserIDAdmin,'admin', 'admin@example.
com', '1234567', 1
)

INSERT INTO Users(
        ID,
        UserName ,
        Email ,
        Password ,
        Status
)VALUES(
        @UserIDMember,'member', 'member@examp
le.com', '1234567', 1
)

INSERT INTO UserRoles(
        UserRoleID,
        UserID,
        RoleID
)VALUES(
        NewID(), @UserIDAdmin, @RoleIDAdmin
)

INSERT INTO UserRoles(
        UserRoleID,
        UserID,
        RoleID
)VALUES(
        NewID(), @UserIDMember, @RoleIDMember
)
GO
```

To make your life easier, I will include an sql file named **SqlScript.sql**, so you can just run the SQL script in one go. I will place this file under a folder name DBScript.

Let's get started with setuping the project now. Open your Visual Studio program and create a new project. I am using Visual Studio 2015 for this example. Choose the ASP.Net Web Application as the project type and I will name the project name as **CustomMVCIdentity**.



In the template option, select **Empty template** and tick MVC checkbox option.



Once the template has already been choosen, our project structure will look like below.

We are going to add a new project that will be used as Identity User management

where it will perform all the business logic we need to query the database. Note: this is optional, you can combine all the codes or classes into one project if you do not like this way. I prefer this way because it will be easier for me to manage. Right click of your project solution and add a new project.

Select the Class library as the project type.

Our new project structure will look like below. Please delete the Class1.cs for now.

Please create the following folders and class files to match the following project structure.

Right of the project reference and Add System.Configuration reference. We require this reference because we want to get access into web.config to retrieve the database connection name.

Later on, in our main project web.config file, we need to setup a database connection with connection name **DBConnectionString** to match with the following code.

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace IdentityManagement.Utilities
{
    public class Utils
    {
        public static String ConnectionString
()
        {
            return System.Configuration.Confi
gurationManager.ConnectionStrings["DBConnecti
onString"].ConnectionString;
        }
    }
}
```

To access and query the SQL stored procedure, we will use external third party called Dapper. Dapper library will be used as ORM mapping for all our database table and entity classes. To add Dapper library into our project, please go to Tools Menu > NuGet Package > Package Console Manager.

In the project default, please select IdentityManagement project. This is because we want to add the library into the IdentityManagement project not the main website project.

Run the following install package command

```
Install-Package Dapper -Version 1.50.4
```

Once it has been successfully installed.

Open the ParameterInfo.cs file and copy

Open the ParameterInfo.cs file and copy and paste the following code.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace IdentityManagement.Data
{
    public class ParameterInfo
    {
        public string ParameterName { get; se
t; }
        public object ParameterValue { get; s
et; }
    }
}
```

Open the SqlHelper.cs file and copy and paste the following code.

```
using Dapper;
using IdentityManagement.Utilities;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace IdentityManagement.Data
{
    public static class SqlHelper
    {
        public static T GetRecord<T>(string s
pName, List<ParameterInfo> parameters)
        {
            T objRecord = default(T);
            using (SqlConnection objConnectio
n = new SqlConnection(Utils.ConnectionString
()))
            {
                objConnection.Open();
                DynamicParameters p = new Dyn
amicParameters();
                foreach (var param in paramet
ers)
```

```
                {
                    p.Add("@" + param.Paramet
erName, param.ParameterValue);
                }

                objRecord = SqlMapper.Query<T
>(objConnection, spName, p, commandType: Comm
andType.StoredProcedure).FirstOrDefault();
                objConnection.Close();
            }
            return objRecord;
        }

        public static List<T> GetRecords<T>(s
tring spName, List<ParameterInfo> parameters)
        {
            List<T> recordList = new List<T>
();
            using (SqlConnection objConnectio
n = new SqlConnection(Utils.ConnectionString
()))
            {
                objConnection.Open();
                DynamicParameters p = new Dyn
amicParameters();
                foreach (var param in paramet
ers)
                {
                    p.Add("@" + param.Paramet
erName, param.ParameterValue);
                }

                recordList = SqlMapper.Query<
T>(objConnection, spName, p, commandType: Com
mandType.StoredProcedure).ToList();
                objConnection.Close();
            }
            return recordList;
        }

        public static int GetIntRecord<T>(str
ing spName, List<ParameterInfo> parameters)
        {
            int intRecord = 0;
            using (SqlConnection objConnectio
n = new SqlConnection(Utils.ConnectionString
()))
            {
                objConnection.Open();
                DynamicParameters p = new Dyn
amicParameters();
                foreach (var param in paramet
ers)
```

```
ers)
                {
                    p.Add("@" + param.Paramet
erName, param.ParameterValue);
                }

                using (var reader = SqlMappe
r.ExecuteReader(objConnection, spName, p, com
mandType: CommandType.StoredProcedure))
                {
                    if (reader != null && rea
der.Read())
                    {
                        intRecord = Convert.T
oInt32(reader[0].ToString());
                    }
                }
                objConnection.Close();
            }
            return intRecord;
        }

        public static int ExecuteQuery(string
spName, List<ParameterInfo> parameters)
        {
            int success = 0;
            using (SqlConnection objConnectio
n = new SqlConnection(Utils.ConnectionString
()))
            {
                objConnection.Open();
                DynamicParameters p = new Dyn
amicParameters();
                foreach (var param in paramet
ers)
                {
                    p.Add("@" + param.Paramet
erName, param.ParameterValue);
                }
                success = SqlMapper.Execute(o
bjConnection, spName, p, commandType: Command
Type.StoredProcedure);
                objConnection.Close();
            }
            return success;
        }

        public static int ExecuteQueryWithInt
OutputParam(string spName, List<ParameterInfo
> parameters)
        {
            int success = 0;
            using (SqlConnection objConnectio
```

```
n = new SqlConnection(Utils.ConnectionString
()))
            {
                objConnection.Open();
                DynamicParameters p = new Dyn
amicParameters();
                foreach (var param in paramet
ers)
                {
                    p.Add("@" + param.Paramet
erName, param.ParameterValue);
                }
                success = SqlMapper.Execute(o
bjConnection, spName, p, commandType: Command
Type.StoredProcedure);
                objConnection.Close();
            }
            return success;
        }
    }
}
```

We are going to add another reference for our IdentityManagement project, which will be the component model data annotations. This reference will be used in as a login model in our login page.

Once the reference has been added, we can modify the LoginInfo entity class file. Open the file and copy and paste the following code.

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace IdentityManagement.Entities
{
    public class LoginInfo
    {
        [Required]
        [Display(Name = "UserName")]
```

```
        [Display(Name = "UserName")]
        public string UserName { get; set; }

        [Required]
        [DataType(DataType.Password)]
        [Display(Name = "Password")]
        public string Password { get; set; }
    }
}
```

In order to get access to Microsoft ASP.Net Identity, we are required to install the Microsoft.AspNet.Identity.Core package. Open your Package Manager Console and run the following code.

```
Install-Package Microsoft.AspNet.Identity.Cor
e -Version 2.2.1
```

Once the package has already been installed, open the RoleInfo.cs file and copy and paste the following code.

```
using Microsoft.AspNet.Identity;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace IdentityManagement.Entities
{
    public class RoleInfo : IRole
    {
        public string Id { get; set; }
        public string Name { get; set; }
    }
}
```

Open the Enums.cs file and copy and paste the following code.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;

namespace IdentityManagement.Utilities
{
    public enum EnumUserStatus
    {
        Pending = 0,
        Active,
        LockedOut,
        Closed,
        Banned
    }
}
```

Open the UserInfo.cs file and copy and paste the following code.

```
using IdentityManagement.Utilities;
using Microsoft.AspNet.Identity;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace IdentityManagement.Entities
{
    public class UserInfo : IUser<string>
    {
        public string Id { get; set; }
        public string UserName { get; set; }
        public string Email { get; set; }
        public string Password { get; set; }
        public EnumUserStatus Status { get; s
et; }
    }
}
```

Open the UserRoleInfo.cs file and copy and paste the following code.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace IdentityManagement.Entities
{
```

```
    public class UserRoleInfo
    {
        public string UserRoleID { get; set;
}
        public string UserID { get; set; }
        public string RoleID { get; set; }
        public string RoleName { get; set; }
    }
}
```

Open the ApplicationUser.cs file and copy and paste the following code.

```
using Microsoft.AspNet.Identity;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Claims;
using System.Text;
using System.Threading.Tasks;

namespace IdentityManagement.Entities
{
    public class ApplicationUser : UserInfo
    {
        public async Task<ClaimsIdentity> Gen
erateUserIdentityAsync(UserManager<Applicatio
nUser> manager)
        {
            // Note the authenticationType mu
st match the one defined in CookieAuthenticat
ionOptions.AuthenticationType
            var userIdentity = await manager.
CreateIdentityAsync(this, DefaultAuthenticati
onTypes.ApplicationCookie);
            // Add custom user claims here
            return userIdentity;
        }
    }
}
```

Open the UserController.cs file and copy and paste the following code.

```
using IdentityManagement.Data;
using IdentityManagement.Entities;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
using System.Text;
using System.Threading.Tasks;

namespace IdentityManagement.DAL
{
    public static class UserController
    {
        public static int NewUser(Application
User objUser)
        {
            List<ParameterInfo> parameters =
new List<ParameterInfo>();
            parameters.Add(new ParameterInfo
() { ParameterName = "Id", ParameterValue = o
bjUser.Id });
            parameters.Add(new ParameterInfo
() { ParameterName = "UserName", ParameterVal
ue = objUser.UserName });
            parameters.Add(new ParameterInfo
() { ParameterName = "Email", ParameterValue
= objUser.Email });
            parameters.Add(new ParameterInfo
() { ParameterName = "Password", ParameterVal
ue = objUser.Password });
            parameters.Add(new ParameterInfo
() { ParameterName = "Status", ParameterValue
= objUser.Status });
            int success = SqlHelper.ExecuteQu
ery("NewUser", parameters);
            return success;
        }

        public static int DeleteUser(Applicat
ionUser objUser)
        {
            List<ParameterInfo> parameters =
new List<ParameterInfo>();
            parameters.Add(new ParameterInfo
() { ParameterName = "Id", ParameterValue = o
bjUser.Id });
            int success = SqlHelper.ExecuteQu
ery("DeleteUser", parameters);
            return success;
        }

        public static ApplicationUser GetUser
(string id)
        {
            List<ParameterInfo> parameters =
new List<ParameterInfo>();
            parameters.Add(new ParameterInfo
() { ParameterName = "Id", ParameterValue = i
d });
```

```
            ApplicationUser oUser = SqlHelpe
r.GetRecord<ApplicationUser>("GetUser", param
eters);
            return oUser;
        }

        public static ApplicationUser GetUser
ByUsername(string userName)
        {
            List<ParameterInfo> parameters =
new List<ParameterInfo>();
            parameters.Add(new ParameterInfo
() { ParameterName = "Username", ParameterVal
ue = userName });
            ApplicationUser oUser = SqlHelpe
r.GetRecord<ApplicationUser>("GetUserByUserna
me", parameters);
            return oUser;
        }

        public static int UpdateUser(Applicat
ionUser objUser)
        {
            List<ParameterInfo> parameters =
new List<ParameterInfo>();
            parameters.Add(new ParameterInfo
() { ParameterName = "Email", ParameterValue
= objUser.Email });
            int success = SqlHelper.ExecuteQu
ery("UpdateUser", parameters);
            return success;
        }
    }
}
```

Open the UserRoleController.cs file and copy and paste the following code.

```
using IdentityManagement.Data;
using IdentityManagement.Entities;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace IdentityManagement.DAL
{
    public static class UserRoleController
    {
        public static int NewUserRole(string
```

```
        public static int NewUserRole(string
userID, string roleName)
        {
            List<ParameterInfo> parameters =
new List<ParameterInfo>();
            parameters.Add(new ParameterInfo
() { ParameterName = "UserID", ParameterValue
= userID });
            parameters.Add(new ParameterInfo
() { ParameterName = "RoleName", ParameterVal
ue = roleName });
            int success = SqlHelper.ExecuteQu
ery("NewUserRole", parameters);
            return success;
        }

        public static int DeleteUserRole(stri
ng userID, string roleName)
        {
            List<ParameterInfo> parameters =
new List<ParameterInfo>();
            parameters.Add(new ParameterInfo
() { ParameterName = "UserID", ParameterValue
= userID });
            parameters.Add(new ParameterInfo
() { ParameterName = "RoleName", ParameterVal
ue = roleName });
            int success = SqlHelper.ExecuteQu
ery("DeleteUserRole", parameters);
            return success;
        }

        public static IList<string> GetUserRo
les(string userID)
        {
            List<ParameterInfo> parameters =
new List<ParameterInfo>();
            parameters.Add(new ParameterInfo
() { ParameterName = "UserID", ParameterValue
= userID });
            IList<string> roles = SqlHelper.G
etRecords<string>("GetUserRoles", parameter
s);
            return roles;
        }

        public static UserInfo GetUserByUsern
ame(string userName)
        {
            List<ParameterInfo> parameters =
new List<ParameterInfo>();
            parameters.Add(new ParameterInfo
() { ParameterName = "Username", ParameterVal
```

```
ue = userName });
            UserInfo oUser = SqlHelper.GetRec
ord<UserInfo>("GetUserByUsername", parameter
s);
            return oUser;
        }
    }
}
```

Open the UserStore.cs file and copy and paste the following code.

```
using IdentityManagement.DAL;
using IdentityManagement.Entities;
using Microsoft.AspNet.Identity;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace IdentityManagement.IdentityStore
{
    public class UserStore : IUserStore<Appli
cationUser>, IUserRoleStore<ApplicationUser>
    {
        #region IUserStore
        public Task CreateAsync(ApplicationUs
er user)
        {
            if (user != null)
            {
                return Task.Factory.StartNew
(() =>
                {
                    user.Id = Guid.NewGuid().
ToString();
                    UserController.NewUser(us
er);
                });
            }
            throw new ArgumentNullException
("user");
        }

        public Task DeleteAsync(ApplicationUs
er user)
        {
            if (user != null)
            {
                return Task.Factory.StartNew
```

```
                return Task.Factory.StartNew
(() =>
                {
                    UserController.DeleteUser
(user);
                });
            }
            throw new ArgumentNullException
("user");
        }

        public void Dispose()
        {

        }

        public Task<ApplicationUser> FindById
Async(string userId)
        {
            if (!string.IsNullOrEmpty(userI
d))
            {
                return Task.Factory.StartNew
(() =>
                {
                    return UserController.Get
User(userId);
                });
            }
            throw new ArgumentNullException
("userId");
        }

        public Task<ApplicationUser> FindByNa
meAsync(string userName)
        {
            if (!string.IsNullOrEmpty(userNam
e))
            {
                return Task.Factory.StartNew
(() =>
                {
                    return UserController.Get
UserByUsername(userName);
                });
            }
            throw new ArgumentNullException
("userName");
        }

        public ApplicationUser FindByName(str
ing userName)
        {
```

```
            if (!string.IsNullOrEmpty(userNam
e))
            {
                return UserController.GetUser
ByUsername(userName);
            }
            throw new ArgumentNullException
("userName");
        }

        public Task UpdateAsync(ApplicationUs
er user)
        {
            if (user != null)
            {
                return Task.Factory.StartNew
(() =>
                {
                    return UserController.Upd
ateUser(user);
                });
            }
            throw new ArgumentNullException
("userName");
        }
        #endregion

        #region IUserRoleStore
        public Task AddToRoleAsync(Applicatio
nUser user, string roleName)
        {
            if (user != null)
            {
                return Task.Factory.StartNew
(() =>
                {
                    UserRoleController.NewUse
rRole(user.Id, roleName);
                });
            }
            else
            {
                throw new ArgumentNullExcepti
on("user");
            }
        }

        public Task RemoveFromRoleAsync(Appli
cationUser user, string roleName)
        {
            if (user != null)
            {
```

```
                return Task.Factory.StartNew
(() =>
                {
                    UserRoleController.Delete
UserRole(user.Id, roleName);
                });
            }
            else
            {
                throw new ArgumentNullExcepti
on("user");
            }
        }

        public Task<IList<string>> GetRolesAs
ync(ApplicationUser user)
        {
            if (user != null)
            {
                return Task.Factory.StartNew
(() =>
                {
                    IList<string> roles = Use
rRoleController.GetUserRoles(user.Id);
                    return roles;
                });
            }
            else
            {
                throw new ArgumentNullExcepti
on("user");
            }
        }

        public Task<bool> IsInRoleAsync(Appli
cationUser user, string roleName)
        {
            if (user != null)
            {
                return Task.Factory.StartNew
(() =>
                {
                    IList<string> roles = Use
rRoleController.GetUserRoles(user.Id);
                    foreach (string role in r
oles)
                    {
                        if (role.ToUpper() ==
roleName.ToUpper())
                        {
                            return true;
                        }
                    }
```

```
                    }

                    return false;
                });
            }
            else
            {
                throw new ArgumentNullExcepti
on("user");
            }
        }
        #endregion

    }
}
```

Once all the above codes have been copied, please try to build the project and make sure there is no error occurred.

Let's get more details on above code we copied. In order for the custom identity authentication and authorization to work, we need to inherit minimums of two interfaces which are **IUserStore** and **IUserRoleStore**. If you do not want to use any authorization, you can remove the **IUserRoleStore** interface. Each of this interface requires an entity class of UserInfo to implement IUser interface. If we look into the snapshot of the interface it will be like below.

While for the RoleInfo entity class, it requires to implement the IRole interface.

The ApplicationUser.cs hold the important role, where it consists a method that will be used to generate the user identity. It inherits the UserInfo class. One important

inherits the UserInfo class. One important thing to remember is to have to match exactly the same the default authentication type which we use ApplicationCookie.

I have divided both of **IUserStore** and **IUserRoleStore** into two separate regions (sections), so you can exactly see what interface implementation required by both interfaces. If you see the code of each interface, you will notice that we call our Data Access Layer (DAL) controller to perform the task required. For example: to find a user by a UserName field, in the **public UserInfo FindByName(string userName)** function interface, we called a UserController function to return an object User as below:

```
return UserController.GetUserByUsername(userN
ame);
```

If you are using MySQL, you can skip the Dapper.dll installation reference and use the MySql sub routine or procedures. The idea in here is to replace the business logic of each individual interface.

The next step is to add this project reference into our main site project. To do it, right click on the reference path in main project and click Add Reference.

In the window dialog, on the left tab panes, select the project tab and tick the IdentityManagement checkbox and click

OK button to add the reference.

If it has been added into the project, You should see the dll reference in the project reference list like below.

Open your Package Manager Console and install the following packages one by one under **CustomMVCIdentity** project.

```
Install-Package Microsoft.AspNet.Identity.Core -Version 2.2.1
Install-Package Microsoft.AspNet.Identity.Owin -Version 2.2.1
Install-Package Microsoft.Owin -Version 4.0.0
Install-Package Microsoft.Owin.Security -Version 4.0.0
Install-Package Microsoft.Owin.Host.SystemWeb -Version 4.0.0
```

Once the packages have been installed, we are going to create 4 pages which are Home Page, Member Page, Admin Page and Login Page. We are going to restrict the Member Page only available for Administrator and Member role. While the Admin Page will be restricted to Administrator role only.

In the Controllers folder, right click and navigate to Add menu and choose Controller.

Choose the MVC Controller empty option and click Add button.

Name the controller as HomeController and click Add button.

Once the controller has already been added, we are going to add the View file. In the Views folder, right click and navigate to Add menu and choose View.

Enter the view name as Home and click Add button. It will automatically add all the default shared and view files including all the javascript and css files. You can manually remove them if you do not need them.

We are going to make some changes on our default page by adding a list of available menus. Open the _Layout.cshtml file and copy the following code and replace the ul nav-bar code with this one.

```
<ul class="nav navbar-nav">
        <li>@Html.ActionLink("Home", "Index",
"Home")</li>
        <li>@Html.ActionLink("Member", "Inde
x", "Member")</li>
        <li>@Html.ActionLink("Admin", "Inde
x", "Admin")</li>
        <li>@Html.ActionLink("Login", "Inde
x", "Login")</li>
        <li>@Html.ActionLink("Logout", "Logou
t", "Login")</li>
</ul>
```

Press F5 on your keyboard to run the project and you will see your homepage

like below. There are couple things we want to change in here. There are some conditions we need to apply to above menus. We only want to display the Member and Admin page, if a user has already been authenticated and authorized to access the pages. The logout link will only be visible if a user has already login and the login link will only be visible if a user has not been authenticated yet.

To make the changes, we are going to create some static function to perform those checks. We are going to use the built in **Identity** keyword to perform the check. Go to your **IdentityManagement** project and add **System.Web** reference in the assembly section. Once the System.Web reference has been added. Open the Utils.cs file under Utilities folder and copy and paste the following code.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Web;

namespace IdentityManagement.Utilities
{
    public class Utils
    {
        public static String ConnectionString
()
        {
            return System.Configuration.Confi
gurationManager.ConnectionStrings[0].Connecti
onString;
        }

        public static bool IfUserAuthenticate
```

```
        public static bool IfUserAuthenticate
d()
        {
            if (HttpContext.Current.User.Iden
tity.IsAuthenticated)
            {
                return true;
            }
            return false;
        }

        public static bool IfUserInRole(strin
g roleName)
        {
            if (IfUserAuthenticated())
            {
                if (HttpContext.Current.User.
IsInRole(roleName))
                {
                    return true;
                }
            }
            return false;
        }
    }
}
```

If you see above code, we create two extra methods which will check if a user has already been authenticated and the other method will perform a check if a role exists in a user roles. Once you have copied the code, re-build the project again. We are going to modify again the _Layout.cshtml file to apply methods we just created above.

Open the layout.cshtml file and copy the following code and replace the ul nav-bar code with this one.

```
<ul class="nav navbar-nav">
        <li>@Html.ActionLink("Home", "Index",
"Home")</li>
        @if (IdentityManagement.Utilities.Uti
ls.IfUserInRole("Member"))
        {
```

```
                <li>@Html.ActionLink("Membe
r", "Index", "Member")</li>
        }else if (IdentityManagement.Utilitie
s.Utils.IfUserInRole("Administrator"))
        {
                <li>@Html.ActionLink("Admin",
"Index", "Admin")</li>
        }

        @if (IdentityManagement.Utilities.Uti
ls.IfUserAuthenticated())
        {
                <li>@Html.ActionLink("Logou
t", "Logout", "Login")</li>
        }
        else
        {
                <li>@Html.ActionLink("Login",
"Login", "Login")</li>
        }
</ul>
```

Once the changes have been made, if we press F5 on your keyboard again to run the site, we should see the following menu changes.

The next step is to create the Controllers and Views for Admin, Member, and Login page.

Please copy the following html code for the Admin Page (Index.cshtml).

```
@{
    ViewBag.Title = "Admin Page";
}

<h2>Welcome to Administration Page.</h2>
<h3>Hi @User.Identity.Name</h3>
```

Please copy the following html code for the Member Page (Index.cshtml)

```
@{
    ViewBag.Title = "Member Page";
}

<h2>Welcome to Member Page.</h2>
<h3>Hi @User.Identity.Name</h3>
```

Please copy the following html code for the Login Page (Index.cshtml).

```
@using IdentityManagement.Entities
@model LoginInfo
@{
    ViewBag.Title = "Log in";
}

<h2>@ViewBag.Title.</h2>
<div class="row">
    <div class="col-md-12">
        <section id="loginForm">
            @using (Html.BeginForm("Index",
"Login", new { ReturnUrl = ViewBag.ReturnUrl
}, FormMethod.Post, new { @class = "form-hori
zontal", role = "form" }))
            {
                @Html.AntiForgeryToken()
                <h4>Use a local account to lo
g in.</h4>
                <hr />
                @Html.ValidationSummary(true,
"", new { @class = "text-danger" })
                <div class="form-group">
                    @Html.LabelFor(m => m.Use
rName, new { @class = "col-md-2 control-labe
l" })
                    <div class="col-md-10">
                        @Html.TextBoxFor(m =>
m.UserName, new { @class = "form-control" })
                        @Html.ValidationMessa
geFor(m => m.UserName, "", new { @class = "te
xt-danger" })
                    </div>
                </div>
                <div class="form-group">
                    @Html.LabelFor(m => m.Pas
sword, new { @class = "col-md-2 control-labe
l" })
                    <div class="col-md-10">
                        @Html.PasswordFor(m =
```

```
> m.Password, new { @class = "form-control"
})
                        @Html.ValidationMessa
geFor(m => m.Password, "", new { @class = "te
xt-danger" })
                    </div>
                </div>
                <div class="form-group">
                    <div class="col-md-offset
-2 col-md-10">
                        <input type="submit"
value="Log in" class="btn btn-default" />
                    </div>
                </div>
            }
        </section>
    </div>
</div>
```

Please copy the following code for the AdminController Page (AdminController.cs).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace CustomMVCIdentity.Controllers
{
    [Authorize(Roles = "Administrator")]
    public class AdminController : Controller
    {
        // GET: Admin
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

If you notice above code, I have included a keyword Authorize apply to whole Controller access and only Administrator role can access this Controller.

Please copy the following code for the MemberController Page

(MemberController.cs).

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace CustomMVCIdentity.Controllers
{
    [Authorize(Roles = "Member")]
    public class MemberController : Controlle
r
    {
        // GET: Member
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

The MemberController will have the Authorize keyword, the only different is the role Member applied into this Controller.

Please copy the following code for the LoginController Page (LoginController.cs).

```
using CustomMVCIdentity.App_Start;
using IdentityManagement.DAL;
using IdentityManagement.Entities;
using IdentityManagement.Utilities;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.Owin;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Principal;
using System.Threading.Tasks;
using System.Web;
using System.Web.Mvc;

namespace CustomMVCIdentity.Controllers
{
    public class LoginController : Controller
    {
        private ApplicationSignInManager _sig
nInManager;
```

```
nInManager;
        private ApplicationUserManager _userM
anager;
        public ApplicationSignInManager SignI
nManager
        {
            get
            {
                return _signInManager ?? Http
Context.GetOwinContext().Get<ApplicationSignI
nManager>();
            }
            private set
            {
                _signInManager = value;
            }
        }

        public ApplicationUserManager UserMan
ager
        {
            get
            {
                return _userManager ?? HttpCo
ntext.GetOwinContext().GetUserManager<Applica
tionUserManager>();
            }
            private set
            {
                _userManager = value;
            }
        }

        public ActionResult Logout()
        {
            SignInManager.AuthenticationManag
er.SignOut(DefaultAuthenticationTypes.Applica
tionCookie);
            return RedirectToAction("Index",
"Home");
        }

        // GET: Login
        [AllowAnonymous]
        public ActionResult Index(string retu
rnUrl)
        {
            ViewBag.ReturnUrl = returnUrl;
            return View();
        }

        [HttpPost]
        [AllowAnonymous]
```

```
        [ValidateAntiForgeryToken]
        public async Task<ActionResult> Index
(LoginInfo objLogin, string returnUrl)
        {
            if (ModelState.IsValid)
            {
                ApplicationUser oUser = await
SignInManager.UserManager.FindByNameAsync(obj
Login.UserName);
                if (oUser != null && oUser.Pa
ssword == objLogin.Password)
                {
                    switch (oUser.Status)
                    {
                        case EnumUserStatus.P
ending:
                            ModelState.AddMod
elError("", "Error: User account has not been
verified.");
                            break;
                        case EnumUserStatus.A
ctive:
                            SignInManager.Sig
nIn(oUser, false, false);
                            IList<string> rol
eList = UserRoleController.GetUserRoles(oUse
r.Id);
                            foreach(string ro
le in roleList)
                            {
                                UserManager.A
ddToRole(oUser.Id, role);
                            }

                            //if no return ur
l provided then redirect page based on role
                            if (string.IsNull
OrEmpty(returnUrl))
                            {
                                if(roleList.I
ndexOf("Administrator") >= 0)
                                {
                                    return Re
directToAction("Index", "Admin");
                                }
                                else
                                {
                                    return Re
directToAction("Index", "Member");
                                }
                            }
                            return RedirectTo
```

```
Local(returnUrl);

                        case EnumUserStatus.B
anned:
                            ModelState.AddMod
elError("", "Error: User account has been ban
ned.");
                            break;
                        case EnumUserStatus.L
ockedOut:
                            ModelState.AddMod
elError("", "Error: User account has been loc
ked out due to multiple login tries.");
                            break;
                    }
                }
                else
                {
                    ModelState.AddModelError
("", "Error: Invalid login details.");
                }
            }
            return View(objLogin);
        }

        private ActionResult RedirectToLocal
(string returnUrl)
        {
            if (Url.IsLocalUrl(returnUrl))
            {
                return Redirect(returnUrl);
            }
            return RedirectToAction("Index",
"Home");
        }
    }
}
```

The way how the login works is when a user submits his or her credentials on the site. We check against the database by calling a method to search a user by UserName field. If the record is found, then we check the password. Note: you should hash the password as this only example, I just provide the basic password only. Once the password match, we then compare the status of the requested user, only

authenticate the user if the status of the user is active. The authorization will work after we use the following code to assign a role to authenticate user.

```
UserManager.AddToRole(oUser.Id, role);
```

The user will be automatically redirected to Admin or Member page based on the role the user had.

The next step we need to do is to set the Startup class. Just under the root of the project, add a new class file named Startup.cs. Then copy the following codes:

```
using CustomMVCIdentity.App_Start;
using Microsoft.AspNet.Identity;
using Microsoft.Owin;
using Microsoft.Owin.Security.Cookies;
using Owin;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CustomMVCIdentity
{
    public class Startup
    {
        public void Configuration(IAppBuilder
app)
        {
            ConfigureAuth(app);
        }

        public void ConfigureAuth(IAppBuilder
app)
        {
            app.CreatePerOwinContext<Applicat
ionUserManager>(ApplicationUserManager.Creat
e);
            app.CreatePerOwinContext<Applicat
ionSignInManager>(ApplicationSignInManager.Cr
eate);
```

```
            app.UseCookieAuthentication(new C
ookieAuthenticationOptions
            {
                AuthenticationType = DefaultA
uthenticationTypes.ApplicationCookie,
                LoginPath = new PathString("/
Login")
            });
        }
    }
}
```

The above method will be called once the site is initially started. It will set the authentication type and login path.

Before you test the new changes we made, please make sure you have added the database connection string in your web.config. Here is the example. Please modify the data source, username, and password for the access database account. You can place this section after the appSettings section.

```
<connectionStrings>
    <add name="DBConnectionString" connection
String="Data Source=.\SQLEXPRESS;Initial Cata
log=CustomMVCIdentity;User ID=userAccount;Pas
sword=userPassword" providerName="System.Dat
a.SqlClient" />
</connectionStrings>
```

Once the connection string has been setup, you can now run the site by pressing the F5 on your keyboard. If you try to access the url path of **/Admin** or **/Member**, you will be automatically redirected to **/login** page.

This is the example of successful page if you login as Administrator account.

This is the example of successful page if you login as Member account.

Download Files

You can download the ASP.Net MVC Identity project on the following link.

Download

If you have any question, please post your questions below.

Comments

swathi

23 MAY, 2018

Thanks for your post ,Can u please provide role based registration for this identity project....

Reply

andy

25 MAY, 2018

Hi Swathi,
Not quite sure you what you mean, if you see on the example code. I have differentiated two different

roles. One for admin access and the other one for member access only.

Reply

Deepak Yadav

28 JUL, 2018

Hi,
I am new in MVC. I want to add a student Detail page after login and add permission based insert update delete, please help me for doing this

Reply

DK

17 SEP, 2020

Just put below line above that action menthod, where you want to put restriction

[Authorize(Roles = "RoleName")]

Reply

Janus

25 AUG, 2018

Great to see that not everybody worships EF.

After 20 years in the business I don't see any idea behind the usage of EF, except for avoiding SQL.
You use Dapper, I use Insight Database, same same :)

Keep up the good work.

Reply

Janus

31 AUG, 2018

Halleluja..

Somehow I have lost the url to your page, pure luck I found it again.

Anyway... why is the code not on Github?

Reply

andy

02 SEP, 2018

Hi Janus,
I have added the project files on the Github site.
Click here for the project files.

Reply

Maurício Seabra

15 SEP, 2018

I am very grateful for your article. I would like to include the claims in this project. Do you have any tip?

Reply

andy

16 SEP, 2018

Hi Mauricio,
under the code comment, if you search for wording 'Add custom user claims here', you can add the claims there.

Reply

Cory Townes

27 SEP, 2018

private ApplicationSignInManager
_signInManager;
private ApplicationUserManager
_userManager;

public ApplicationSignInManager
SignInManager

In the LoginController I keep getting errors for ApplicationSignInManager, ApplicationUserManager. It says "the type of namespace name 'ApplicationSignInManger' could not be found. Can you geive me some insight on what I am doing wrong?

Reply

andy

28 SEP, 2018

Hi Cory,
Have you followed up the tutorial above and download a copy of the project. The ApplicationUserManager class is defined under IdentityConfig.cs file. Easy way to do is to download the project, unzip it, and then rebuild the project. There should be no error. You can then use this

template and copy them to your existing project.

Reply

Mohamed Cisse

01 APR, 2019

Hi,
This is great.
Can you do the same about asp.net core web api with token generation ?
Thanks

Reply

Charl Malherbe

26 APR, 2019

Very detailed and well presented.
Thank you for sharing your efforts.

Reply

ravi

08 MAY, 2019

Is it possible to perform crud operation in repository pattern using mvc without entity

framework

Reply

andy

08 MAY, 2019

Hi Ravi,
The sample code I provided already an example of CRUD using stored procedure.

Reply

Marco

11 MAY, 2019

Hi,
Thank you for your post.

Very useful, but what if I want to save my User information all with the role information? I mean, one user can only have one role and this is stored in the User table (RoleID I think). The UserRoles table dissapear because there is no many to many relationship, but how can I fix the Identity implementation without that table? How can I get the User role to obtain the same result as your

post?

Thank you.
Marco.

Reply

DInesh kumar

22 JUL, 2019

I am also waiting this Question's answer once if you will get your answer please let me know
Marco'
Thanks.

Reply

andy

27 JUL, 2019

Hi Marco,
You can have more than one role as you needed. If you see the UserRoles database table. It contains both UserId and RoleId field. One thing you need to modify is to give an extra access filter in the authorize property. Let's says you want to allow Administrators and Members to be able access the

controller. You can use the following example.

[Authorize(Roles="Administrators, Members")]

Reply

DMC

18 JAN, 2020

Is it possible for this instruction [Authorize(Roles="Administrators, Members")] to load Roles ("Administrators", "Members") from database? So developers are not obliged to touch the code if there are some changes: if there is a new role to access to a controller or if the role "members" have to be remove for example.

Reply

andy

18 JAN, 2020

I believe you have to create your own

custom authorization.
Have a look the
following article.
https://www.codeproje
ct.com/Articles/1079552
/Custom-Roles-Based-
Access-Control-RBAC-
in-ASP-NE

It should help you.

Reply

DInesh kumar

22 JUL, 2019

Thanks allot 'andy'
Really it will very useful for me if it will run, From last 10 days i was searching but i was getting all Code With EF, but i think finally i got my correct result what i have to do.
thanks you so much.

Reply

Mic

15 AUG, 2019

Not enough credit given! Thank
you very much

Reply

Mathew

03 JAN, 2020

Hello, great article however im using VS 2019 and getting multiple "Inconsistent accessibility: parameter type 'List' is less accessible than method 'SqlHelper.GetRecord(string, List)'" in the sqlhelper.cs file.

Reply

andy

05 JAN, 2020

Hi Matt,
I havent used the VS 2019. When you converted the project to VS 2019. Does the compilation work fine?

Reply

Mathew

24 JAN, 2020

Hello Andy, I haven't converted it to VS 2019, I've created the app from start using VS 2019.

Reply

Valery

22 JAN, 2020

Hi, your guide is great.

Can you explain few things ?

1. What kind of magic string is "ASP.NET Identity" in IdentityConfig.cs
{
manager.UserTokenProvider = new DataProtectorTokenProvider(dataProtectionProvider.Create("ASP.NET Identity"));
}

2. How to register account, and how to do it with e-mail activation?

3. What is right way to store passwords ?

Reply

Alex

05 NOV, 2020

Great article, saved my life

Reply

julius atagana

30 MAY, 2021

Thanks so much,
I am New to MVC, how can I get this template to download, so I can inspect where my code went wrong. on my "IdentityConfig.cs" as this needs me to enter a "dbcontext" object
thanks

Reply

Write Comment

Name

Name

Email

Email Address

Comment

Enter your comment here

0 characters entered. Maximum characters allowed are 1000 characters.

Post Comment