

ASP. NET MVC Roles Demo

Amit Ahlawat

CSE 686 – Internet Programming

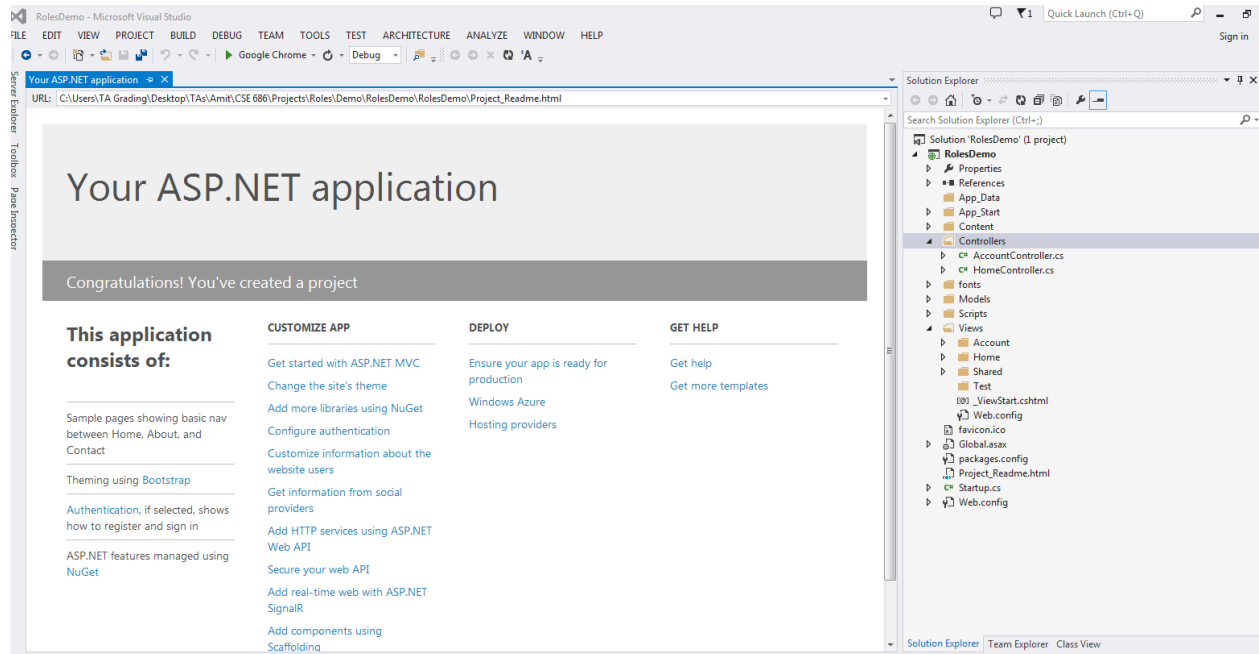
Spring 2014

Goals of document

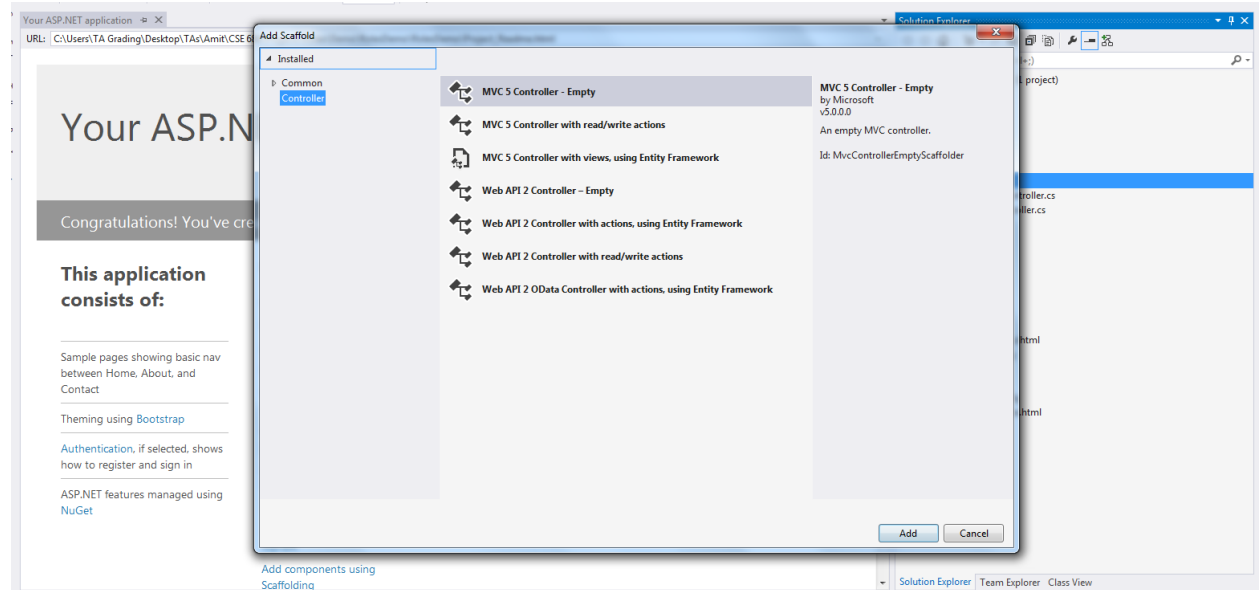
- 1) Show how to make a group of pages available only after registration with built-in authentication and registration mechanisms
- 2) Show to make specific pages available only to select users
- 3) Show to make specific pages available only to select roles

Using Built-in Registration and Authentication Mechanisms

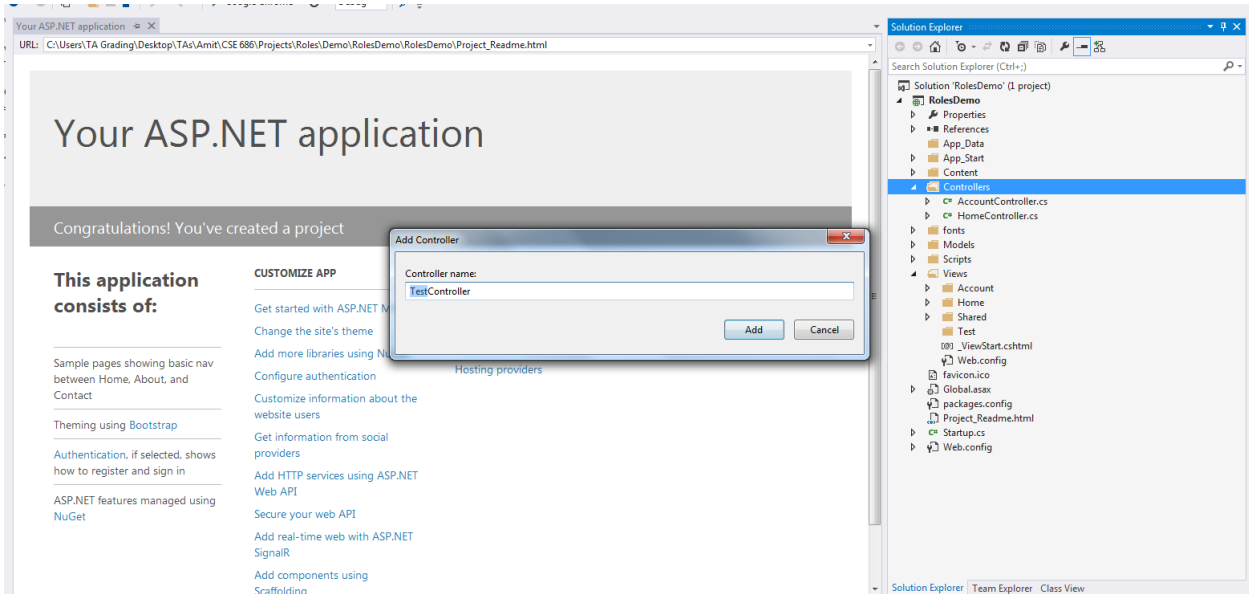
Create an ASP .NET MVC application.



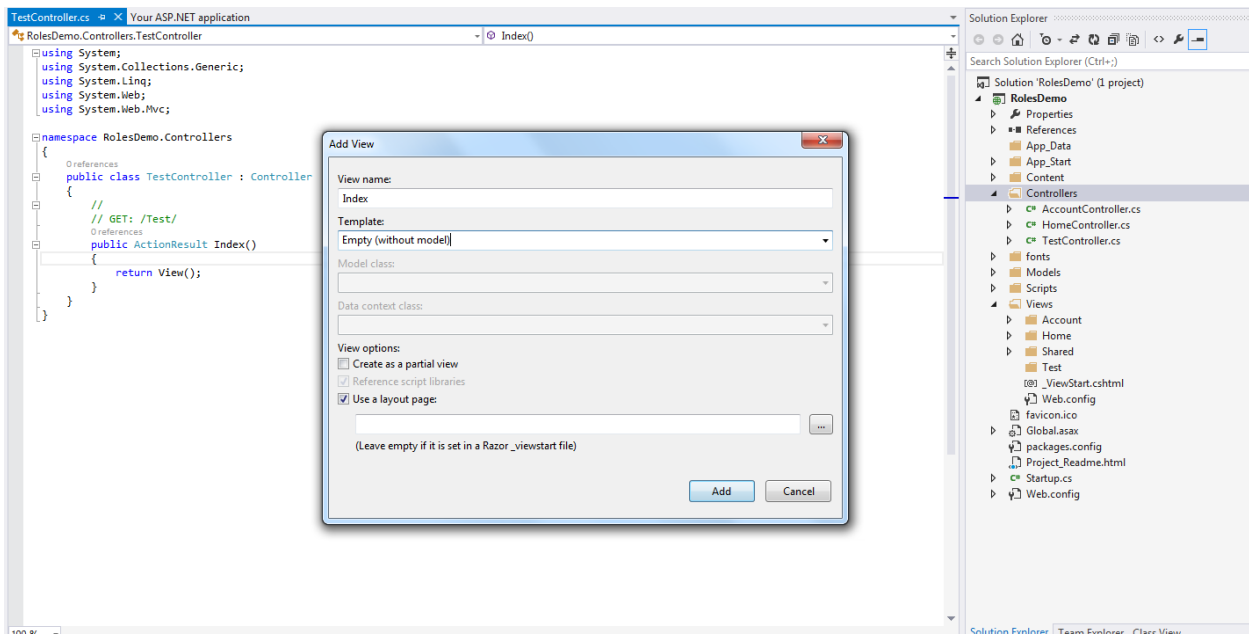
Add a new controller to the application.



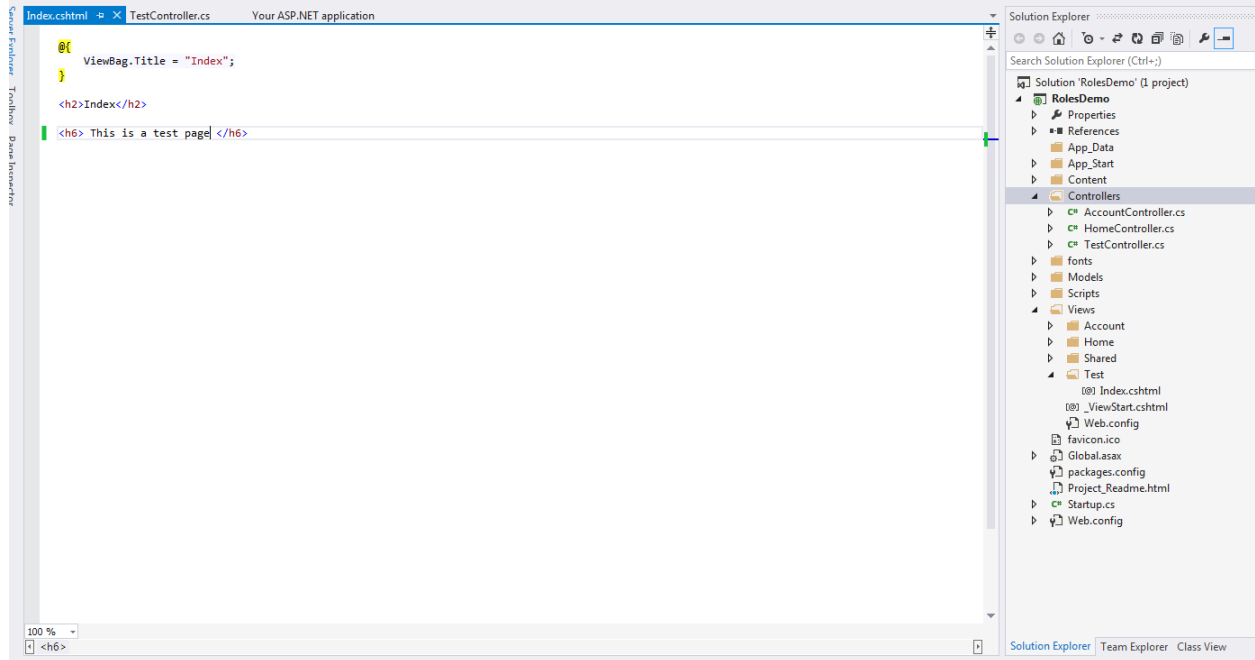
Name the controller 'TestController'.



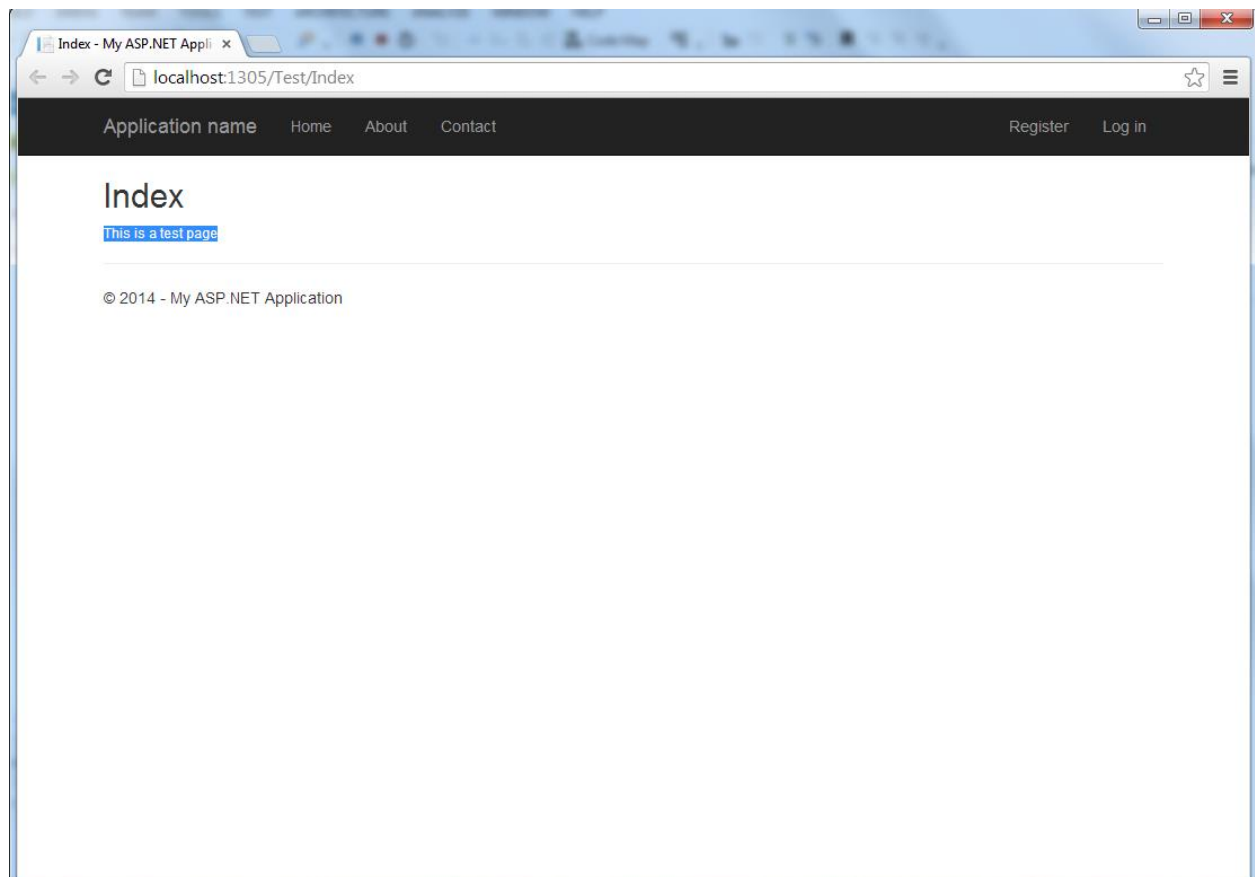
Add a view for the Index method of the TestController.



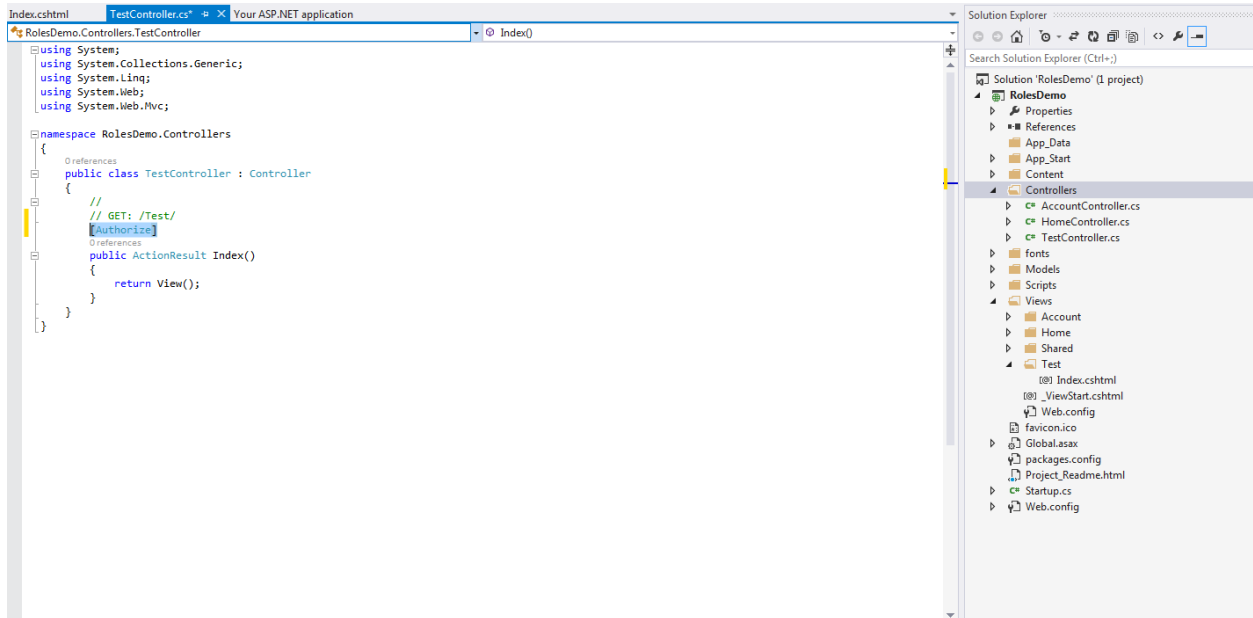
Here the view is shown. An additional heading has been added (This is a test page).



Run the application and navigate to /Test/Index as shown:

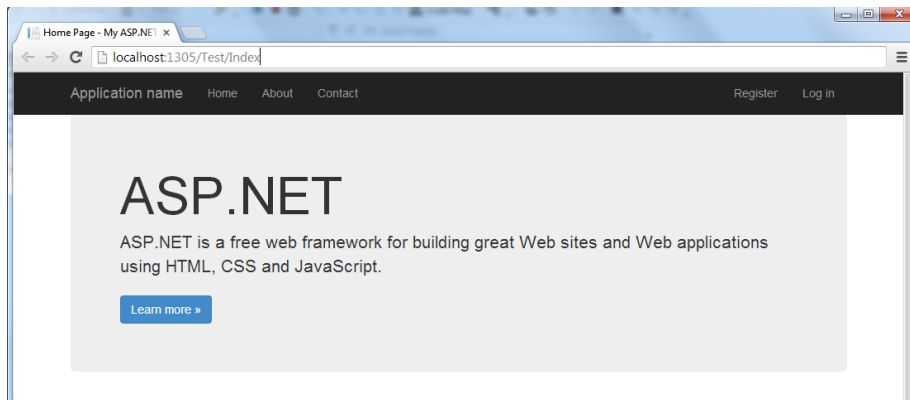


To add the requirement of user authentication to view a page, we will use the [Authorize] attribute as shown :

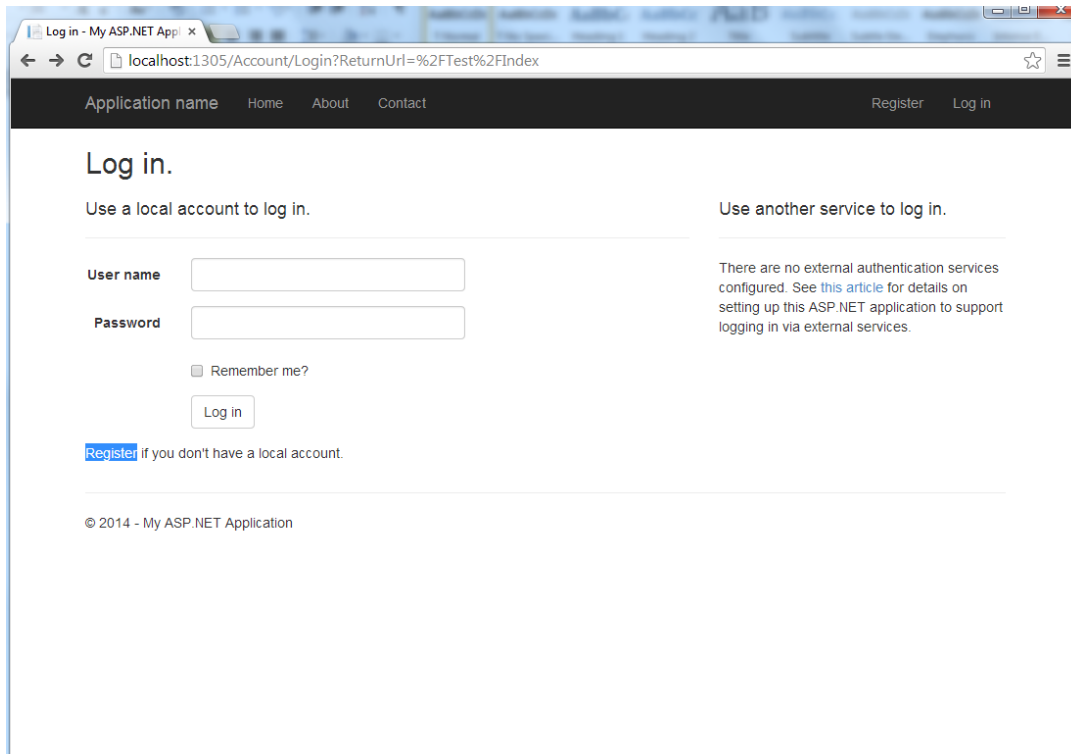


The above should not allow any user to view page `/Test/Index` unless logged in.

Run the application and navigate to `/Test/Index`.

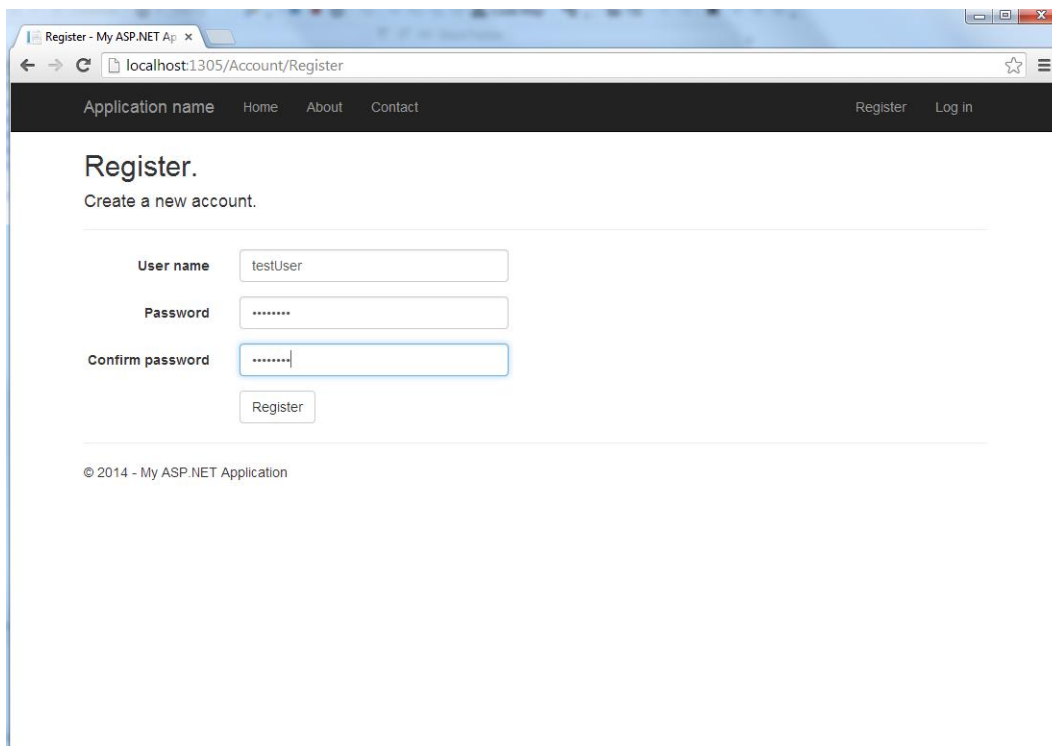


The page is redirected to the login page.



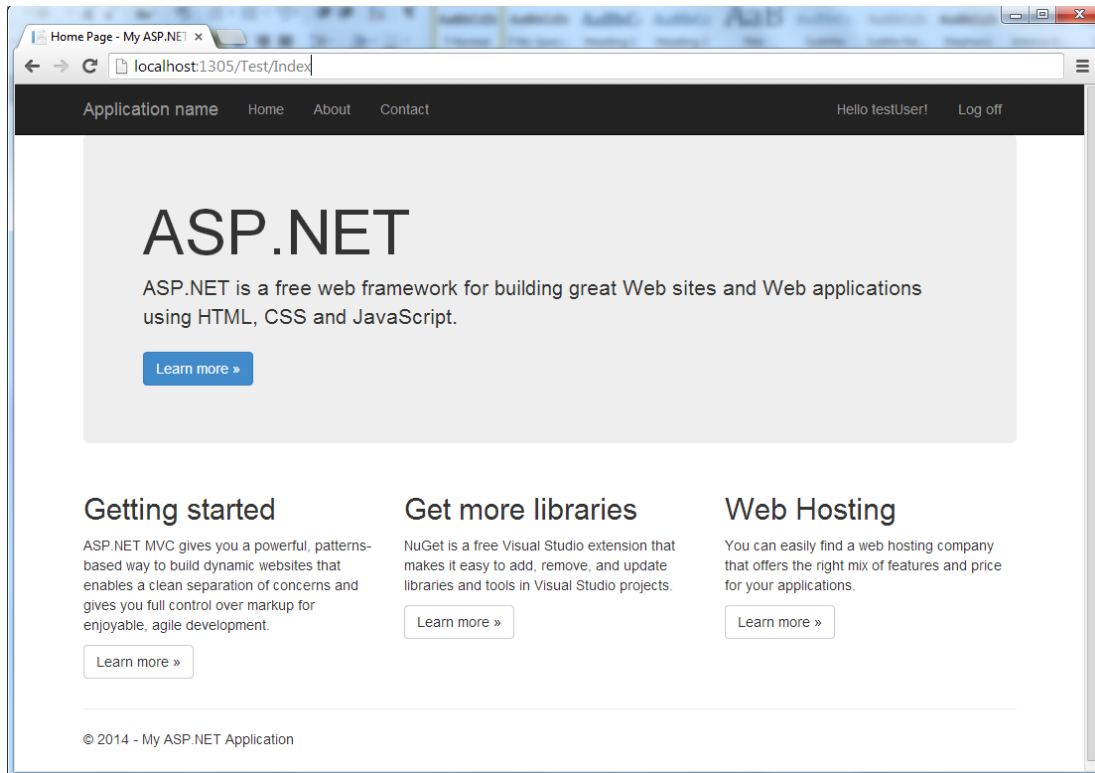
The screenshot shows a web browser window with the address bar displaying 'localhost:1305/Account/Login?ReturnUrl=%2FTest%2FIndex'. The page has a dark navigation bar with 'Application name', 'Home', 'About', 'Contact', 'Register', and 'Log in' links. The main content area is titled 'Log in.' and is divided into two columns. The left column is for local accounts, with fields for 'User name' and 'Password', a 'Remember me?' checkbox, and a 'Log in' button. Below these fields is a link to 'Register' with the text 'if you don't have a local account.' The right column is titled 'Use another service to log in.' and contains a message: 'There are no external authentication services configured. See this article for details on setting up this ASP.NET application to support logging in via external services.' The footer shows '© 2014 - My ASP.NET Application'.

Create a new account called 'testUser'.

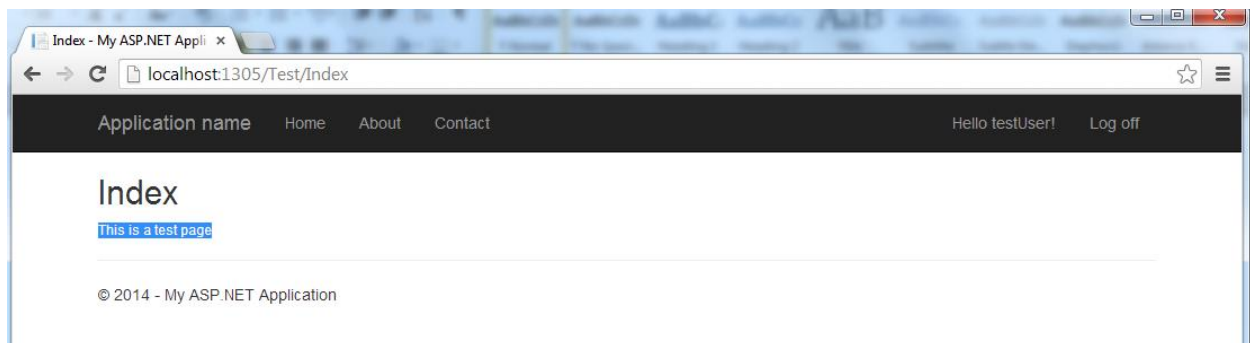


The screenshot shows a web browser window with the address bar displaying 'localhost:1305/Account/Register'. The page has a dark navigation bar with 'Application name', 'Home', 'About', 'Contact', 'Register', and 'Log in' links. The main content area is titled 'Register.' and is for creating a new account. It has fields for 'User name' (containing 'testUser'), 'Password' (masked with dots), and 'Confirm password' (masked with dots). A 'Register' button is located below the fields. The footer shows '© 2014 - My ASP.NET Application'.

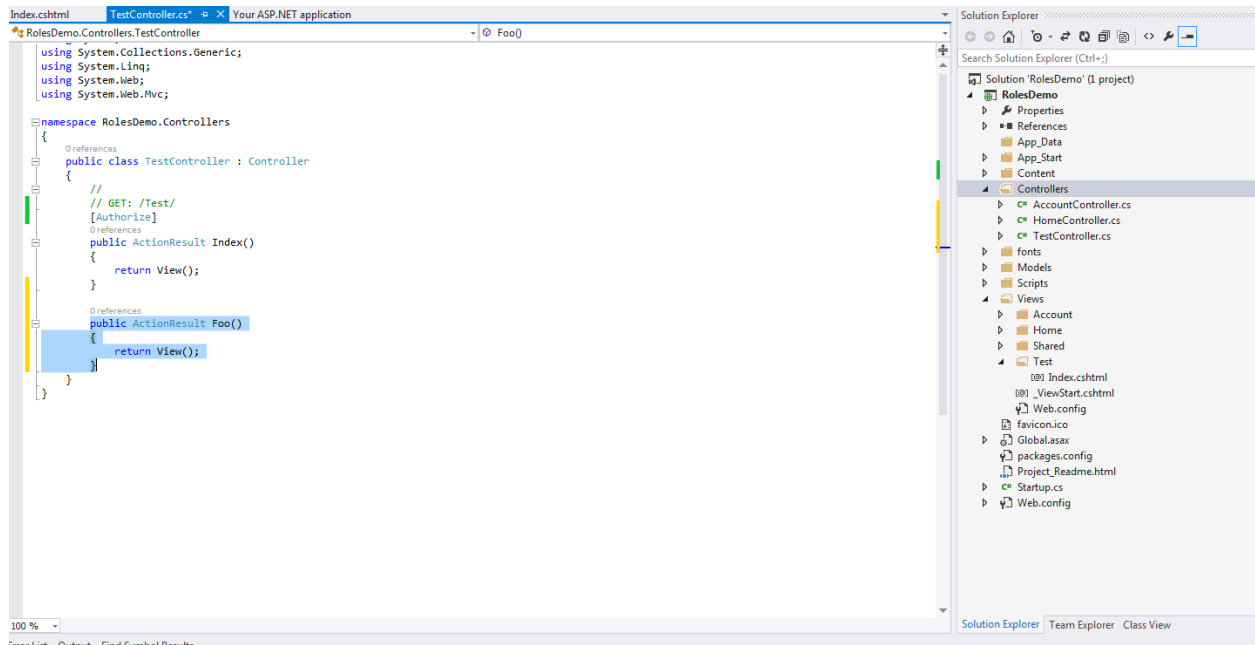
After registration, navigate to /Test/Index



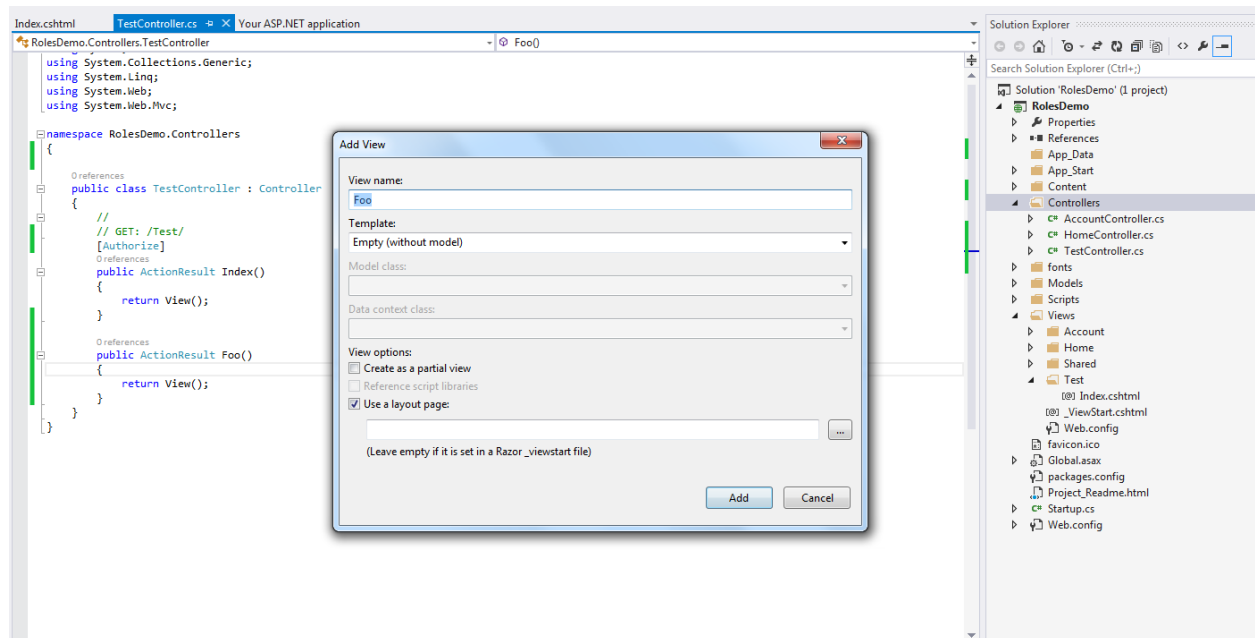
Now the user is logged in, /Test/Index can be viewed.



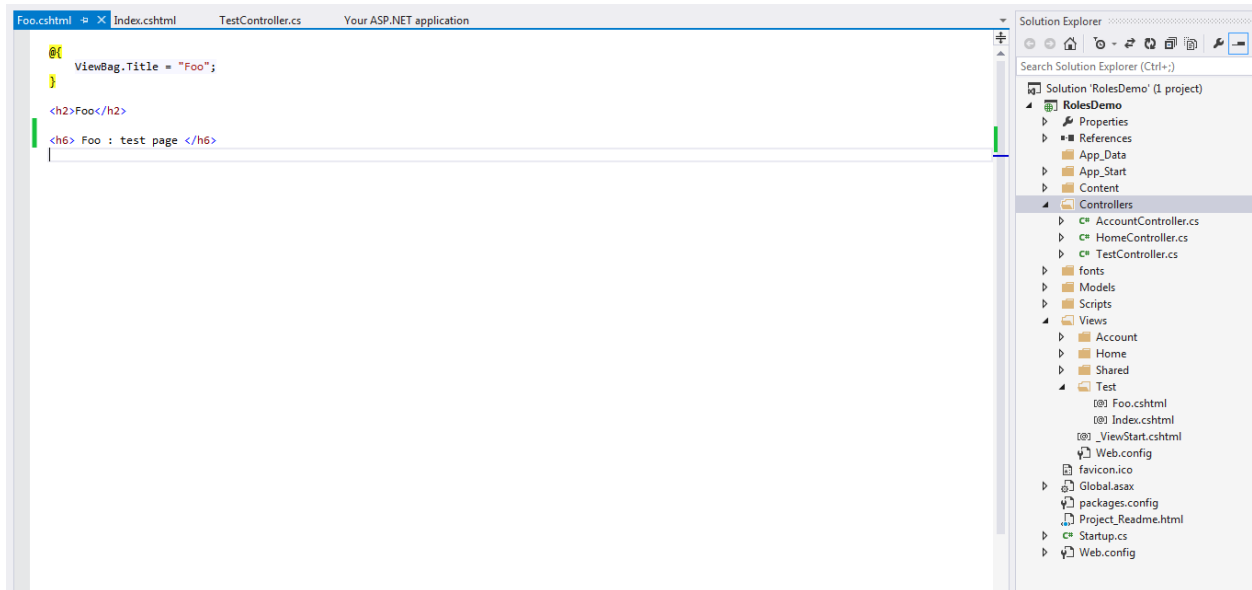
Add another action to the TestController called 'Foo'.



Add a view to the action 'Foo'.

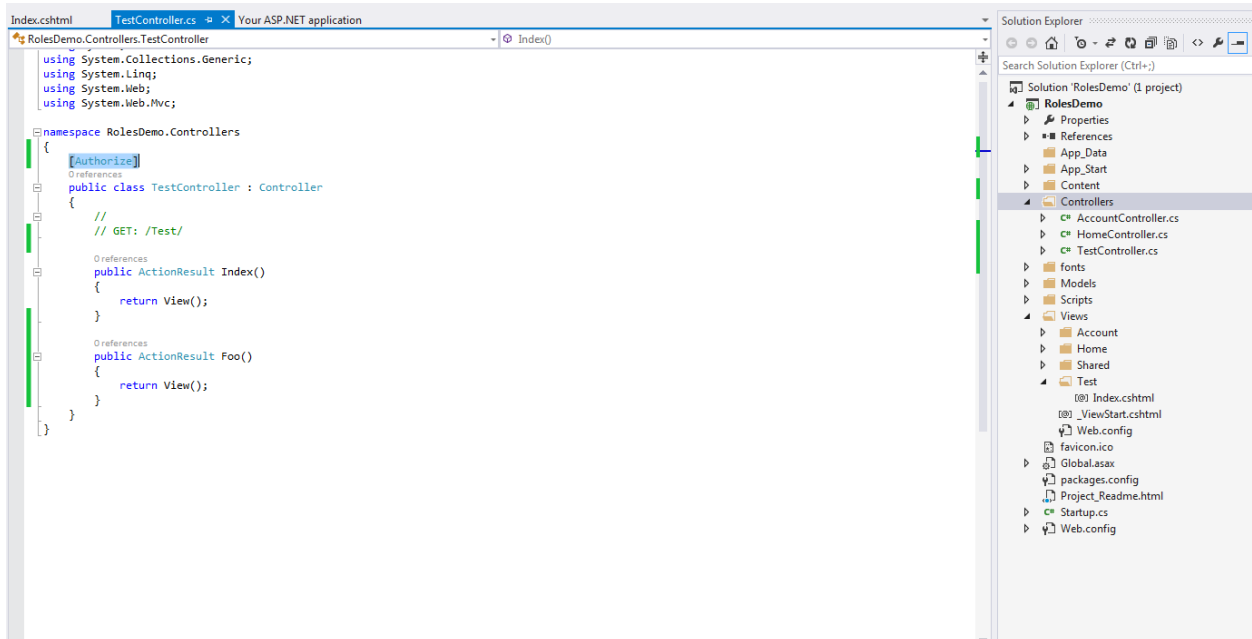


An additional heading has been added to the new page (Foo : test page).

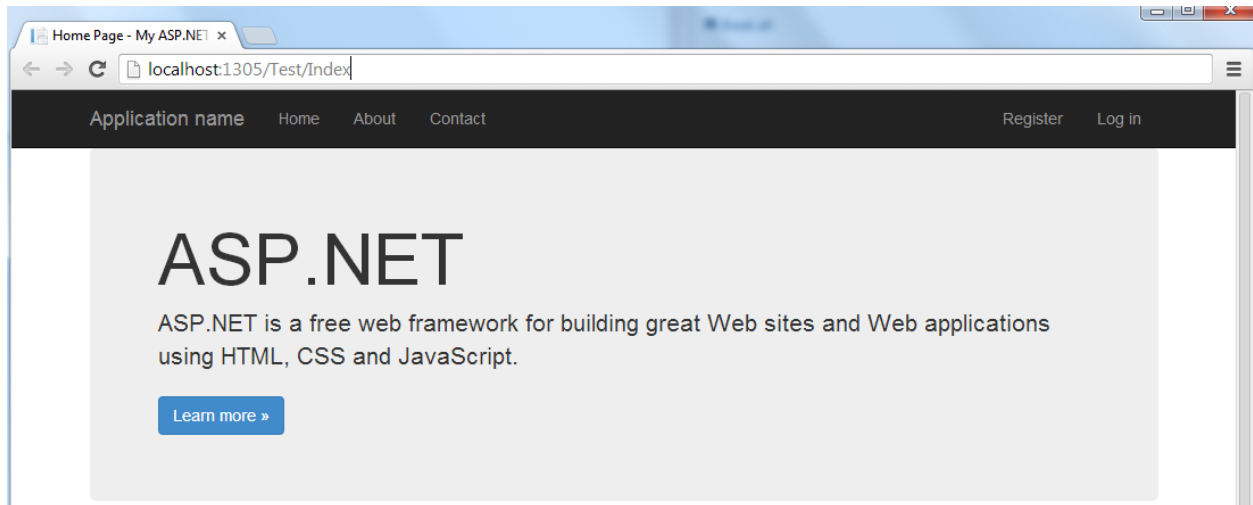


The [Authorize] attribute can be applied to the Controller as well. That will affect every action in that Controller.

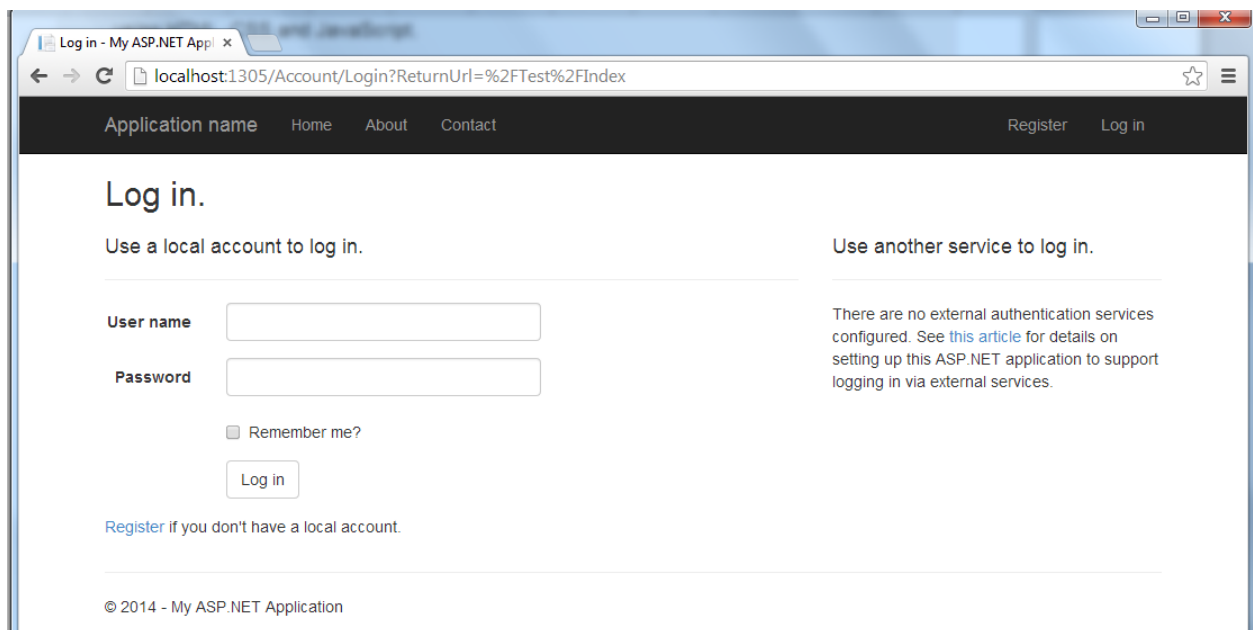
Remove the [Authorize] attribute from the Index action and add it to the TestController as follows:



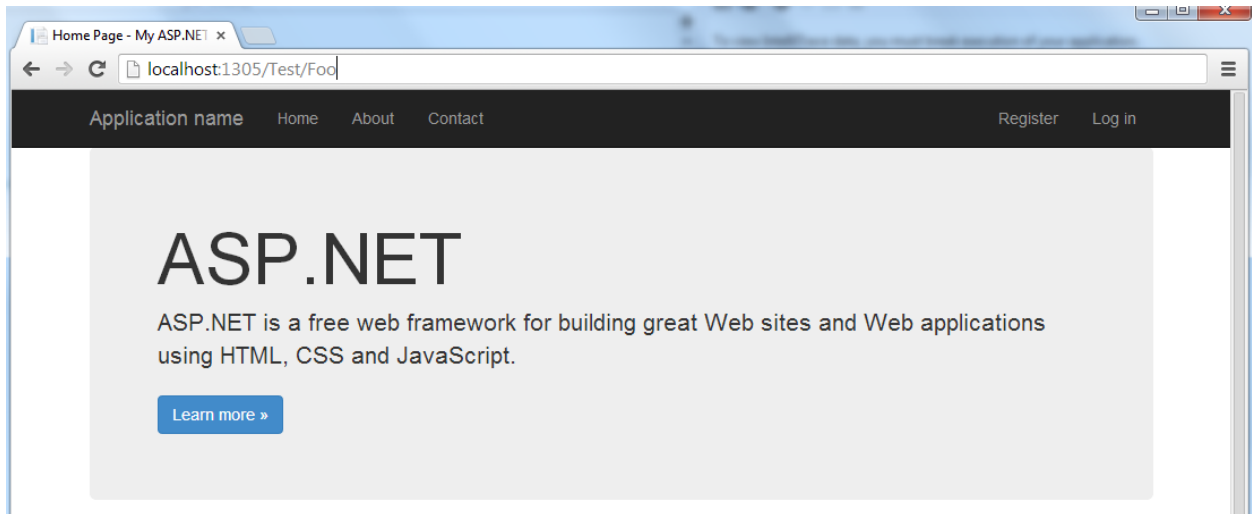
Run the application, and navigate to /Test/Index :



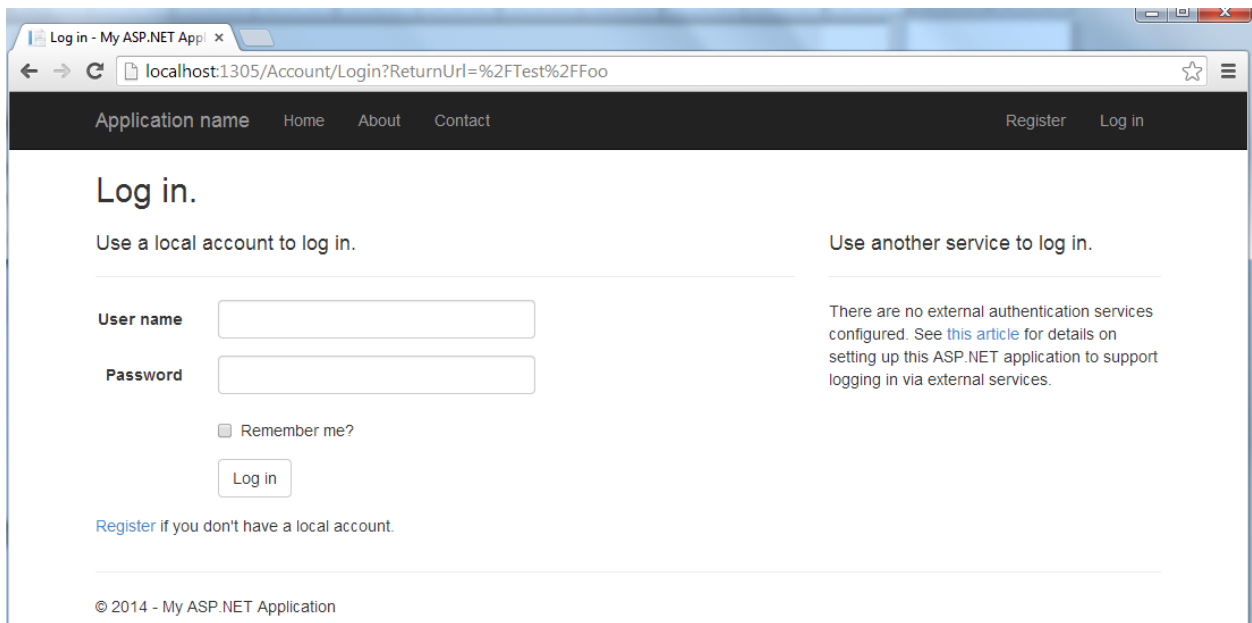
Navigation to /Test/Index redirects the page to login page.



Now try navigating to /Test/Foo. Since the attribute was applied to the controller, this navigation should also redirect to login page if user is not logged in.

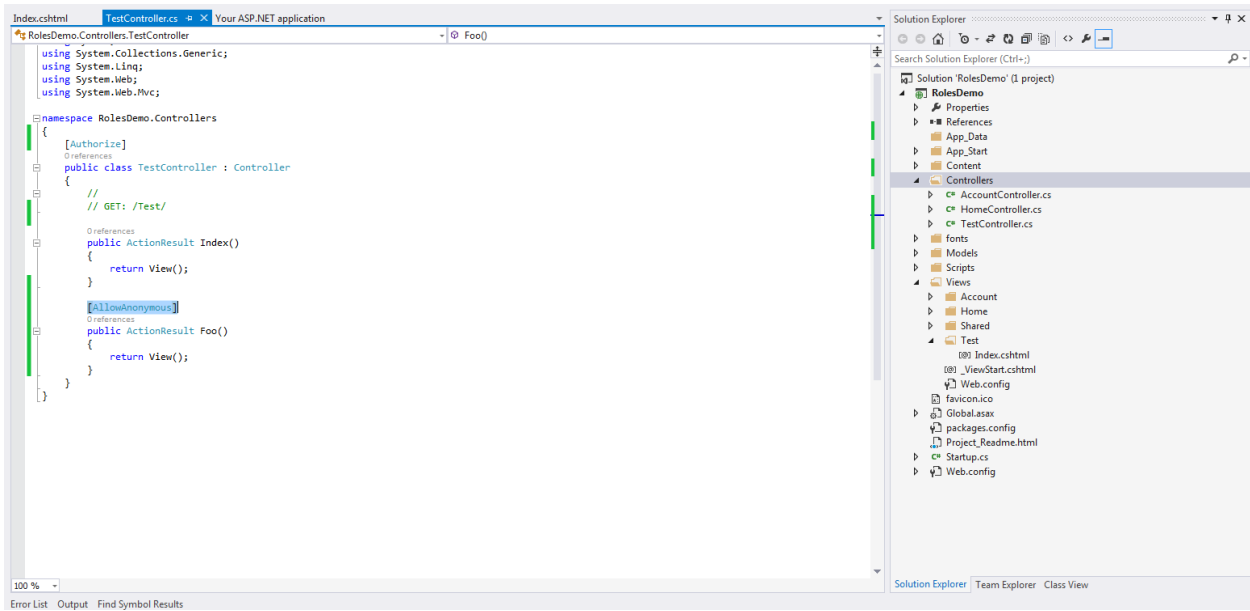


Navigation to /Test/Foo redirects to login page.



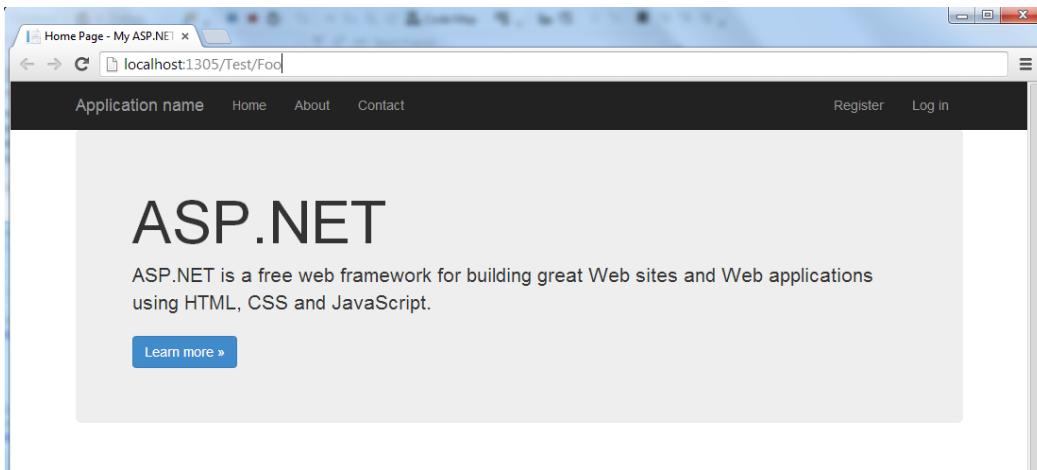
There are situations when you want to exclude some of actions, but still apply the [Authorize] attribute to the controller. For this, the [AllowAnonymous] attribute is used.

Apply the [AllowAnonymous] attribute to the Foo action as follows:

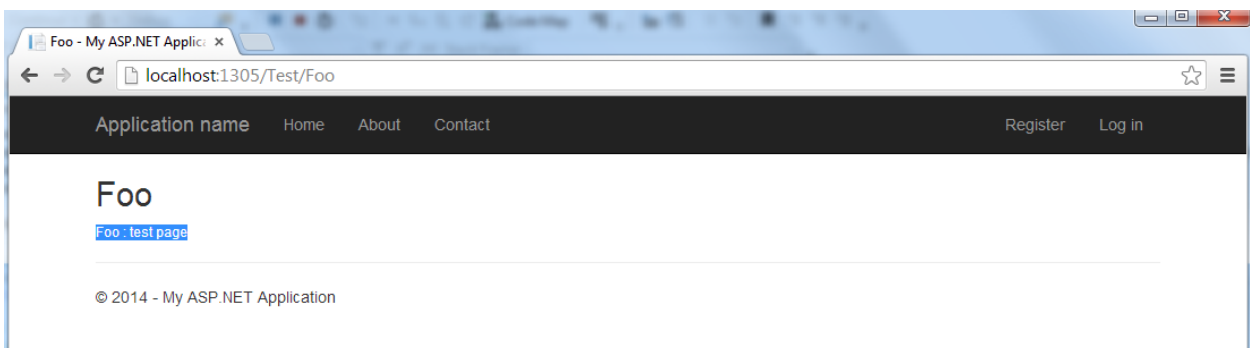


From the above, navigation to action methods other than 'Foo' should require authentication.

Run the application and navigate to /Test/Foo:

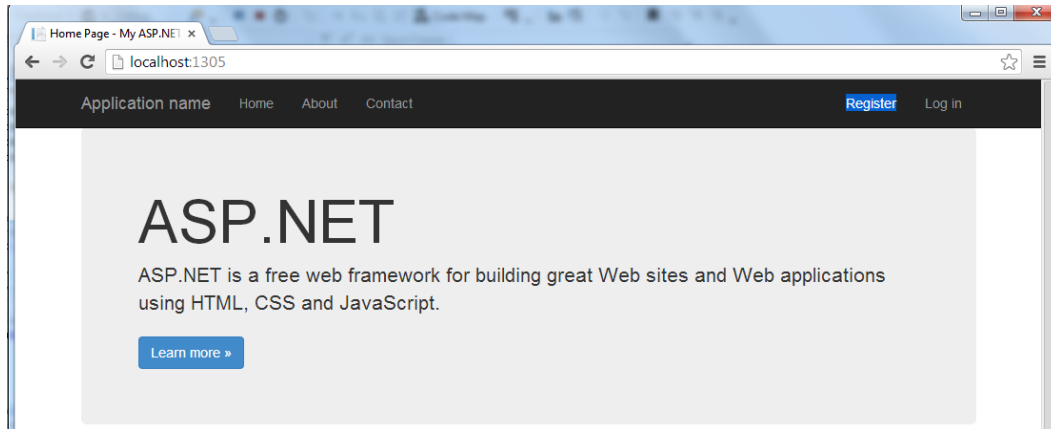


The page is accessible even though user is logged out, because of the [AllowAnonymous] attribute.

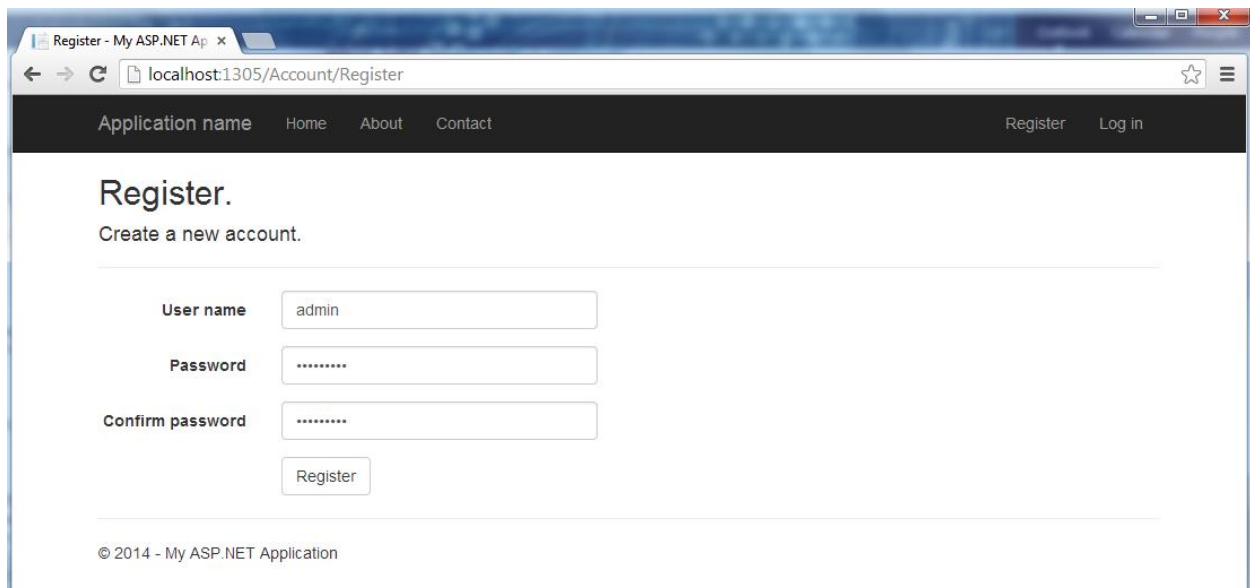


How to make specific pages available only to select users

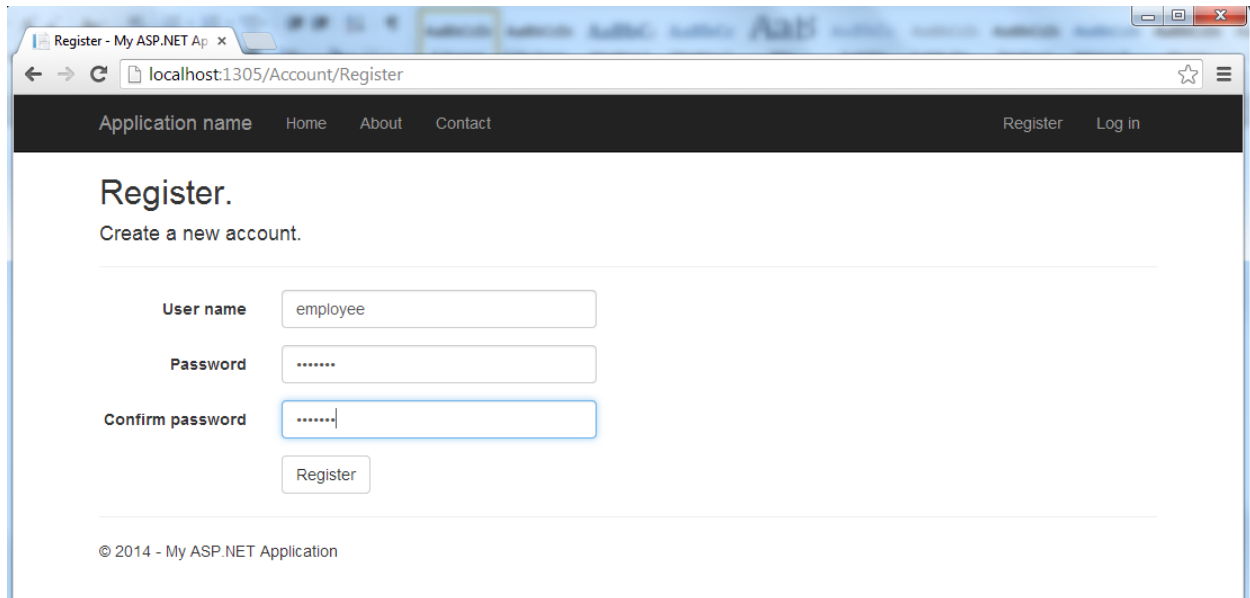
For this demo, we need to create two test users. Run the application and click Register.



Register a user named 'admin'.



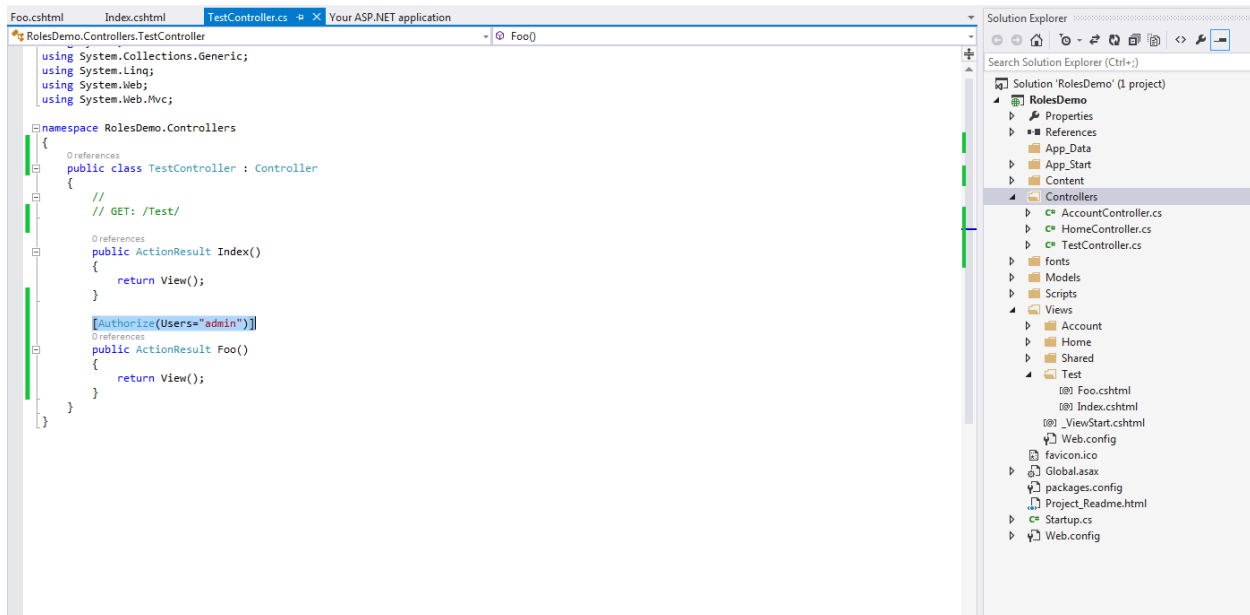
After registering 'admin' user, register another user named 'employee'.



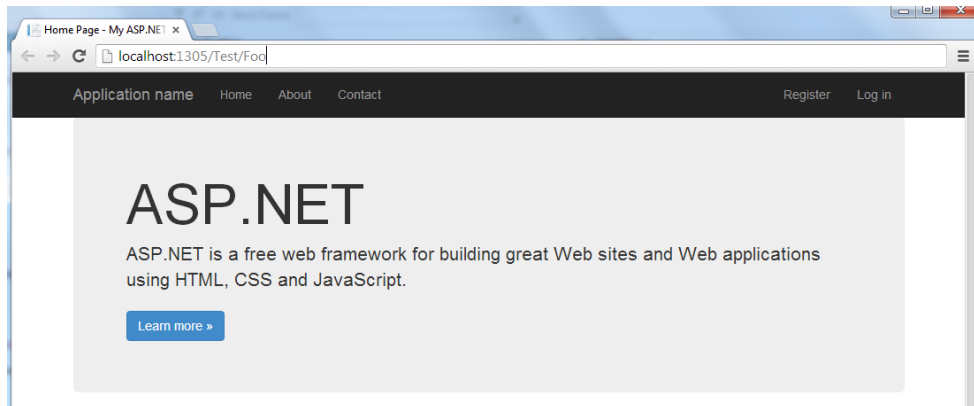
To specify which users are allowed to view a page after authentication, we use the [Authorize] attribute, and specify a comma separated list of users.

Remove the [Authorize] attribute from the TestController.

Add the [Authorize (Users="admin")] attribute to Foo action method to specify only 'admin' user can view that page after logging in.

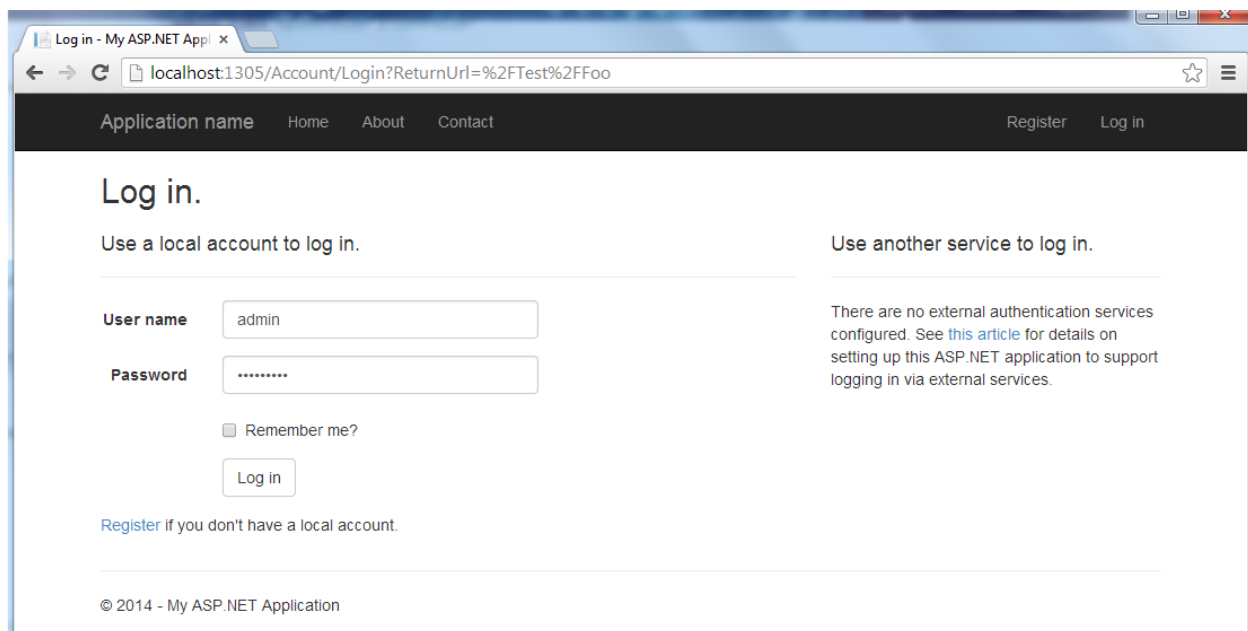


Run the application and navigate to /Test/Foo.

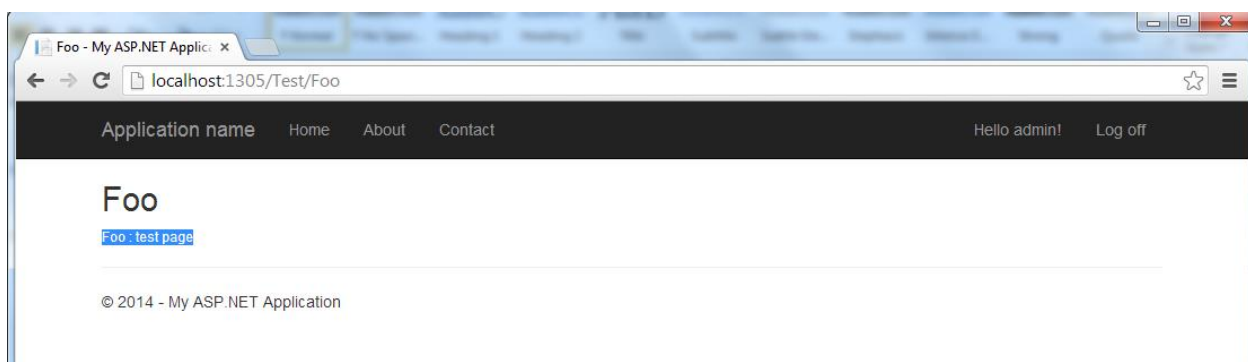


Because of the [Authorize] attribute, the page is redirected to login page.

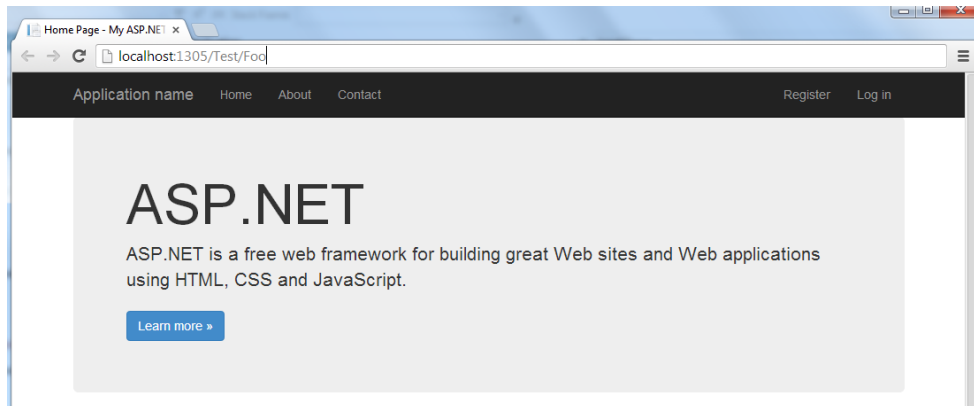
Login as the 'admin' user.



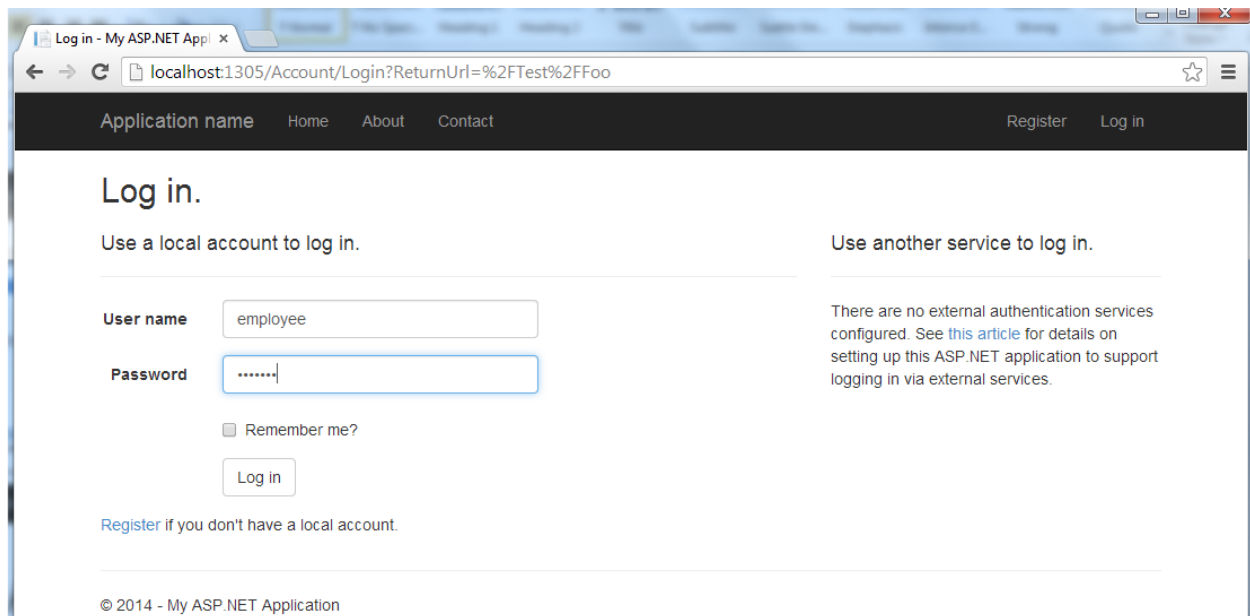
After logging in, the 'admin' user can view the page.



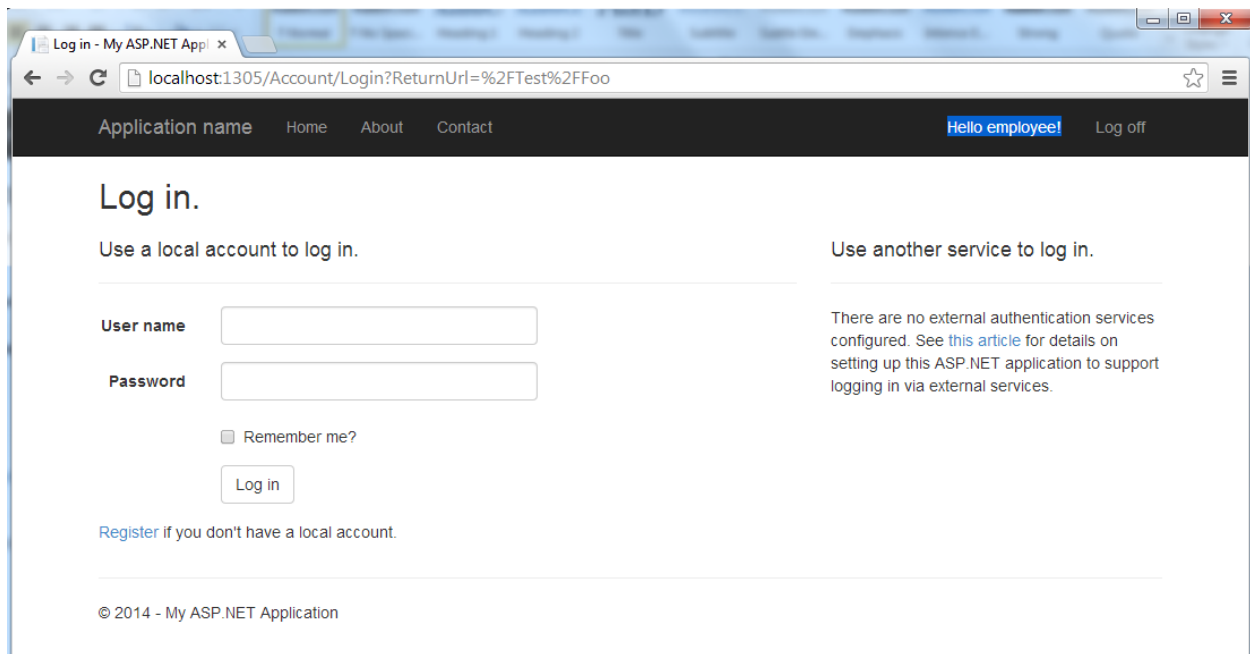
Log out the 'admin' user. Navigate to /Test/Foo as follows:



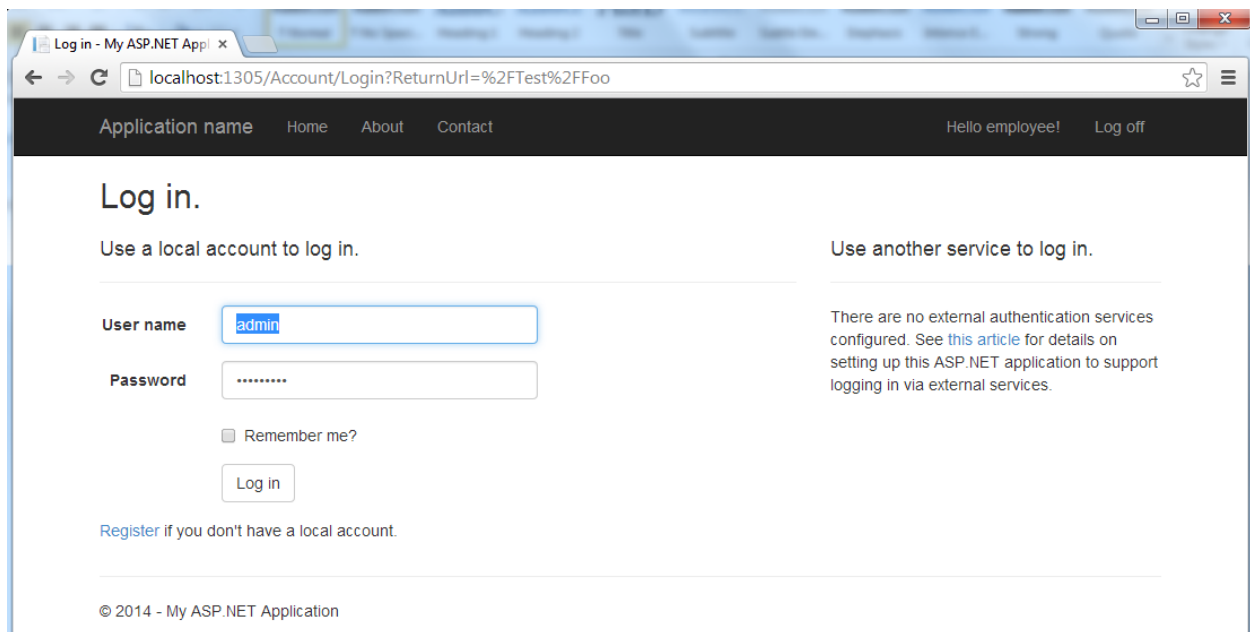
The page will be redirected. Now try logging in as 'employee'. Since 'employee' is not in the list of allowed users, even after logging in, 'employee' must not be able to navigate to /Test/Foo.



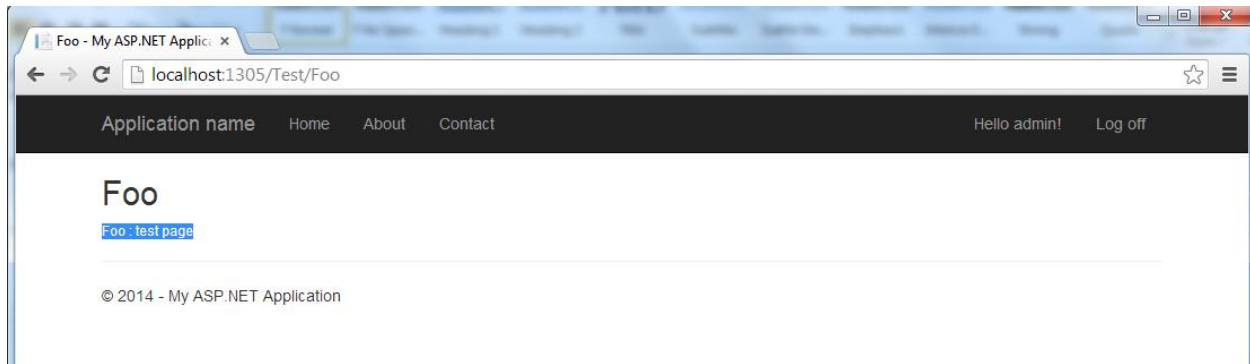
Now try navigating to /Test/Foo. You will observe that the page is navigated to the login page again, even though the 'employee' user is logged in. It won't allow the navigation to /Test/Foo unless a user in the [Authorize] attribute list is logged in.



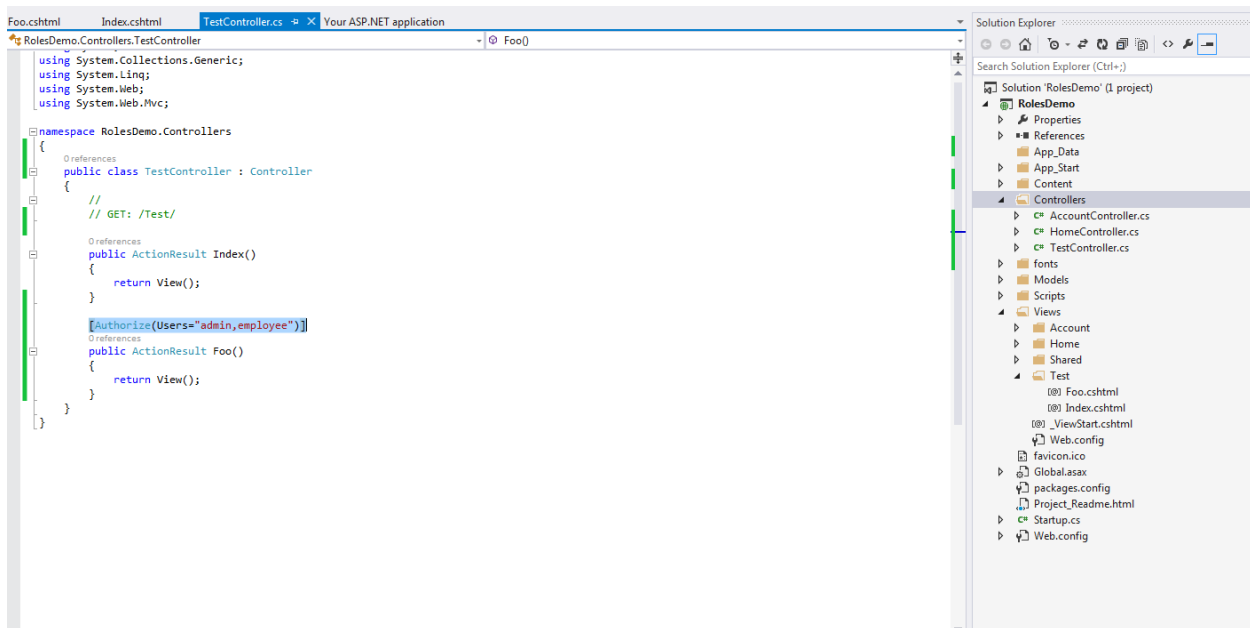
Log in as the 'admin' user again.



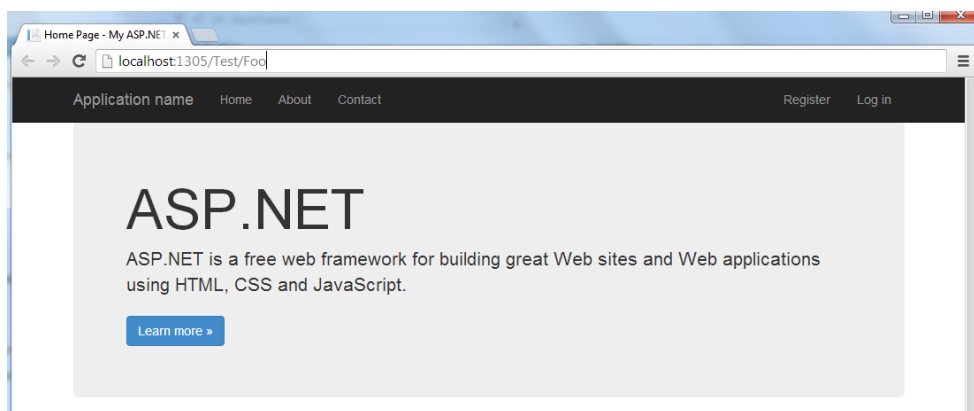
It can be observed that the navigation was allowed, once a user in the list logged in.



To allow both 'admin' and 'employee' users to view that page, edit the 'Users' list in the [Authorize] attribute of the Foo action method and add 'employee' to that list as follows:

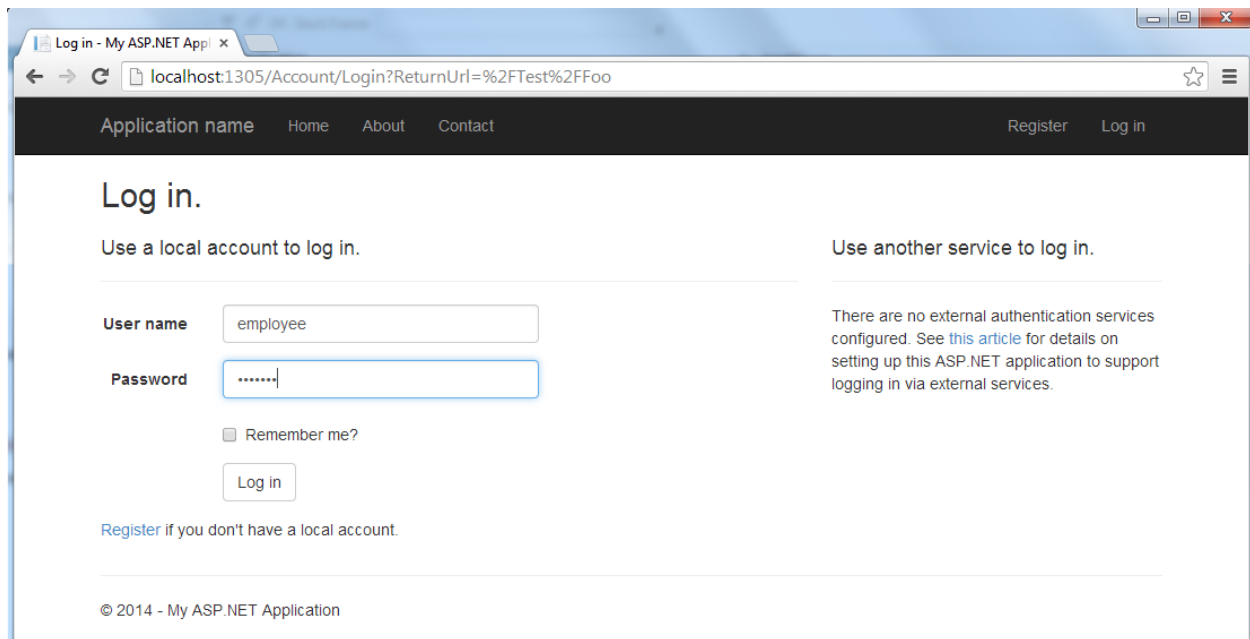


Run the application and navigate to /Test/Foo.

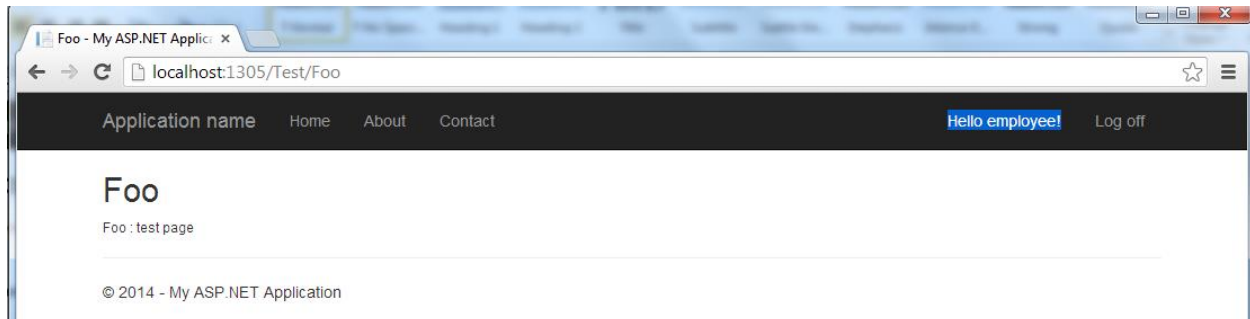


It will be redirected to login page.

Login as the 'employee' user.



Navigate to /Test/Foo, and the page will be displayed for 'employee' user.



How to make specific pages available only to select roles

In Visual Studio 2013, the ASP .NET Web configuration tool was removed which assisted in managing roles.

To complete this task, we have to manage roles by using an alternative method, by editing some tables using SQL Server Explorer.

The tables we are interested in are:

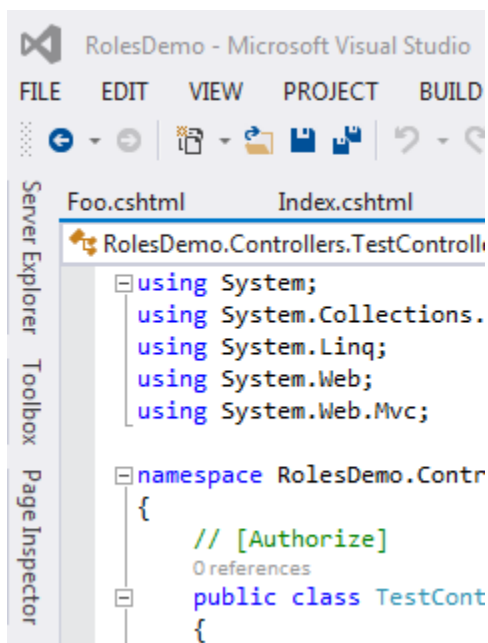
- 1) AspNetRoles – stores role id and name
- 2) AspNetUserRoles – stores mapping of user id to role id
- 3) AspNetUsers – stores user id and user name

The first step is to create some users in the website. For the remainder of this task, it will be assumed that the following users exist:

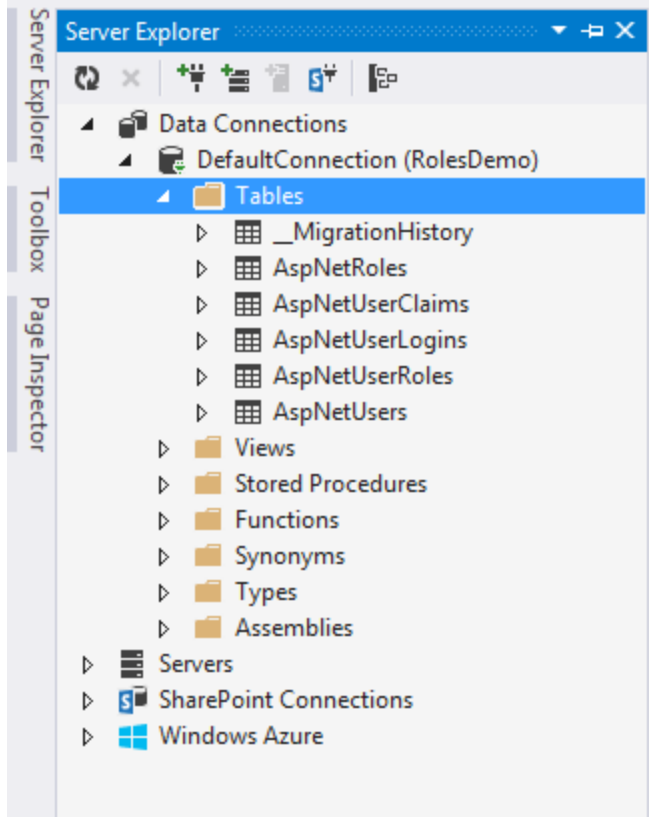
- 1) admin
- 2) employee
- 3) employee2

The first step is to open Server Explorer and get to the window where we can run some queries in the tables.

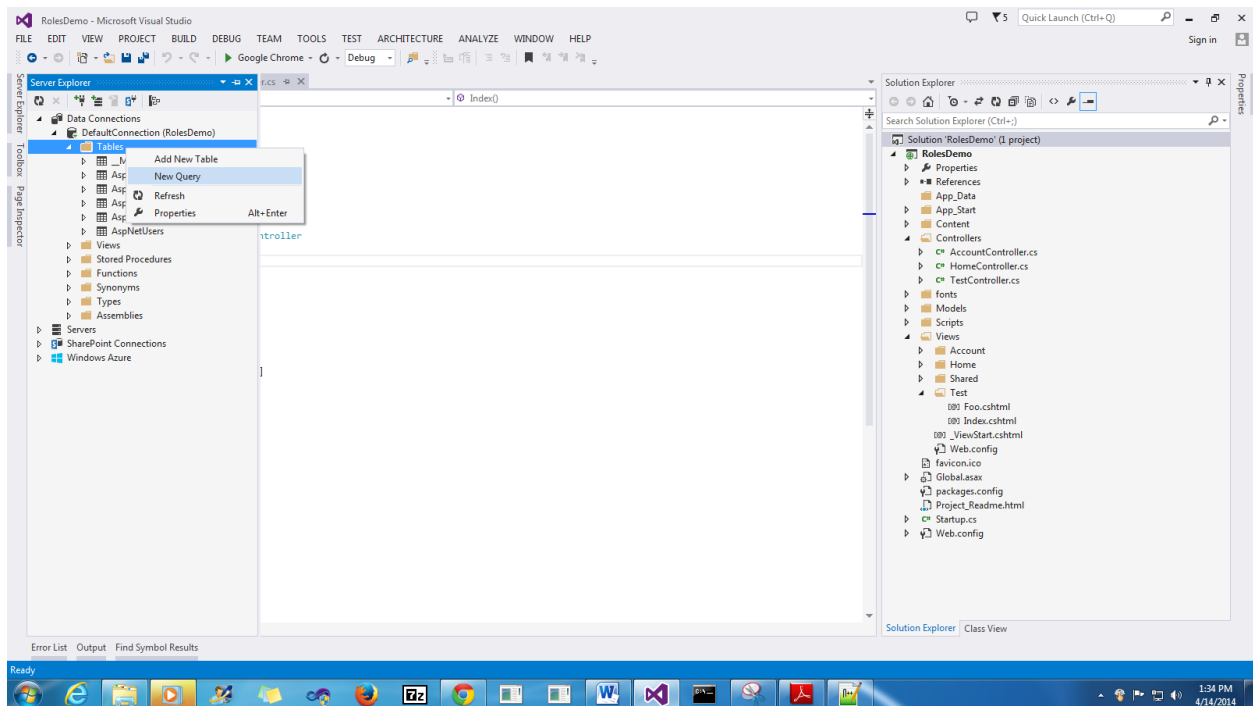
Server Explorer appears as an option in the left column as shown:



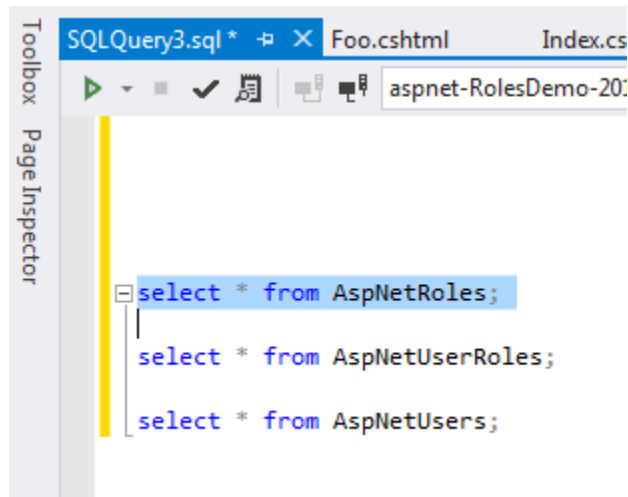
Click and navigate to the Server Explorer and open the tables menu as shown:



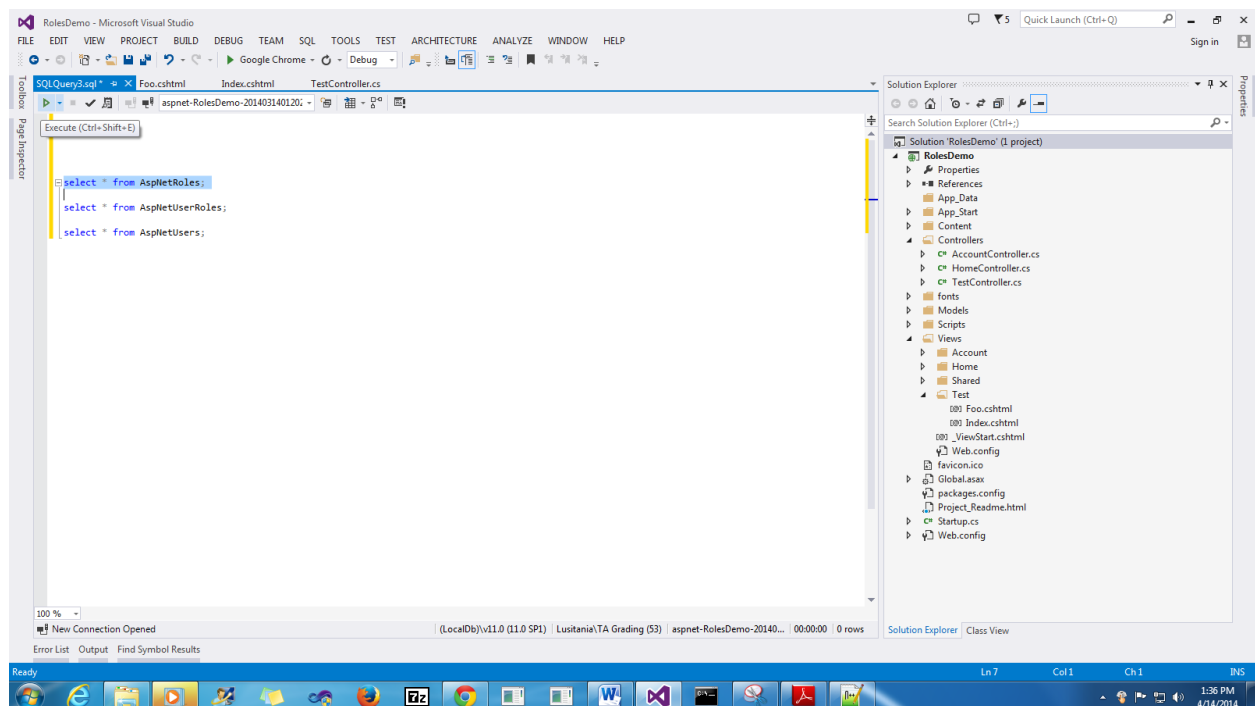
Right click and select new query.



First let's examine the 3 tables. Use select statements as shown below.



In Server Explorer you can highlight and execute individual lines. The execute button is the green play button in the top left corner.



```

select * from AspNetRoles;
select * from AspNetUserRoles;
select * from AspNetUsers;

```

100 %

T-SQL Results Message

Id	Name
----	------

```

select * from AspNetRoles;
select * from AspNetUserRoles;
select * from AspNetUsers;

```

0 %

T-SQL Results Message

Userid	Roleid
--------	--------

```

select * from AspNetRoles;
select * from AspNetUserRoles;
select * from AspNetUsers;

```

100 %

T-SQL Results Message

	Id	UserName	PasswordHash	SecurityStamp	Discriminator
1	009e87dd-7c36-47e9-8eab-778f39e289d9	admin	AOoZ5Rd7qgoCOK7B617x/QPBU8ZkbXzbxzSN1y7WQhazPm+E...	de333473-13f5-47d7-b9a3-2c80c74421bc	Application User
2	4d33b74f-91e2-4788-9904-33ecfa754e3b	testUser	AEIzLuH3N8Yx0yjTEuch50C9T9/ZhE8P+4XVOAdr/GGSrLpP5h1T...	16df5549-8b66-4d22-8464-1c0f90f978a7	Application User
3	64682311-5437-445a-b696-2234b48bc1c	employee	AKUv8Bok#TXb7PX4Gj8spPsT92F+O5PwTkZyrGMq9rO2G8Pv7...	3849eee9-157e-4a58-940a-4b23559c3619	Application User
4	b20de7fa-ef90-4b70-959d-87ab3c83622a	employee2	AJ06MJeIW10hjrH0HuSfBjno5fRdXBo+aleTdQ1nZ05FNVZTJH...	0bc2aa58-a181-4f4d-a87e-d21af205c15a	Application User

As can be seen, the AspNetRoles, and AspNetUserRoles tables are empty, whereas the AspNetUsers contains details about the current users.

In this task, we will create two roles, namely admin and employee and assign them to users as follows:

- 1) admin user -> admin role
- 2) employee & employee2 user -> employee role

The next step is to create the two roles, admin and employee.

Use insert statements as shown below:

```
select * from AspNetRoles;
select * from AspNetUserRoles;
select * from AspNetUsers;
insert into AspNetRoles(Id,Name) values(1,'admin');
insert into AspNetRoles(Id,Name) values(2,'employee');
```

Execute the insert statements (you can highlight them both and click the execute button).

```
select * from AspNetRoles;
select * from AspNetUserRoles;
select * from AspNetUsers;
insert into AspNetRoles(Id,Name) values(1,'admin');
insert into AspNetRoles(Id,Name) values(2,'employee');
```

100 %

T-SQL Message

(1 row(s) affected)

(1 row(s) affected)

Check the contents of the AspNetRoles table.

```
select * from AspNetRoles;
select * from AspNetUserRoles;
select * from AspNetUsers;
insert into AspNetRoles(Id,Name) values(1,'admin');
insert into AspNetRoles(Id,Name) values(2,'employee');
```

100 %

T-SQL Results Message

	Id	Name
1	1	admin
2	2	employee

The next step is to identify the user ids we want to map.

Record the user ids of the desired users somewhere (in this demo, those are just pasted in the same window to keep track of them).

```
select * from AspNetRoles;
select * from AspNetUserRoles;
select * from AspNetUsers;
insert into AspNetRoles(Id,Name) values(1,'admin');
insert into AspNetRoles(Id,Name) values(2,'employee');
```

100 %

T-SQL Results Message

	Id	UserName	PasswordHash	SecurityStamp	Discriminator
1	009e87dd-7c36-47e9-8eab-77f39e289d9	admin	AOoZ5Rd7qgoCOk7B617x/QPBU8ZkbXzbxzSN1y7WQhazPm+E...	de333473-13f5-47d7-b9a3-2c80c74421bc	ApplicationUser
2	4d33b74f-91e2-478-9904-33ecfa754e3b	testUser	ABzLuH3N8Yx0yjTEuch50C9T9/ZhE8P+4XV0Adr/GGSrLtP5h1T...	16df5549-8b66-4d22-8464-1c0f90f978a7	ApplicationUser
3	64682311-5437-445a-b696-2234b48fbc1c	employee	AKUvBBokfITXb7PX4Gj8spPsT92F+O5PwTkZfyrGMq9O2GBPV7...	3849eee9-157e-4a58-940a-4b23559c3619	ApplicationUser
4	b20de7a-ef90-4b70-959d-87ab3c83622a	employee2	AIJ06MJeIW10hjrHfOHuSfBjno5fRdXBo+aleTdQ1nZ05FNVZTJH...	0bc2aa58-a181-4f4d-a87e-d21af205c15a	ApplicationUser

```

select * from AspNetUsers;

insert into AspNetRoles(Id,Name) values(1,'admin');

insert into AspNetRoles(Id,Name) values(2,'employee');

```

admin - 009e87dd-7c36-47e9-8eab-776f39e289d9

employee - 64682311-5437-445a-b696-2234b48fbc1c

employee2 - b20de7fa-ef90-4b70-959d-87ab3c83622a

	Id	UserName	PasswordHash	SecurityStamp	Discriminator
1	009e87dd-7c36-47e9-8eab-776f39e289d9	admin	AOoZ5Rd7qgoCOK7B617x/QPBU8ZkbXztkzSN1y7WQhazPm+E...	de333473-13f5-47d7-b9a3-2c80c74421bc	ApplicationUser
2	4d33b74f-91e2-4788-9904-33ecfa754e3b	testUser	AEIzLuH3N8Yx0yjTEuch50C9T9/ZhE8P+4XVOAdr/GGSrLp5h1T...	16df5549-8b66-4d22-8464-1c0f90f978a7	ApplicationUser
3	64682311-5437-445a-b696-2234b48fbc1c	employee	AKUvBBoklFTXb7PX4Gj8spPsT92F+O5PwTkZfyrGMq9O2GBPv7...	3849eee9-157e-4a58-940a-4b23559c3619	ApplicationUser
4	b20de7fa-ef90-4b70-959d-87ab3c83622a	employee2	AJ06MJeIW10hjrHfOHuSfBjno5fRdXBo+aleTdQ1nZ05FNVZTJH...	0bc2aa58-a181-4f4d-a87e-d21af205c15a	ApplicationUser

The next step is to insert the mappings between user id and role id. From before, we inserted the role ids:

- 1) admin role -> id 1
- 2) employee role -> id 2

```

employee - 64682311-5437-445a-b696-2234b48fbc1c

employee2 - b20de7fa-ef90-4b70-959d-87ab3c83622a

insert into AspNetUserRoles(UserId,RoleId) values('009e87dd-7c36-47e9-8eab-776f39e289d9',1);

insert into AspNetUserRoles(UserId,RoleId) values('64682311-5437-445a-b696-2234b48fbc1c',2);

insert into AspNetUserRoles(UserId,RoleId) values('b20de7fa-ef90-4b70-959d-87ab3c83622a',2);

```

Execute the insert statements.

```
admin - 009e87dd-7c36-47e9-8eab-776f39e289d9
employee - 64682311-5437-445a-b696-2234b48fbc1c
employee2 - b20de7fa-ef90-4b70-959d-87ab3c83622a

insert into AspNetUserRoles(UserId,RoleId) values('009e87dd-7c36-47e9-8eab-776f39e289d9',1);
insert into AspNetUserRoles(UserId,RoleId) values('64682311-5437-445a-b696-2234b48fbc1c',2);
insert into AspNetUserRoles(UserId,RoleId) values('b20de7fa-ef90-4b70-959d-87ab3c83622a',2);
```

100 %

T-SQL Message

(1 row(s) affected)

(1 row(s) affected)

(1 row(s) affected)

Check the contents of the AspNetUserRoles table.

```
select * from AspNetRoles;
select * from AspNetUserRoles;
select * from AspNetUsers;

insert into AspNetRoles(Id,Name) values(1,'admin');
insert into AspNetRoles(Id,Name) values(2,'employee');
```

100 %

T-SQL Results Message

	UserId	RoleId
1	009e87dd-7c36-47e9-8eab-776f39e289d9	1
2	64682311-5437-445a-b696-2234b48fbc1c	2
3	b20de7fa-ef90-4b70-959d-87ab3c83622a	2

Now our user to role mapping is complete.

The last step is to use these roles in Authorize attribute. Edit the Test Controller as follows (replacing a user list with role list):

```
SQLQuery3.sql *   Foo.cshtml   Index.cshtml   TestController.cs -P X
RolesDemo.Controllers.TestController
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

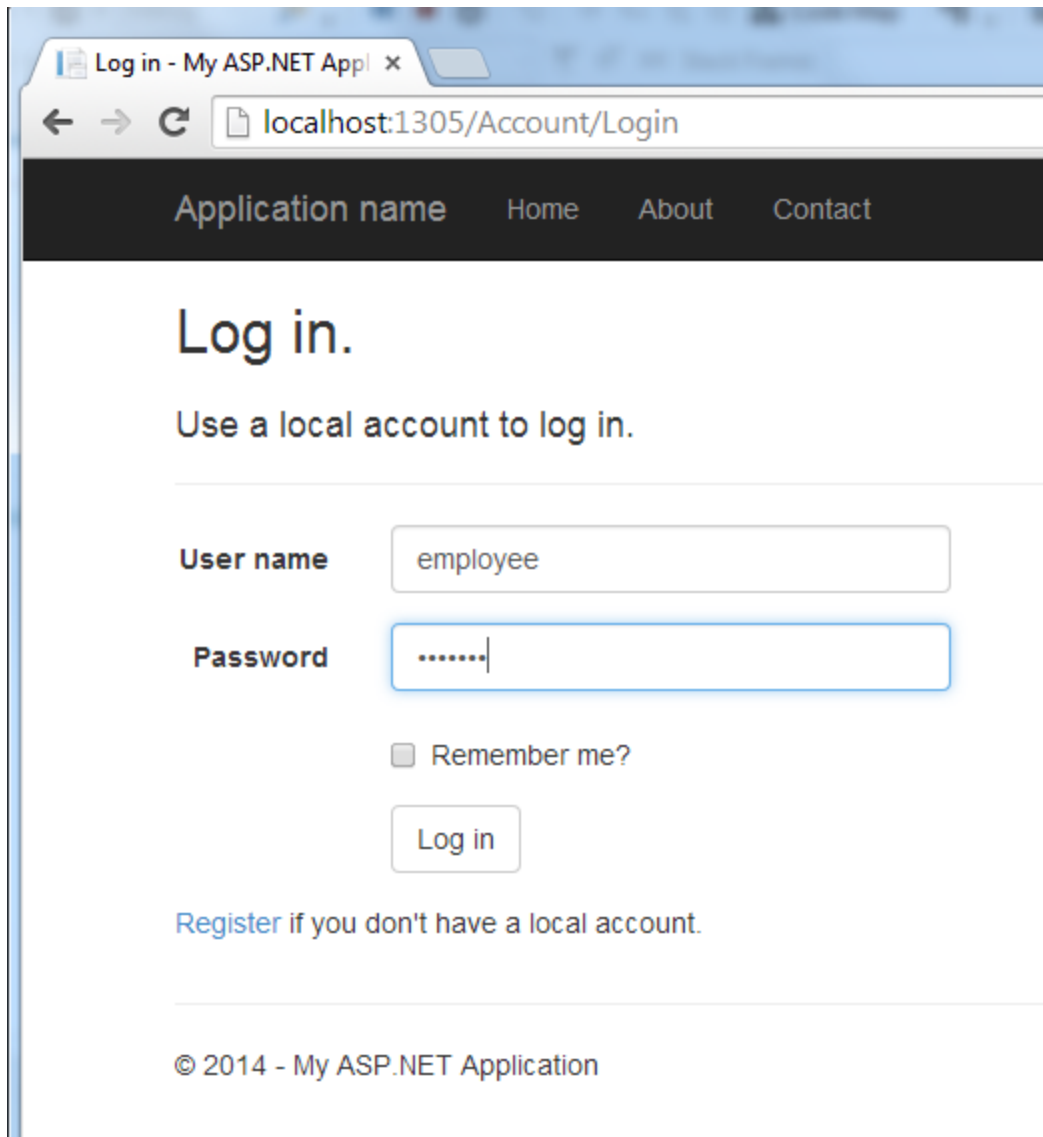
namespace RolesDemo.Controllers
{
    // [Authorize]
    // 0 references
    public class TestController : Controller
    {
        //
        // GET: /Test/

        // 0 references
        public ActionResult Index()
        {
            return View();
        }

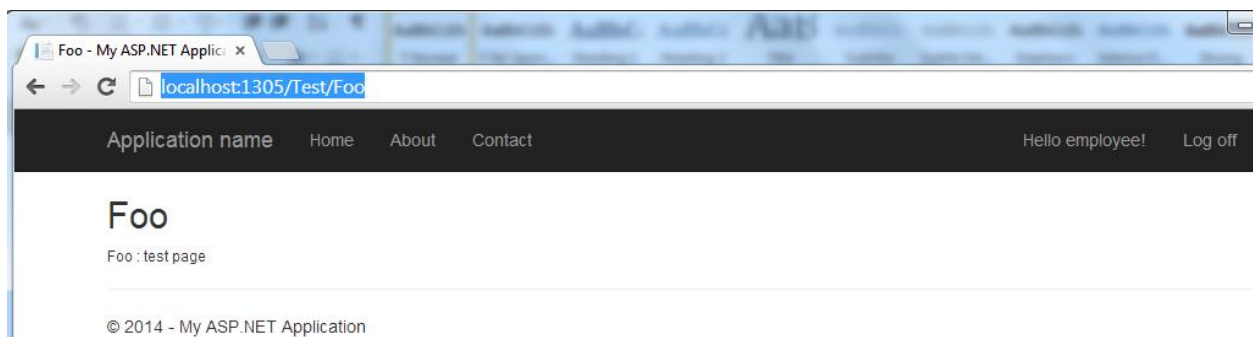
        [Authorize(Roles="employee")]
        // [AllowAnonymous]
        // 0 references
        public ActionResult Foo()
        {
            return View();
        }
    }
}
```

This should only allow users in employee role (employee & employee2) to view the result of Foo Action method.

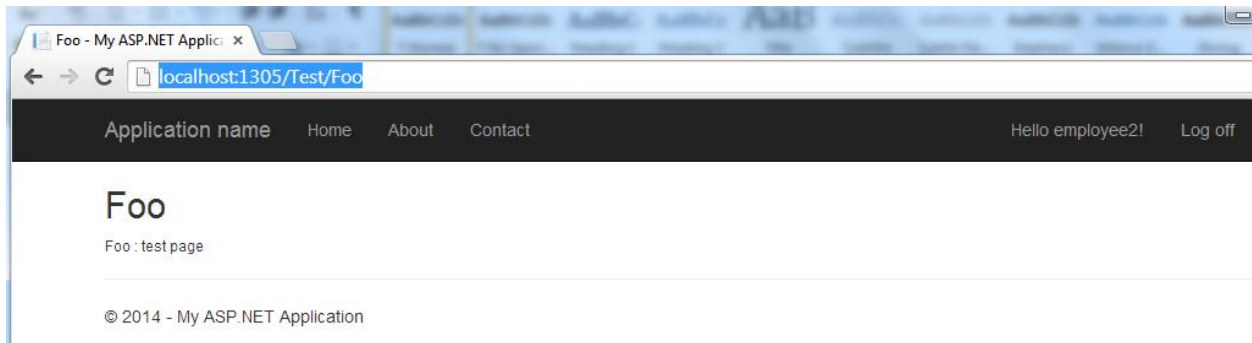
Run the application and login as employee user.



Navigate to Test/Foo as follows:



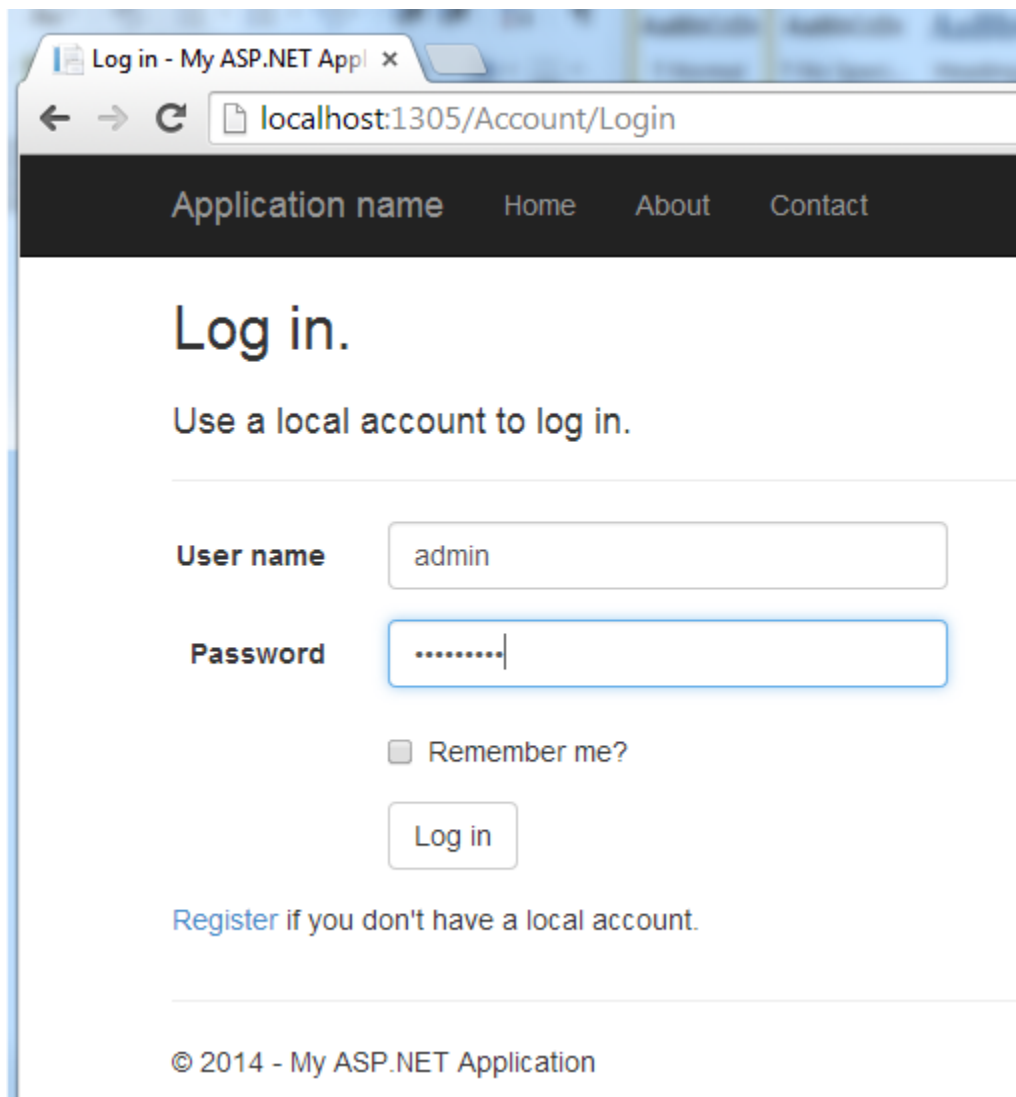
As expected this user can view this page. Repeat the test for employee2 user.



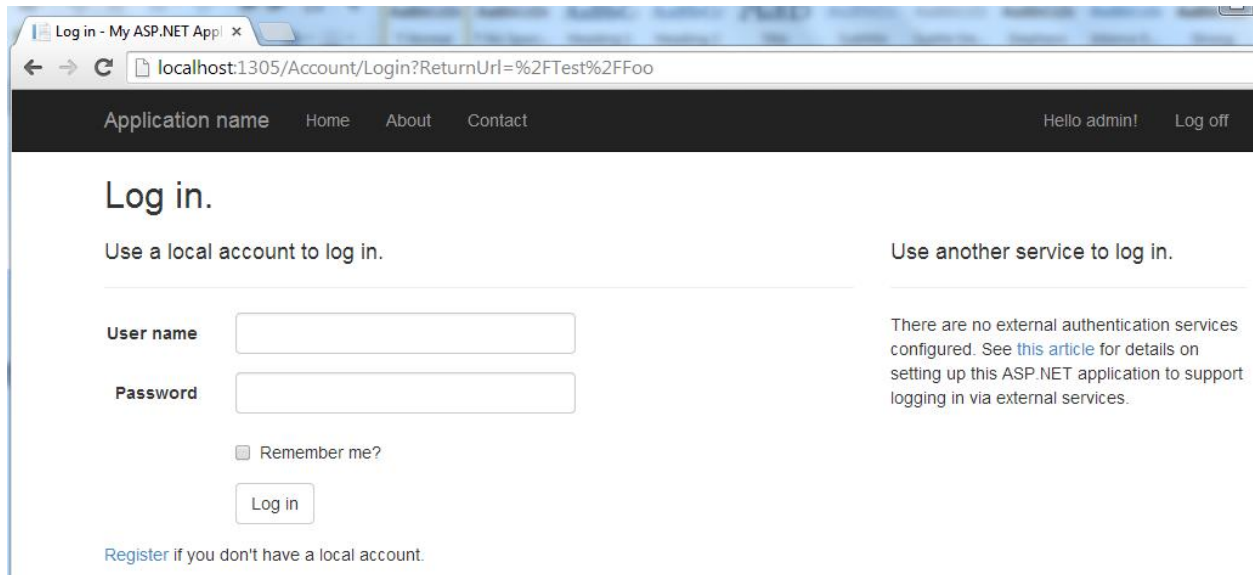
As expected, users in employee role can access the page.

Now let's test the admin user, which belongs to admin role.

Login as the admin user:



Navigate to Test/Foo.



The screenshot shows a web browser window with the title 'Log in - My ASP.NET App'. The address bar displays 'localhost:1305/Account/Login?ReturnUrl=%2FTest%2FFoo'. The page has a dark navigation bar with 'Application name', 'Home', 'About', and 'Contact' on the left, and 'Hello admin!' and 'Log off' on the right. The main content area is titled 'Log in.' and is split into two columns. The left column, 'Use a local account to log in.', contains a 'User name' field, a 'Password' field, a 'Remember me?' checkbox, and a 'Log in' button. Below this is a link: 'Register if you don't have a local account.' The right column, 'Use another service to log in.', contains a message: 'There are no external authentication services configured. See [this article](#) for details on setting up this ASP.NET application to support logging in via external services.'

As can be seen, the admin user is not allowed to view the page. This is as we expected, given we only authorized the employee role to access the page.