LAB MANUAL

Cloud Computing

Course Code: CSL-313

Department of Software Engineering
Bahria University Karachi Campus
13 National Stadium Road

Table of Contents

Preface

This Lab will discuss following things:

1. Learn to perform Cloud Computing tasks using a cloud supporting toolkit.
2. Understand the development, deployment and security of any cloud base application.
3. Demonstrate the working of workflows, security is the main task.
4. To obtain Practical Experience Working with all real cloud base application.

Emphasize hands-on experience working with .net core and other

SE Department, Bahria University Karachi *Engr. Muhammad Faisal and Engr. Ramsha Mashood*

Lab No. 1

Introduction to Cloud Computing

LAB # 01

Introduction to Cloud Computing

What is Cloud Computing?

Cloud computing is all about using many computer systems that are "out there", whose physical address we do not even know. The "many" can range from just a few to thousands. Each of these systems may have TB of RAM, peta-bytes of hard disk space.

- TBD High Performance Computing HPC.
- High throughput computing
- Computing aimed at "beating" humans.
- Playing chess, Jeopardy, Voice recognition. Speech recognitions. Dictation.
- Elapsed time versus Total computing time.
- By 2014, cloud computing will be as common as mobile phones are today.

Distributed Computing Models

Remote Computing

We can run computations on remote machines. The remote machine may not have our files, or our programs. It becomes our task to transport these.

Process Migration

Just like people migrate, we can think of processes migrating from one machine to another. Just like humans cannot migrate to Mars (yet?), processes can only (as of 2012) migrate from one Linux machine to another highly similar machine. Process migration involves "check-points", creating a frozen image of all their address spaces, and transporting the image to a remote machine and "thawing" it there.

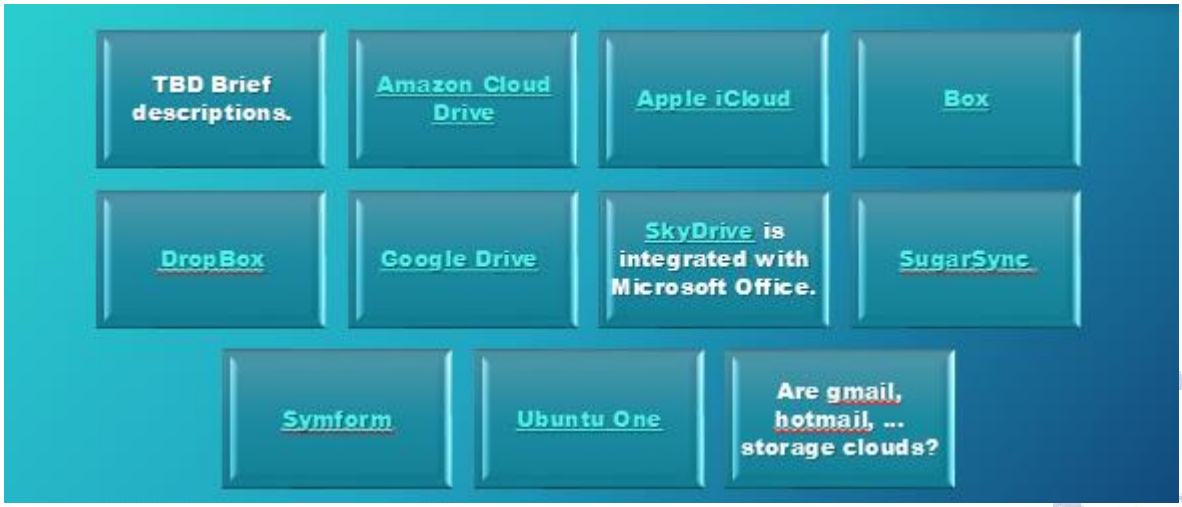
Using Idle Computers

In most departmental offices, there are probably dozens of PCs running screen savers. It is possible to put such idling machines to productive computing. E.g., a package known as Condor can be installed on Windows or Linux PCs that takes a description of jobs to be run and runs them on idle PCs as they become available. As soon as the "owner" of the idle PC begins an activity, the guest computation gracefully vacates and migrates to another. Condor is a standard package in Ubuntu: apt-get install condor

Cluster Computing

A group of machines can be clustered together so that they an "awareness" of each other (cf. Star Trek Borgs). The Top Ten of the world's most powerful supercomputers are Linux clusters (visit http://www.top500.org/).

Well-Known Storage Clouds



Time Boxing

Activity Name	Activity Time	Total Time
Login Systems + Setting up Cloud account	3 mints + 5 mints	8 mints
Walk through Theory & Tasks	60 mints	60 mints
Implement Tasks	80 mints	80 mints
Evaluation Time	30 mints	30 mints
	Total Duration	178 mints

Objectives

After completing this lab the student should be able to:

- Clearly understand the purpose and benefits that **Cloud Computing** has to offer.
- Understand the concept of **Cloud Computing**.
- Use and learn **Cloud Computing** need.

Lab Tasks/Practical Work

1. Setting up First cloud account using Microsoft azure

Lab No. 2

Implementing Microsoft core application using ASP.Net MVC Frame work

LAB # 02

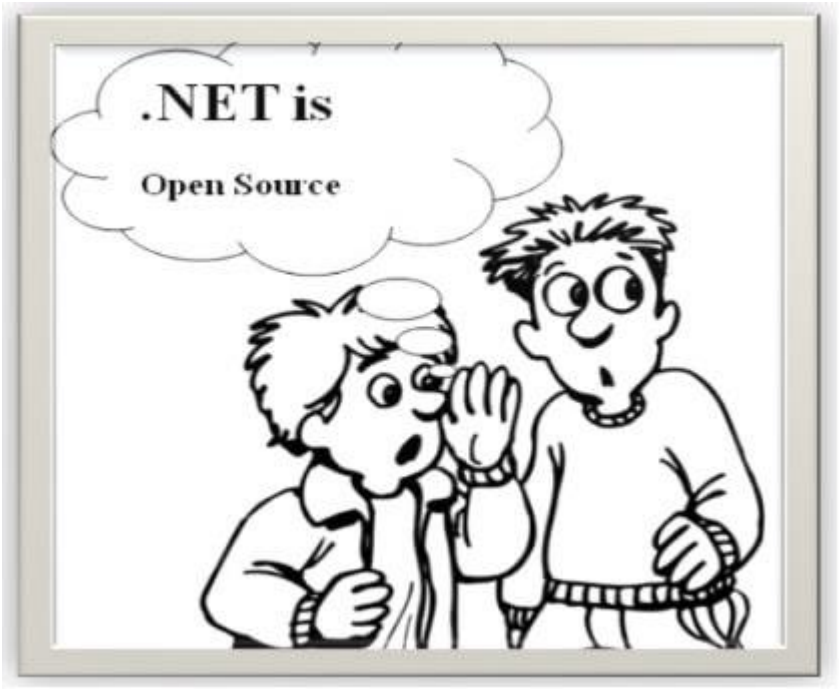
Implementing Microsoft core application using ASP.Net MVC Frame work

What is .NET Core?

.NET Core is a general-purpose, modular, cross-platform and open-source implementation of the .NET Platform. It contains many of the same APIs as the .NET Framework (but .NET Core is a smaller set) and includes runtime, framework, compiler and tools components that support a variety of operating systems and chip targets.

The .NET Core implementation was primarily driven by the ASP.NET Core workloads but also by the need and desire to have a more modern runtime. It can be used in the device, cloud, and embedded/IoT scenarios.

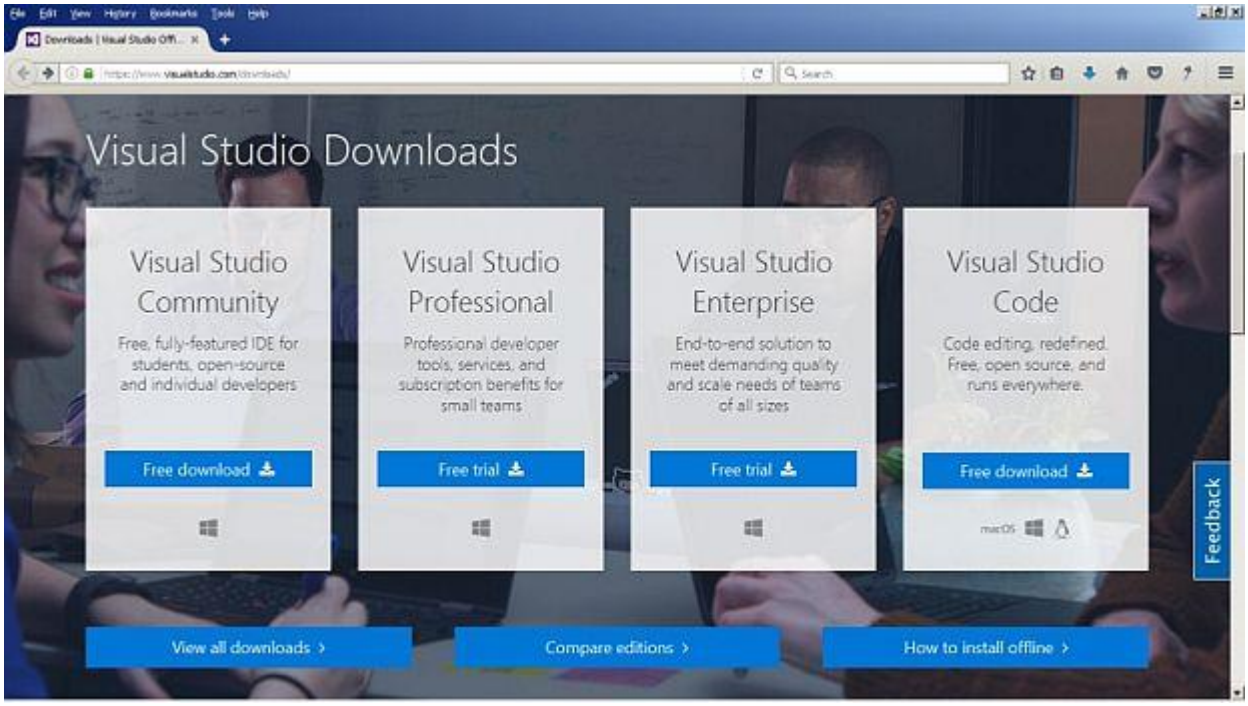
Referenced from https://msdn.microsoft.com/en-us/library/dn878908(v=vs.110).aspx



Tools Required

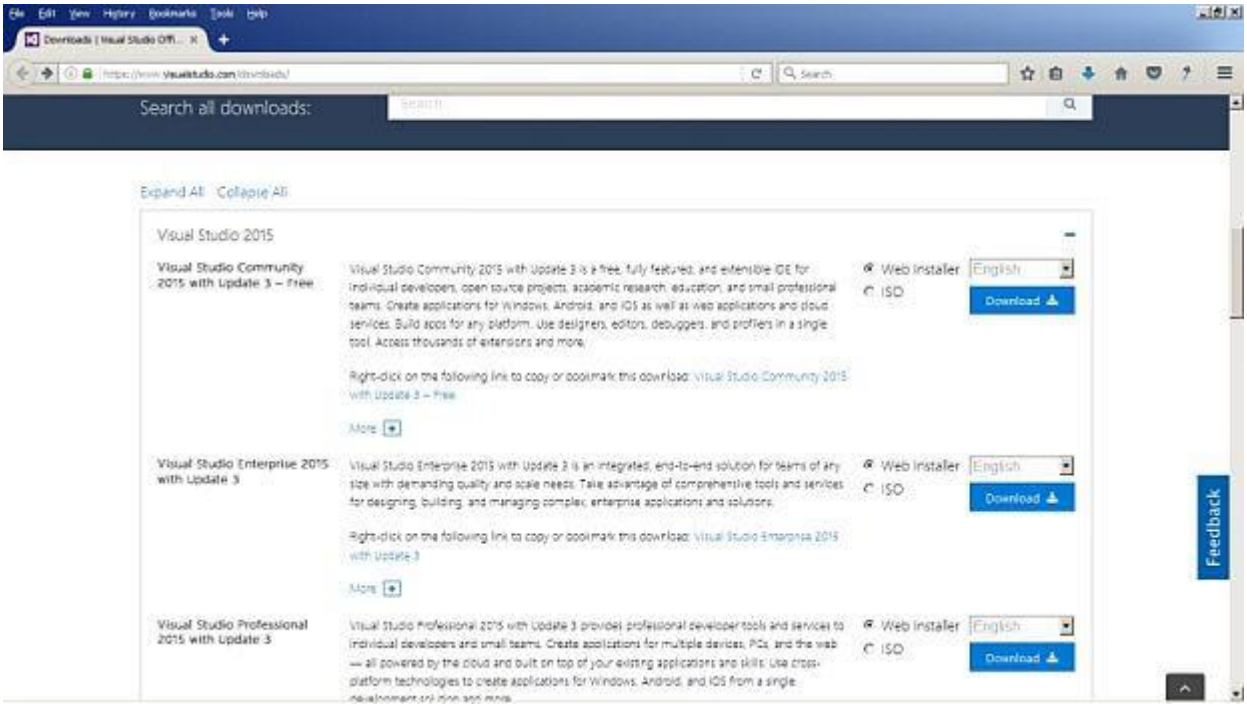
1. *Visual studio 2015 with Update 3*

URL for downloading Visual studio 2015 https://www.visualstudio.com/downloads/



If you already have Visual Studio 2015 installed just download Visual Studio 2015 Update 3.

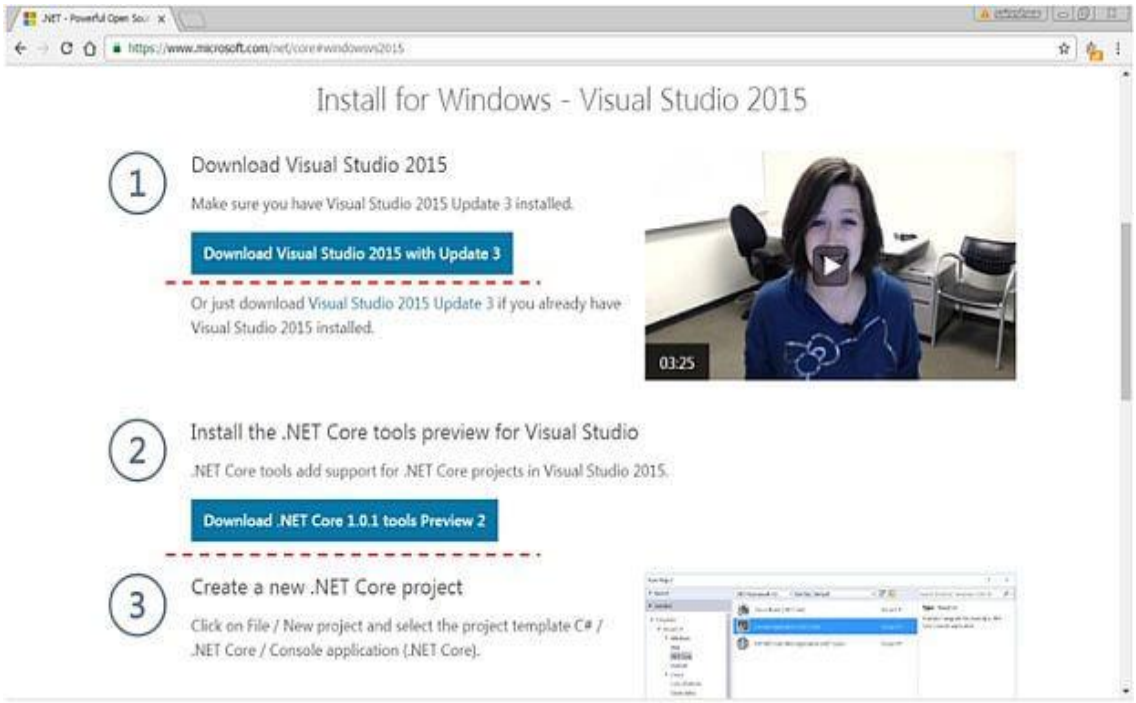
URL for downloading Visual Studio 2015 Update 3
- https://go.microsoft.com/fwlink/?LinkId=691129

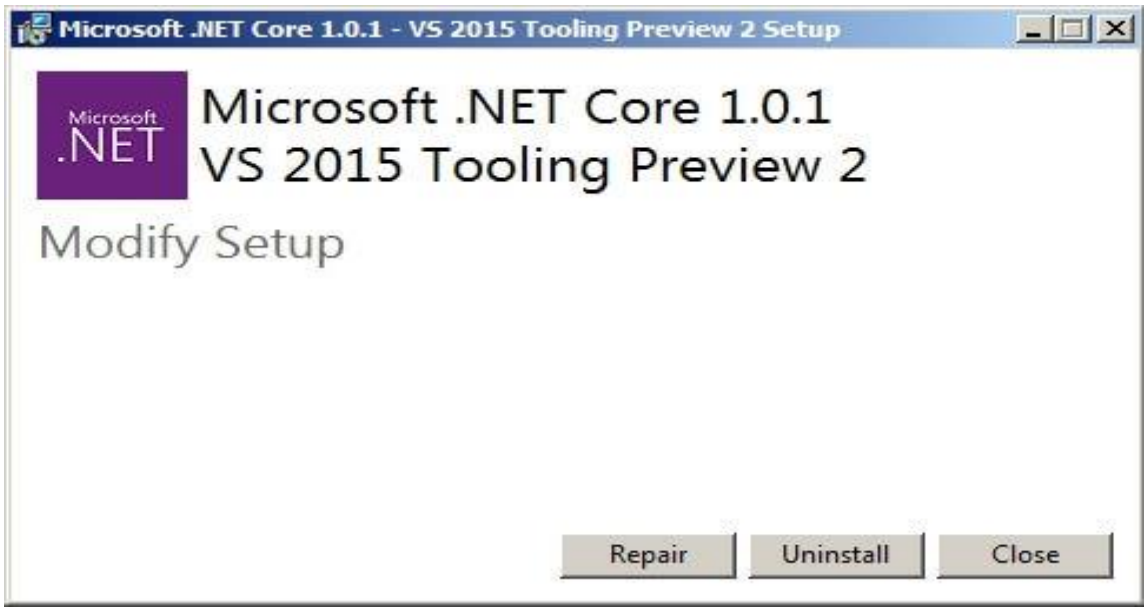


2. *NET Core 1.0.1 framework*

   For downloading .NET Core 1.0.1 tools Preview 2 framework.

   URL https://www.microsoft.com/net/core#windowsvs2015

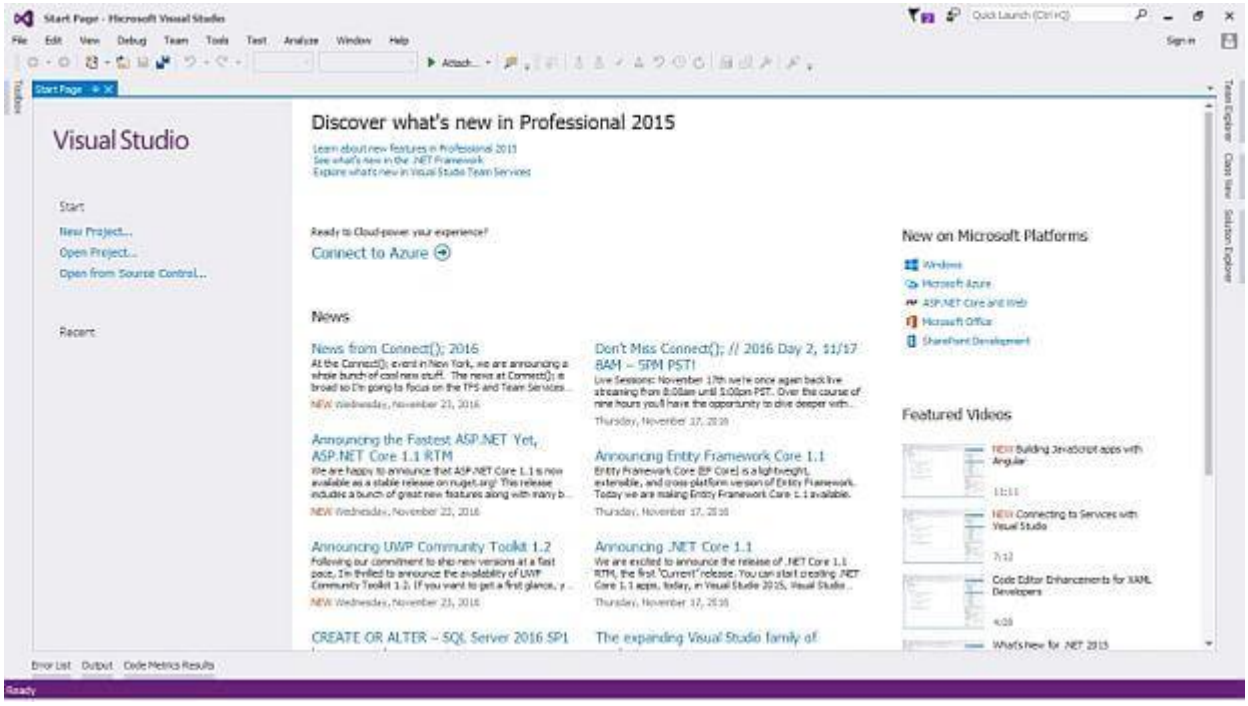




After installing visual studio and framework you are ready to work with New. NET Core 1.0.1.

Getting Started Creating .Net Core web application

Open visual studio 2015 IDE.



Start page of Visual studio IDE

Before moving forward let's confirm we have updated new visual studio update properly.

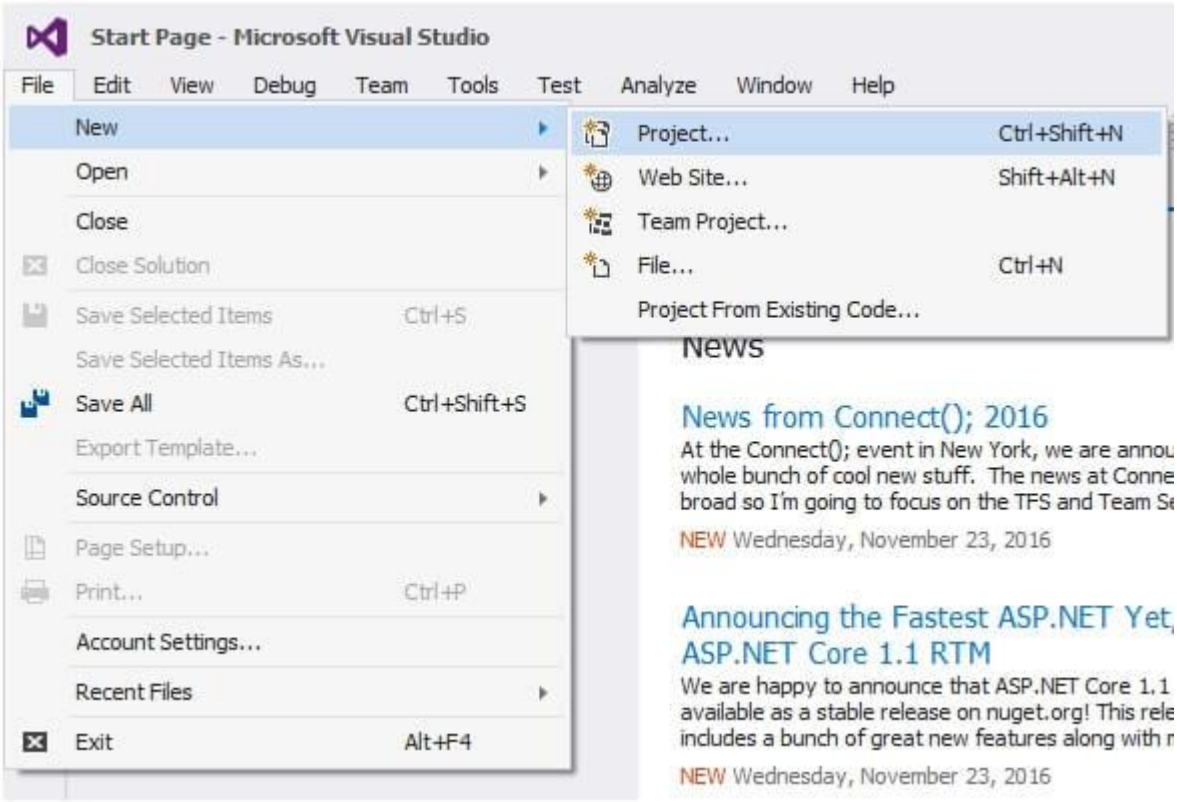
For doing that choose Help menu from start page of Visual studio IDE and inside that menu choose About Microsoft visual studio.

An About Microsoft Visual Studio dialog will pop up as shown below.

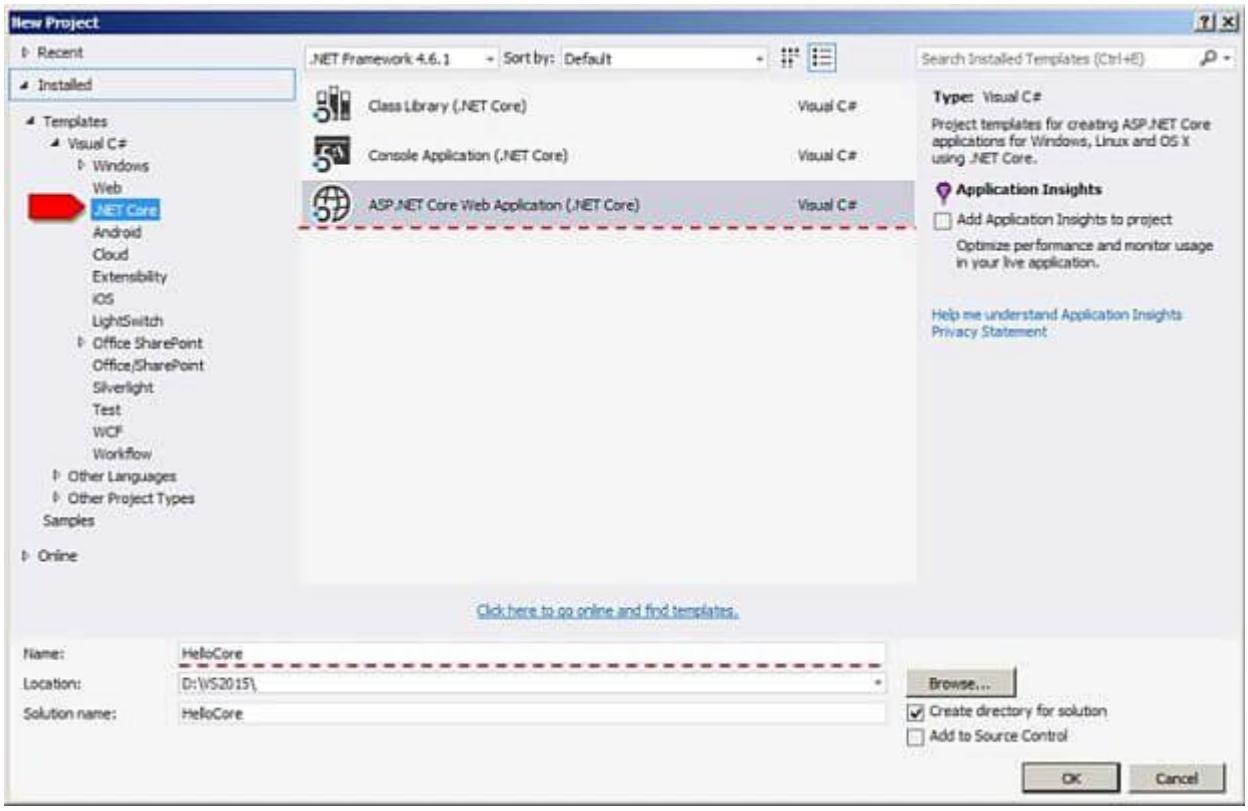
Check Visual studio installed version

We have confirmed update is proper now let’s create a web application.

From Visual IDE choose File Menu inside that choose New, then choose Project.



Add Project
After choosing Project menu a new dialog will pop up with name “New Project” as shown below.



Naming Project

First, think to look in new project dialog in .Net framework Version dropdown, it must be above Net framework 4.

In this dialog just have a look on left pane [Templates], inside that you will see new option [.Net Core] just choose [.Net Core] after that in middle pane you will find all project templates related to .Net Core from this templates we are going to choose “ASP.NET Core Web Application (.Net Core)” template because we are going to learn how to create a web application using .Net Core.

Next step after choosing template is to Name Project here we are creating the first application with .Net Core that's why I am going to name this project as “HelloCore” and finally click on OK button to create a project.

After clicking on the OK button I will pop up a new dialog for choosing templates.

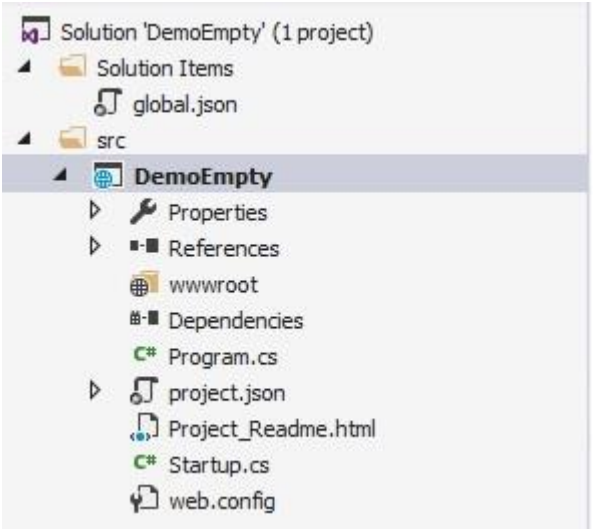
Choosing Templates

In this dialog, we have 3 templates to choose from.

1. Empty
2. Web API
3. Web Application

Empty Template

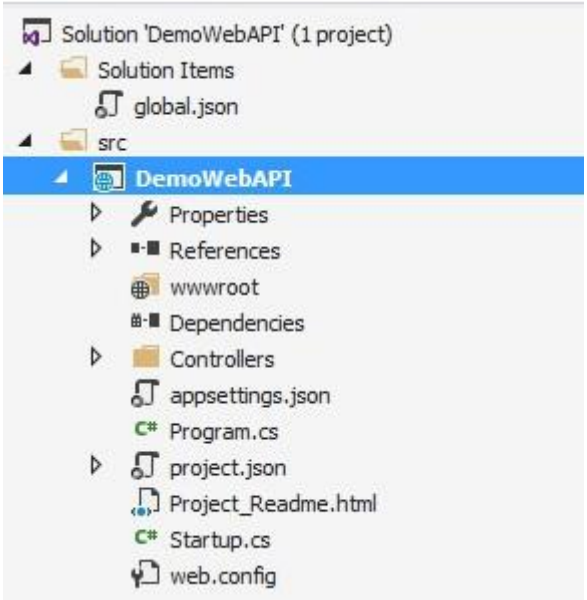
The Empty template Contains only empty wwwroot folder along with basic configuration files. In case if we select Empty Template for project our folder structure looks as shown below



Empty template

WEBAPI Template

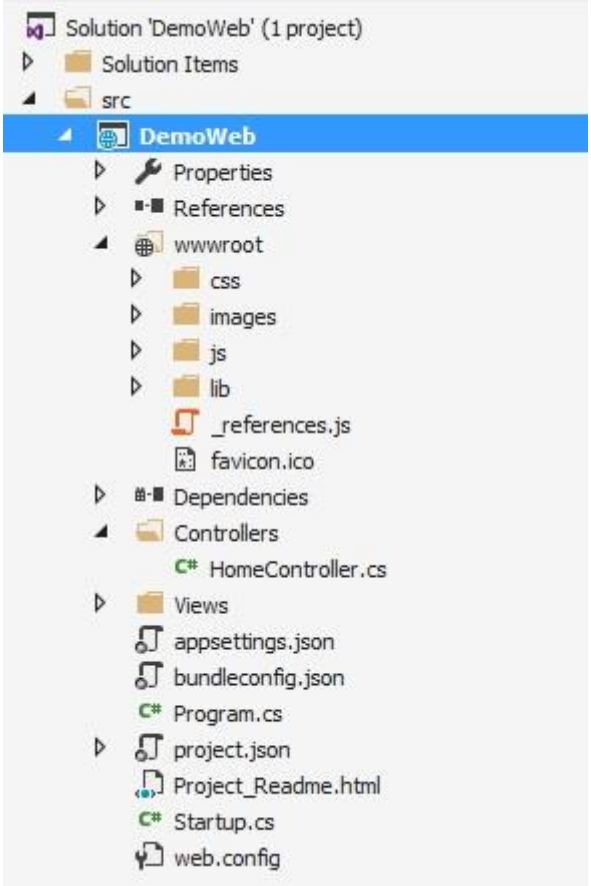
The WEBAPI template contains an empty wwwroot folder and along with that, it has Controllers folder with ValuesController in it along with basic configuration files.



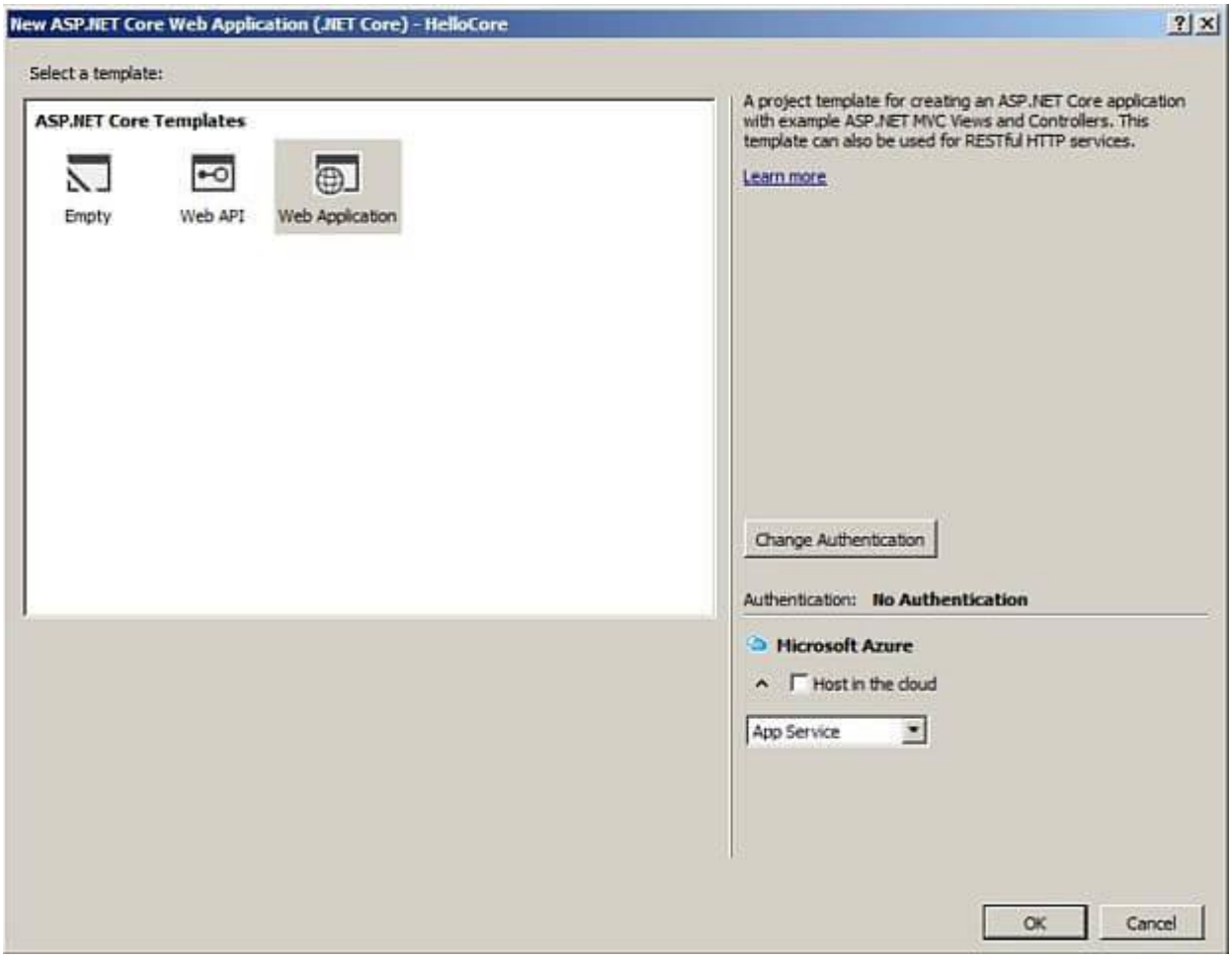
WEBAPI template

Web Application Template

The Web Application template contain a wwwroot folder and along with that it has Controllers folder with default HomeController and Views folder contains Home and Shared folder , the Home folder contains Views related to Home Controller and Shared Folder has Layout and error view in it.



Web Application template

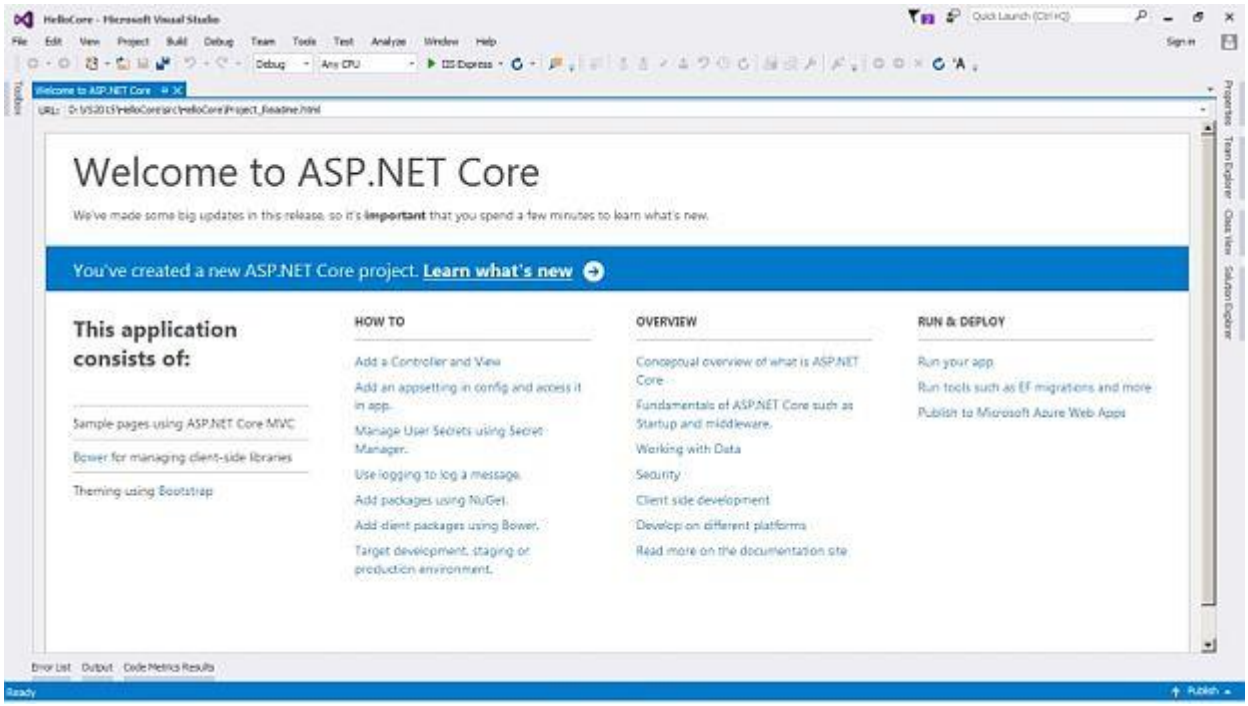


Choosing Project template

We want to create a web application, and for that we are going to choose web application template and click on OK button to create a project.

After clicking on the OK button, it will create a project and show Project_Readme.html file as shown below.

This file has a useful link to learn .Net core basics.

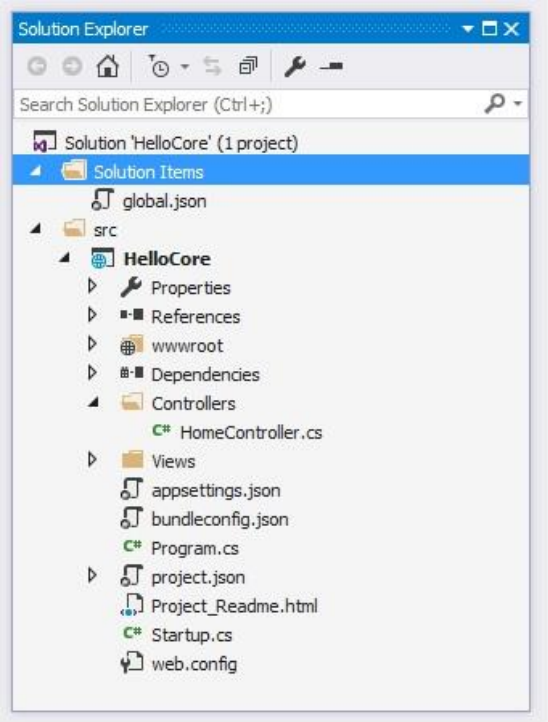


Project Readme HTML file

Next step -- let's have a look at Project structure which is created.

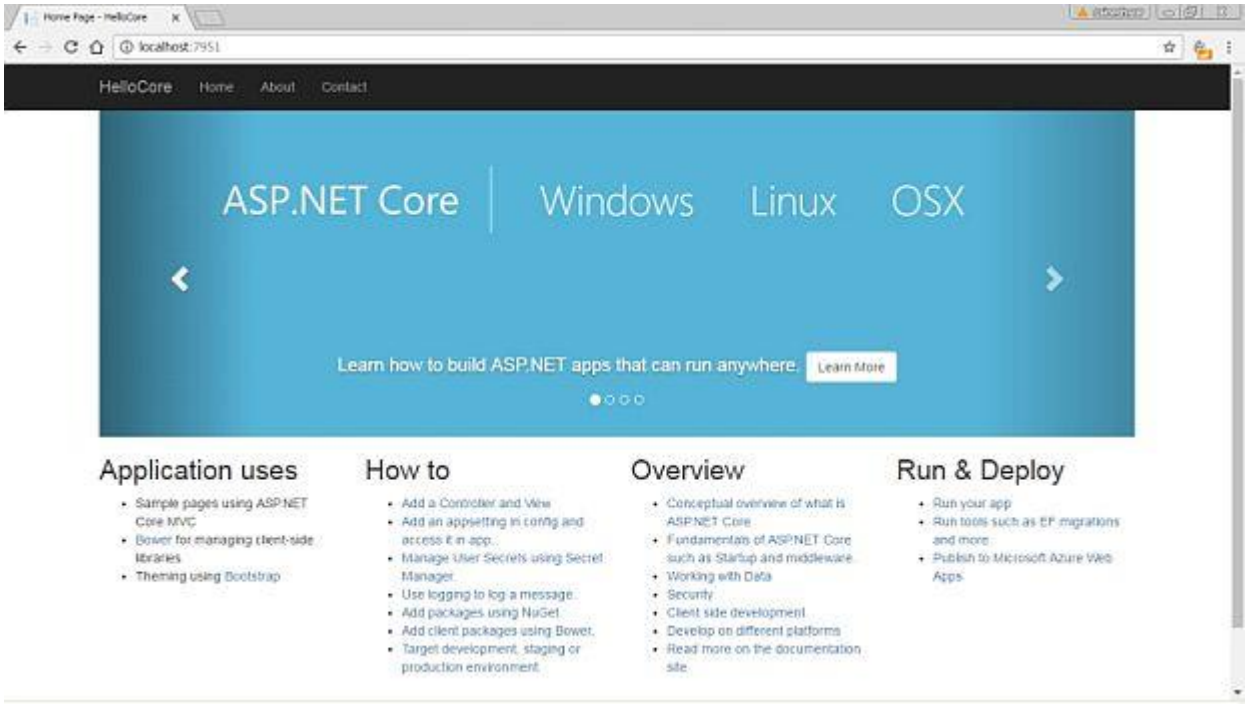
Project Structure

This is project structure which is created after choosing web application template.



Project View after creating a new project.

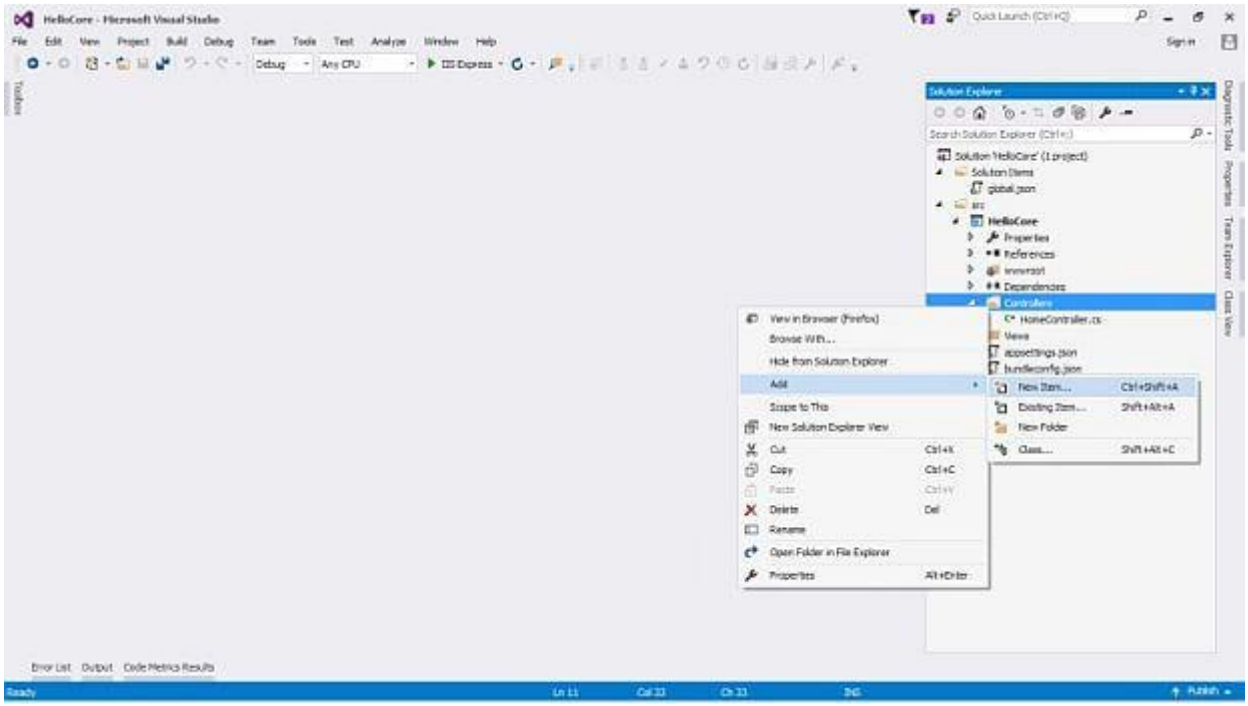
Now just save the application and run.



The first view of .Net core application

Adding Controller

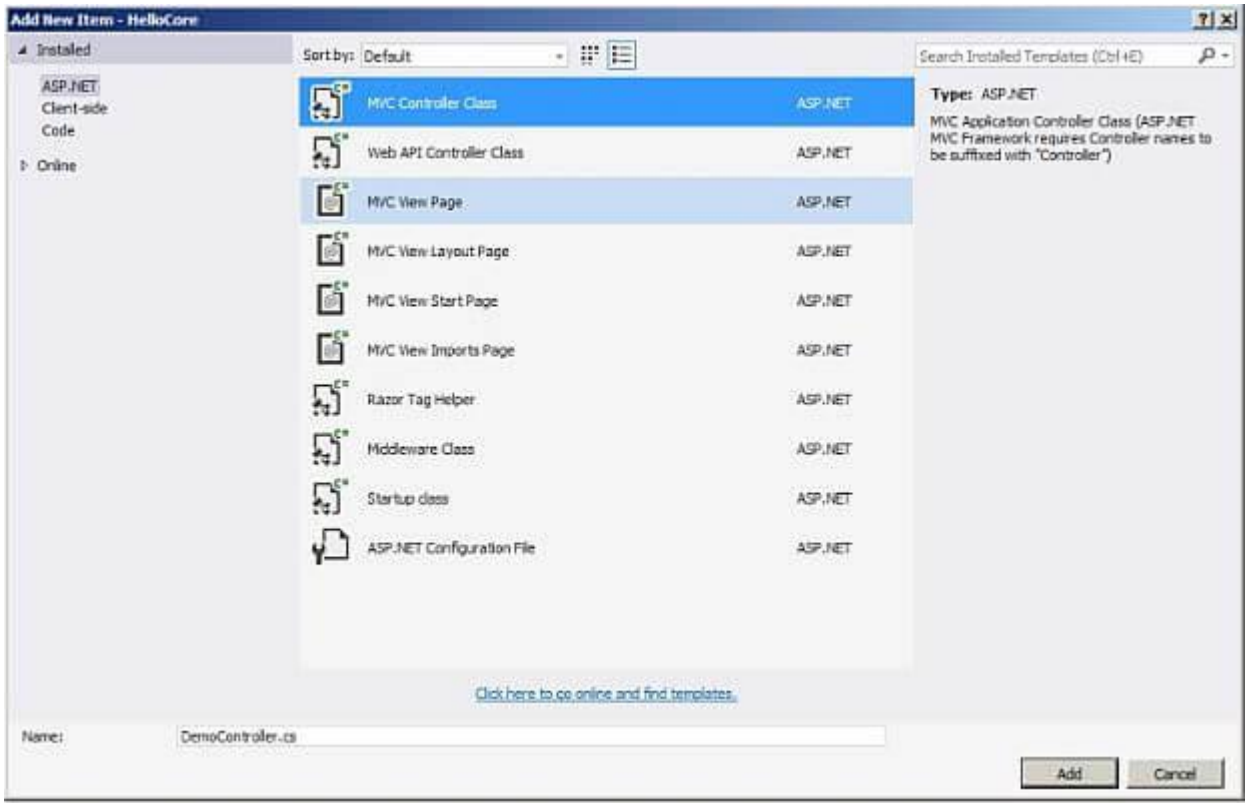
For adding controller just right click on Controller folder then select Add -- inside that select New Item.



Steps for Adding Controller

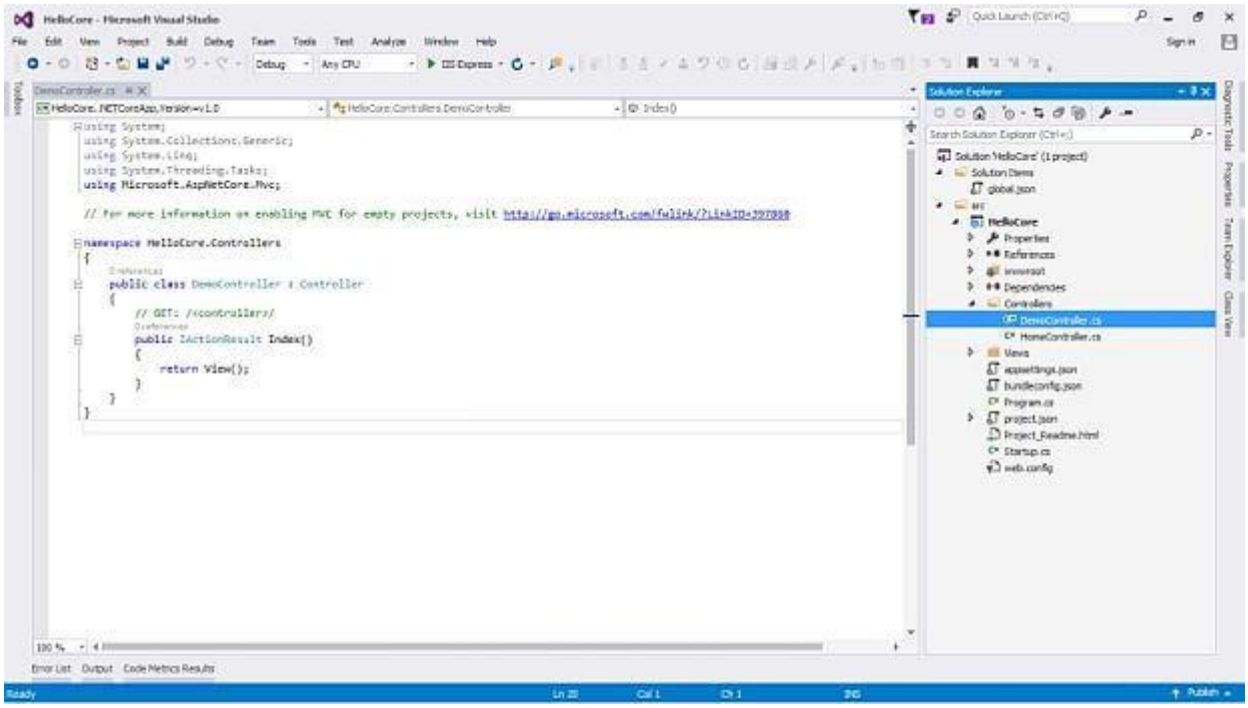
Note

If you compare this with MVC 4 or MVC 5 you will see something different -in the process of adding a controller in MVC 4 and 5 we have controller option directly to choose from -- here we do not have it.
After selecting New Item a new dialog of Add New Item will pop up.
Inside that just choose “MVC Controller Class” then name Controller as DemoController and Click on Add button to create Controller.



Choosing Controller class and Naming Controller [Demo]

After we have clicked on Add button it has created DemoController as shown in below view.

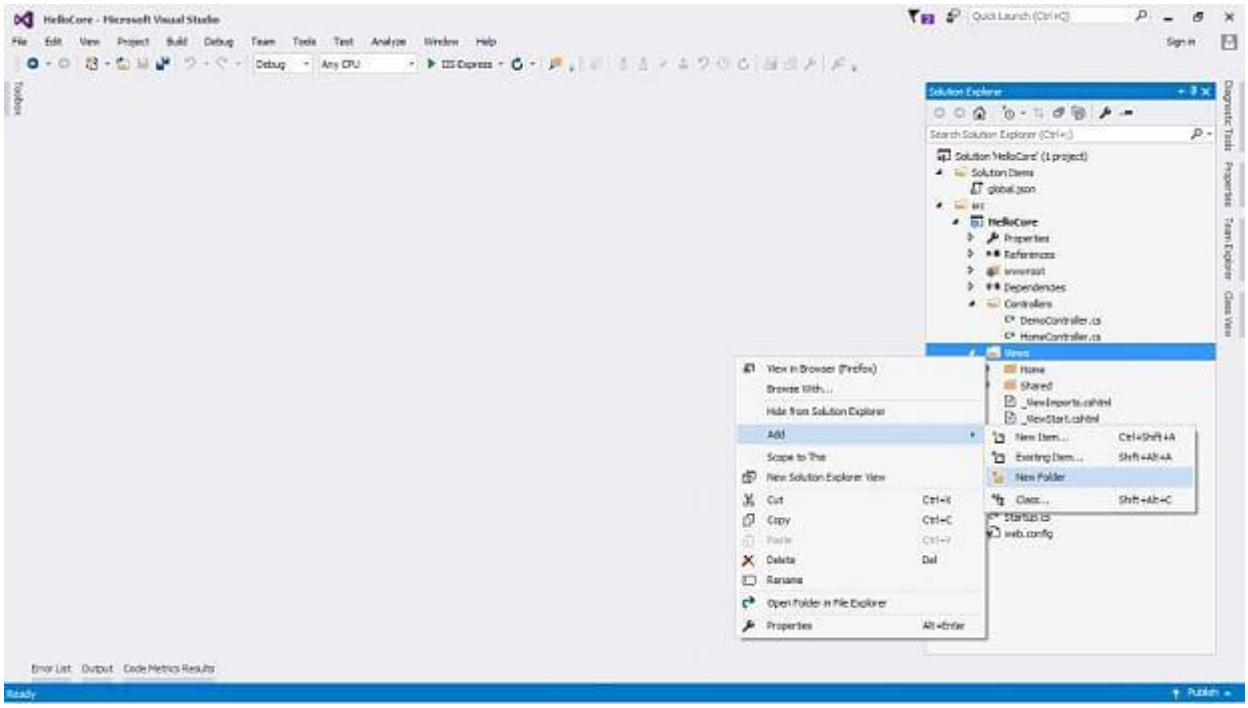


Code snippet of Demo Controller

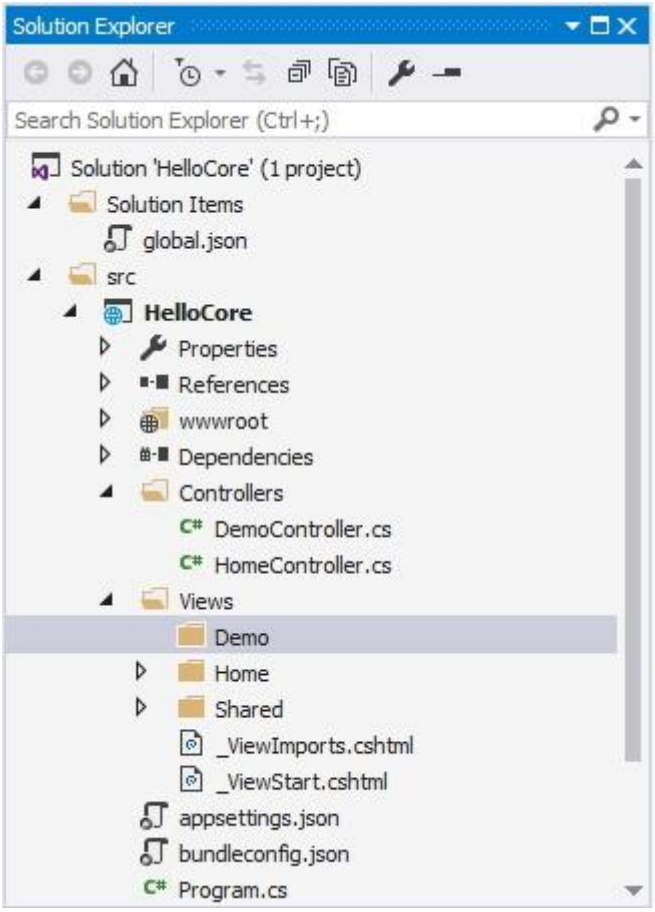
Adding View

For adding View, we cannot use our old style just right click inside action Method then choose Add View.

In .Net Core for adding View just Right click on the Views folder, and then Add à New Folder and name the folder “Demo”



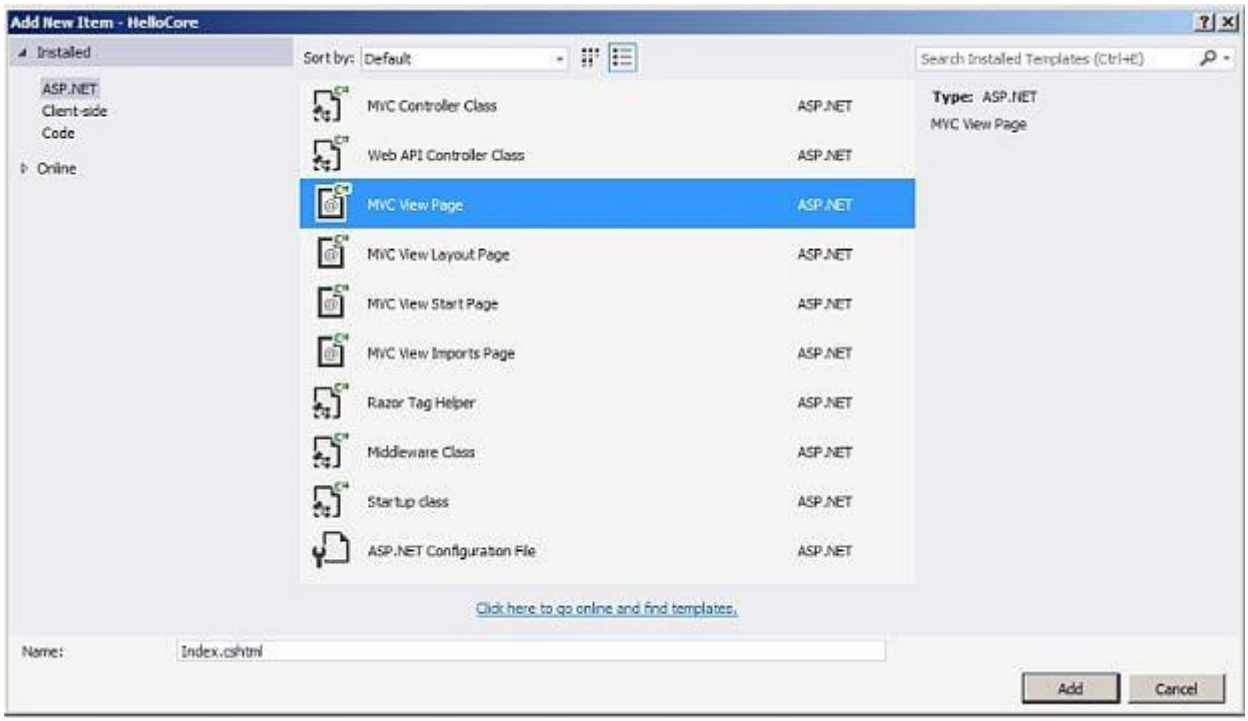
Steps for Adding “Demo “Folder



Project structure after adding Demo folder

After adding Demo folder now we are going to add View inside this folder.

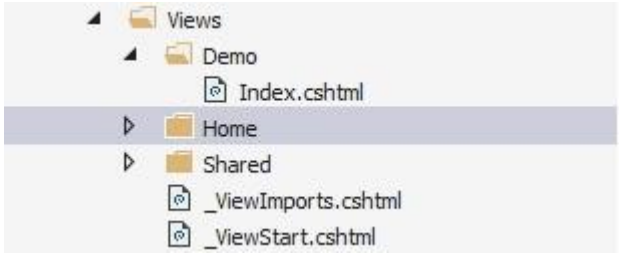
For adding View Right click on the Demo folder which we have added and then select Add -- inside that select New Item then a new dialog with Name of Add New Item will pop up.



Choosing MVC View page template and naming it.

From Add New Item dialogues just choose “MVC View Page” for adding View, then next step is to giving name to View, the View name must be the name same as Action method name, we are going name it as “Index” [“Index.cshtml”] and click on Add button to add View.

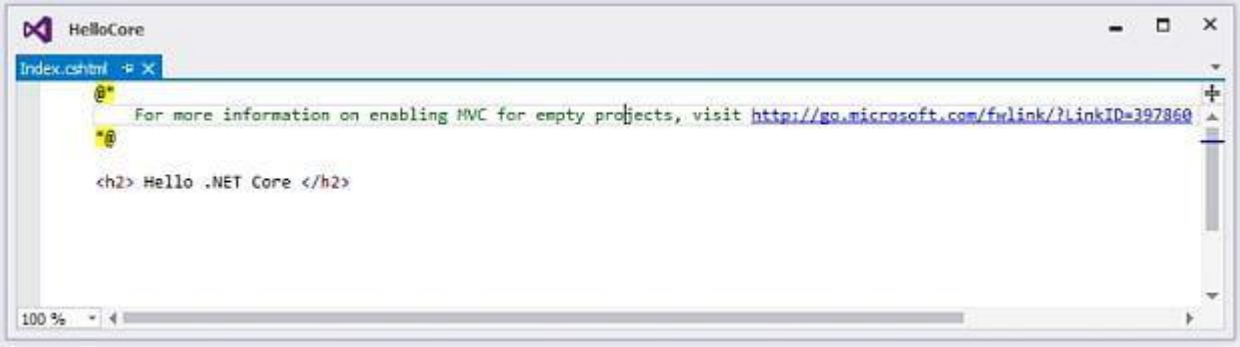
Project View after adding Index View



Project View after adding Index View

After generating Index View it is blank I have just added simple text to it.

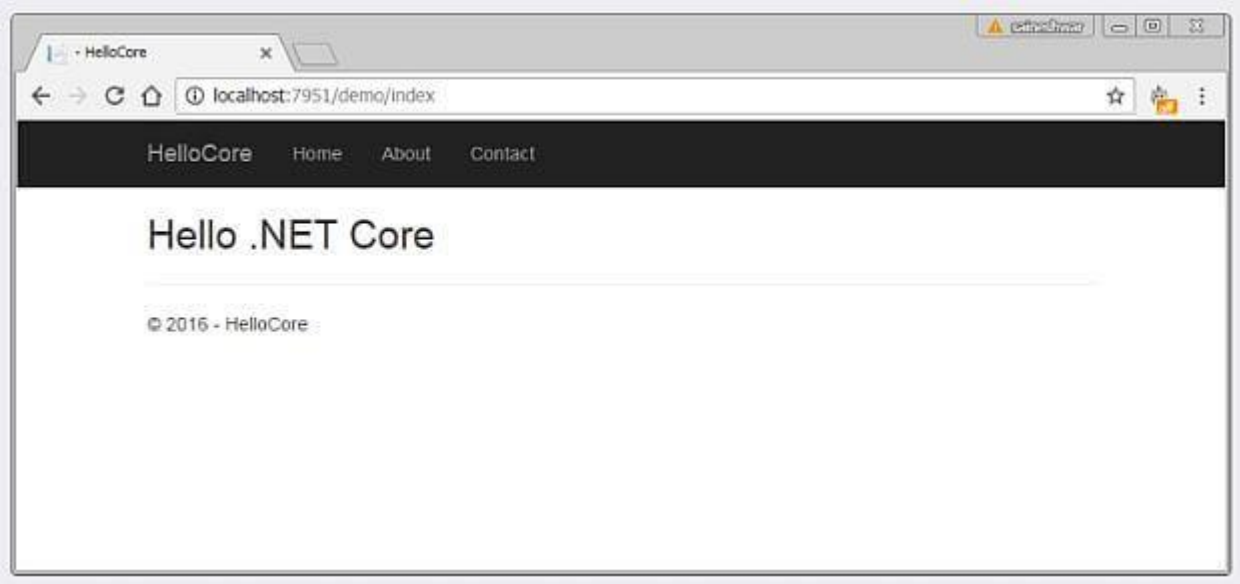
Index.cshtml View



Index.cshtml View

Now save and run the application.

To access newly created View URL http://localhost:####/demo/index



Time Boxing

Activity Name	Activity Time	Total Time
Login Systems + Setting up Visual studio Environment	3 mints + 5 mints	8 mints
Walk through Theory & Tasks	60 mints	60 mints
Implement Tasks	80 mints	80 mints
Evaluation Time	30 mints	30 mints
	Total Duration	178 mints

Objectives

After completing this lab the student should be able to:

- Clearly understand the purpose and benefits that Cloud Computing has to offer.
- Understand the concept of Cloud Computing.
- Build a basic Cloud Computing base application.
- Use and learn Cloud Computing need.

Lab Tasks/Practical Work

1. Create login and registration form using simple MVC net core
2. Create feedback form using simple mvc net core

Lab No. 3

MVC Entity Framework and Query data by using Entity Framework Core.

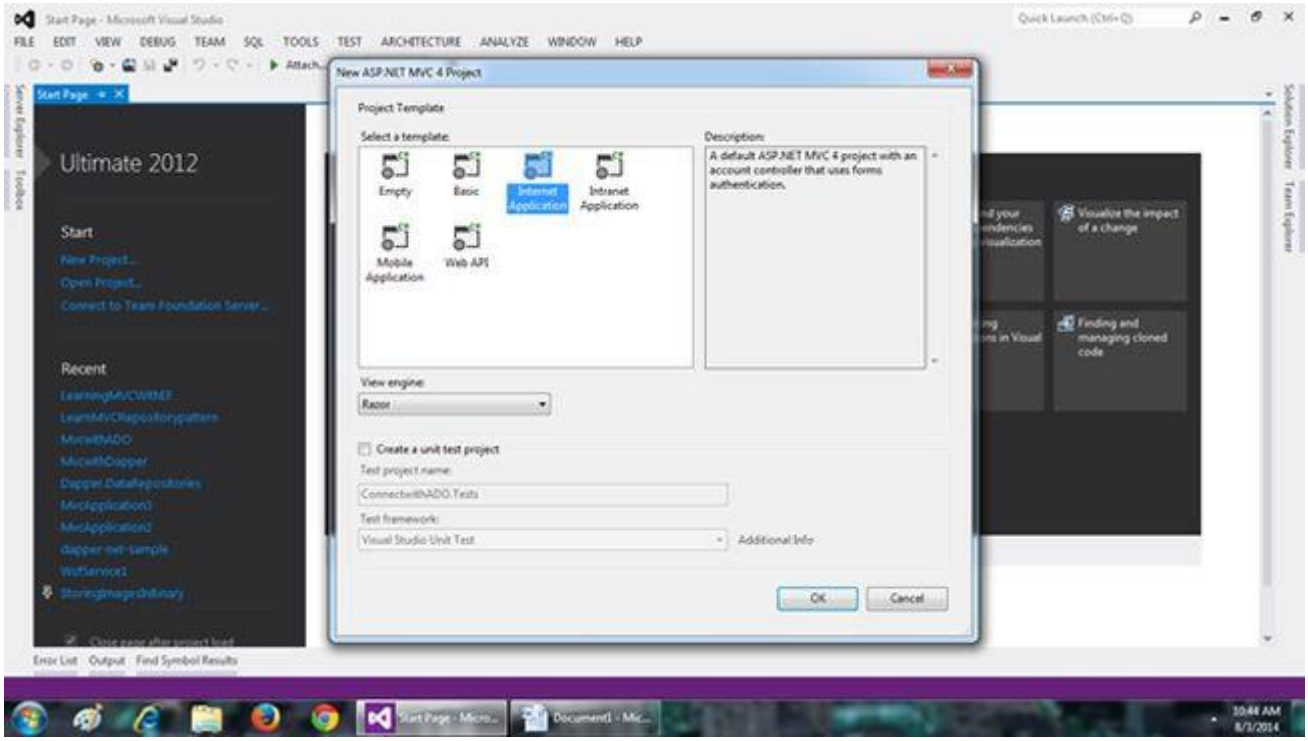
LAB # 03

MVC Entity Framework and Query data by using Entity Framework Core.

Step 1: Create a new project; choose ASP.NET MVC 4 Web Application (Visual C#).

And I am naming it ConnectwithADO and Click OK.

A new wizard will pop-up; from that just select Internet Application and View Engine choose Razor.

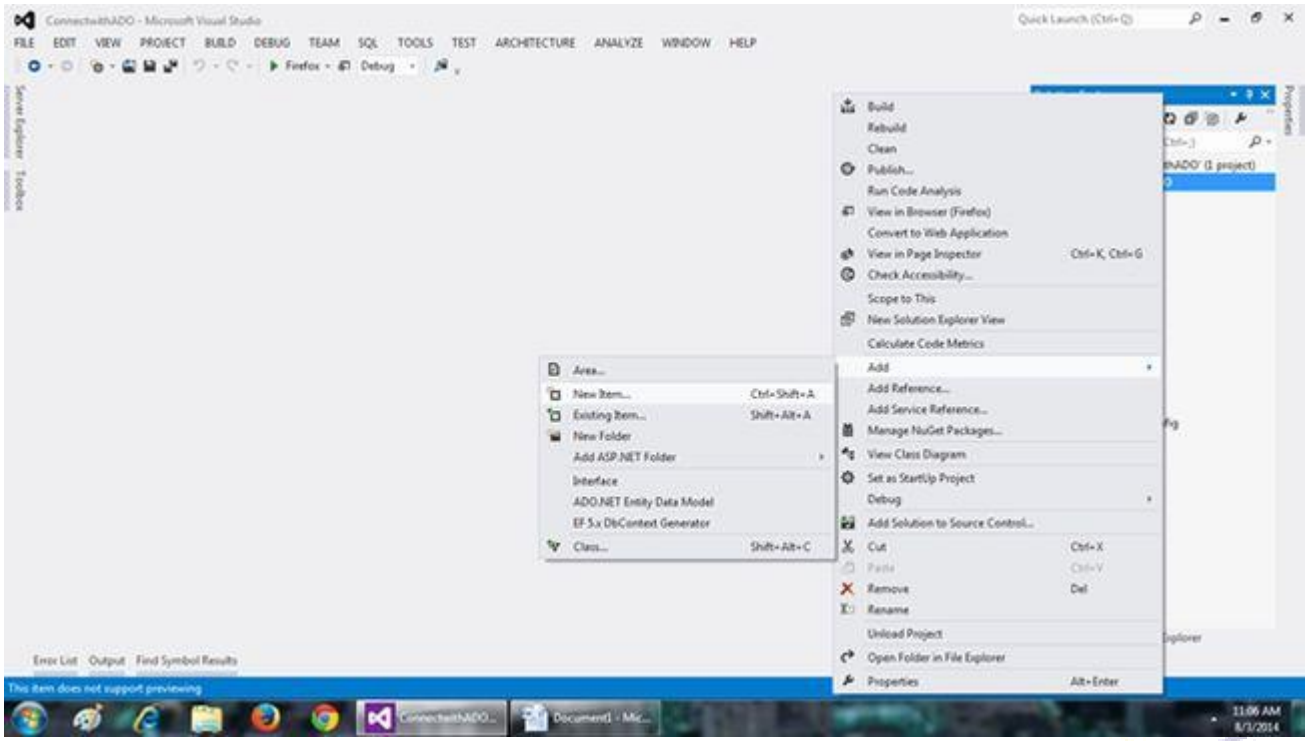


And after clicking OK, a default Home Controller screen will be visible.

Step 2: Adding ADO.NET Entity Framework.

Open Solution Explorer then right-click on the project (ConnectwithADO).

From the list choose Add and from the sub-list choose New Item.



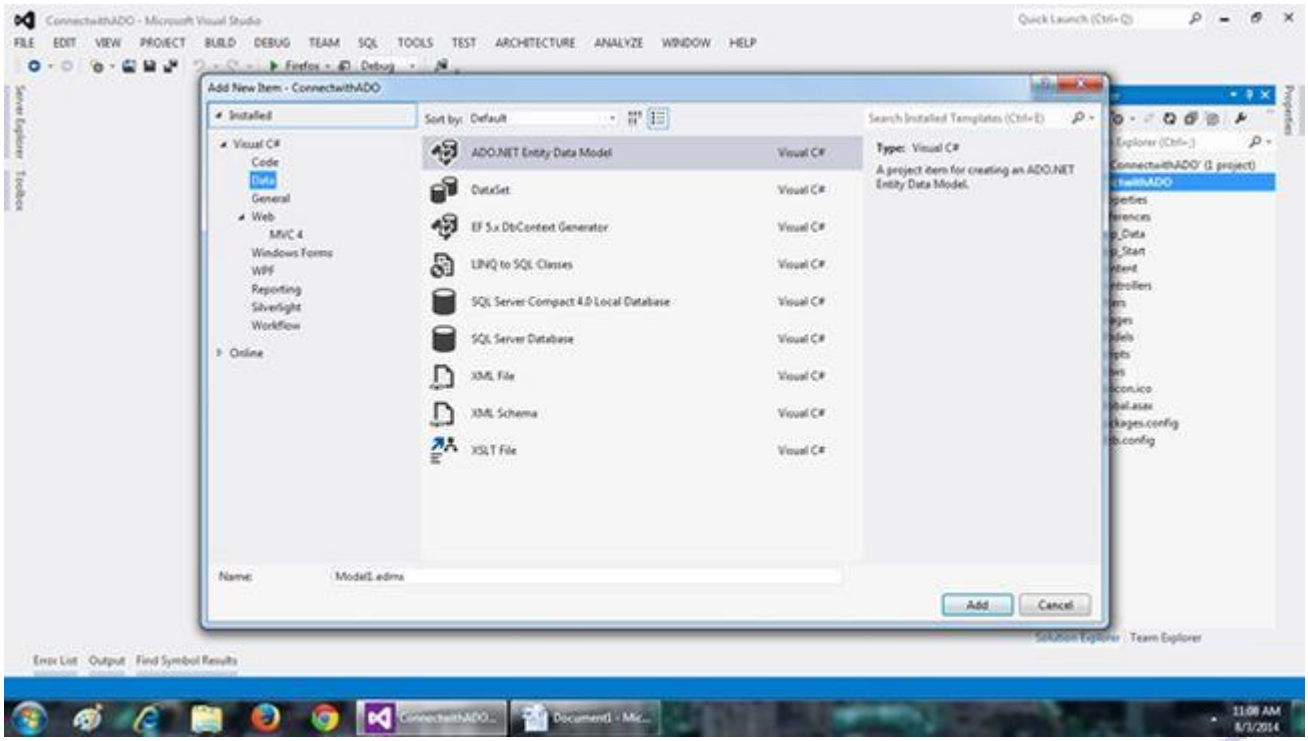
After clicking New Item a new Wizard will pop-up.

From the left type Visual C# to select the data.

On Select Data you will see ADO.NET Entity Data Model and another item related to Data.

In that select ADO.NET Entity Data Model and click Add.

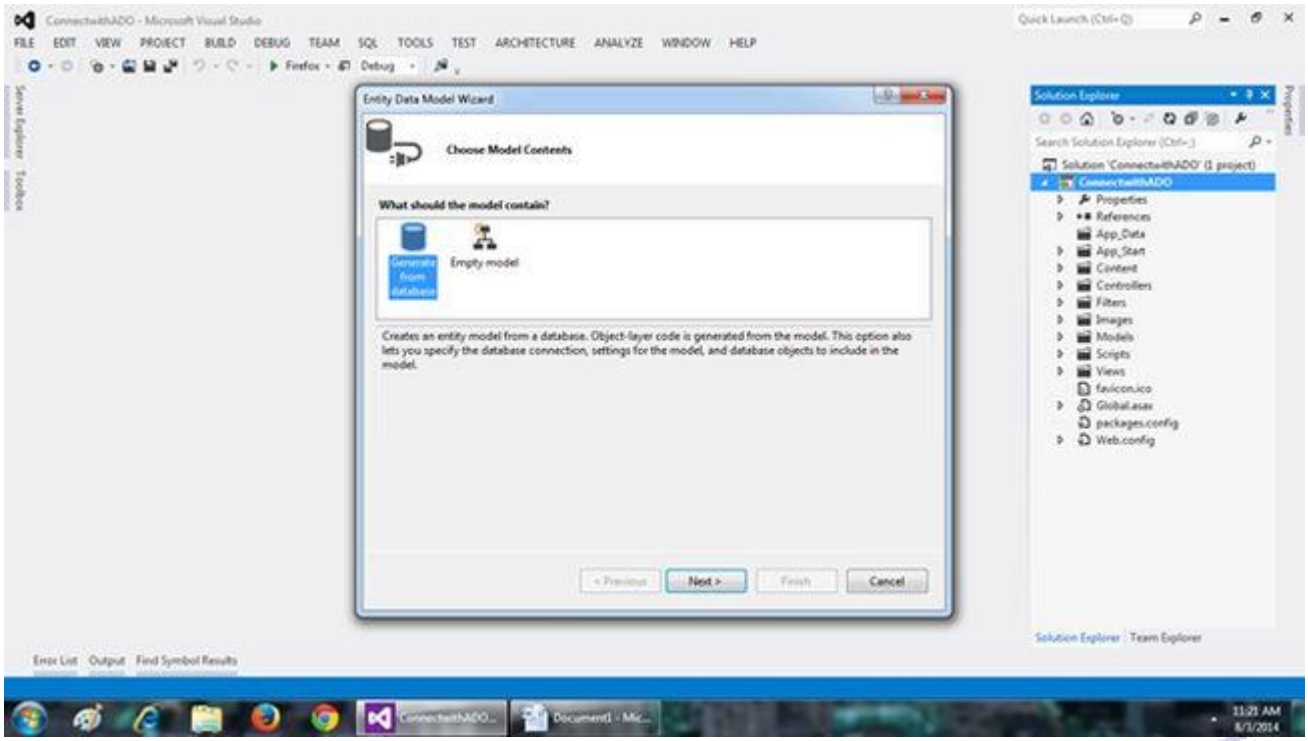
I am not changing the name of it for the Demo.



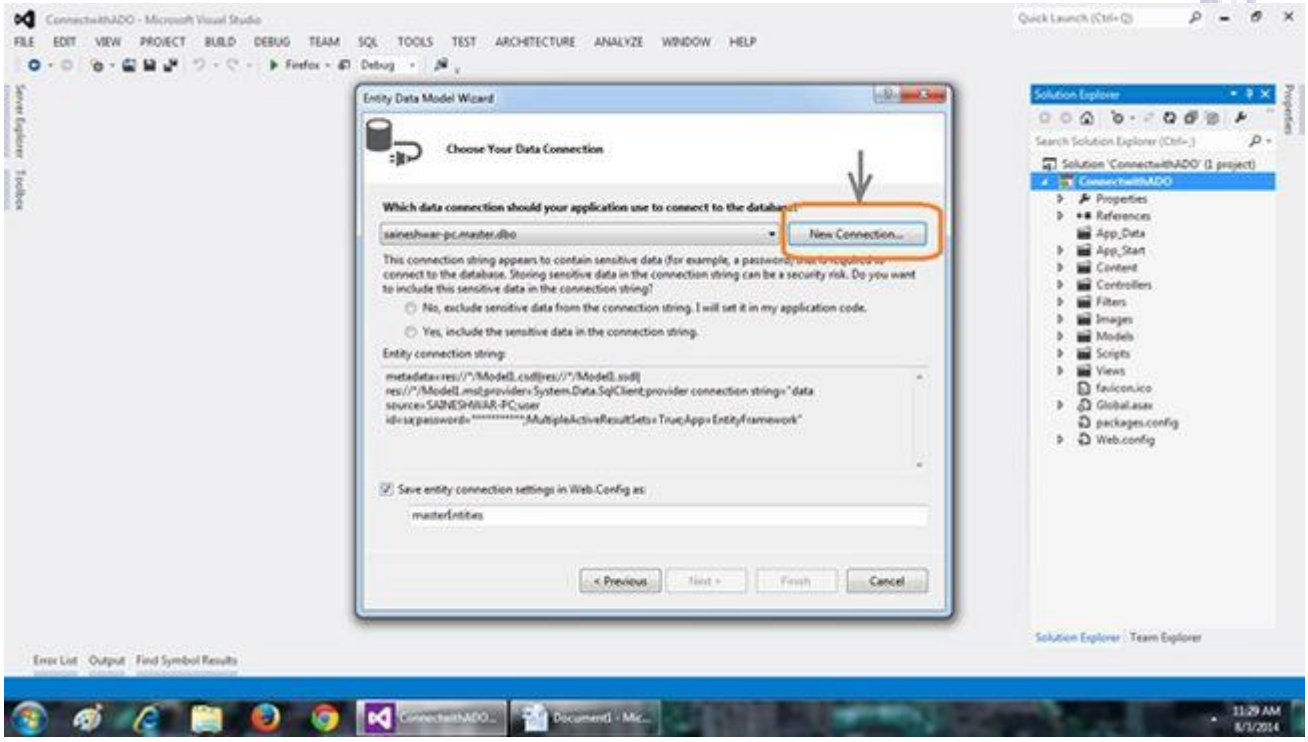
Step 3: Configuring ADO.NET Entity Data Model.

As soon as you click on Add a new wizard of Entity Data Model will appear.

I will choose the Generate from database option from these options and click on the Next button.

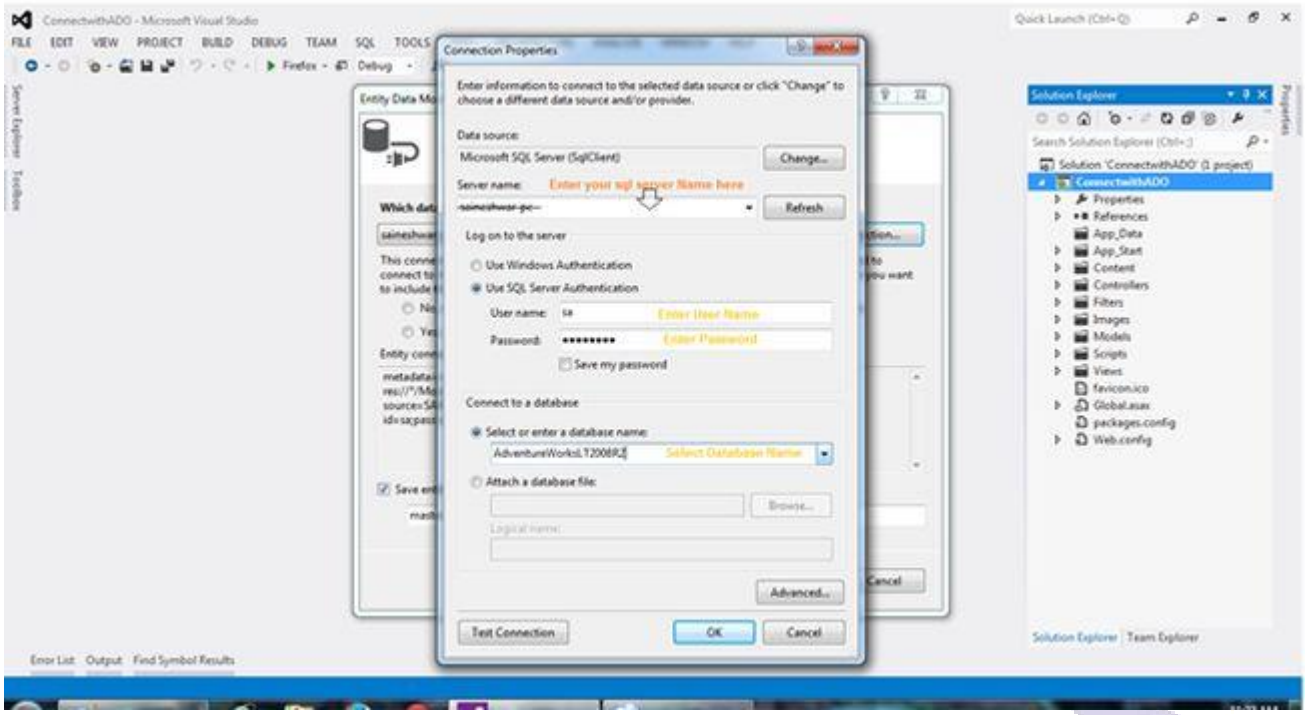


Then it will prompt for your Database Connection.



Click on New Connection; a Connection Properties Wizard will pop-up; fill in all your details.

In the following snapshot you will see where to fill in the details.



After Entering the details of Username and Password a list of databases will pop up.

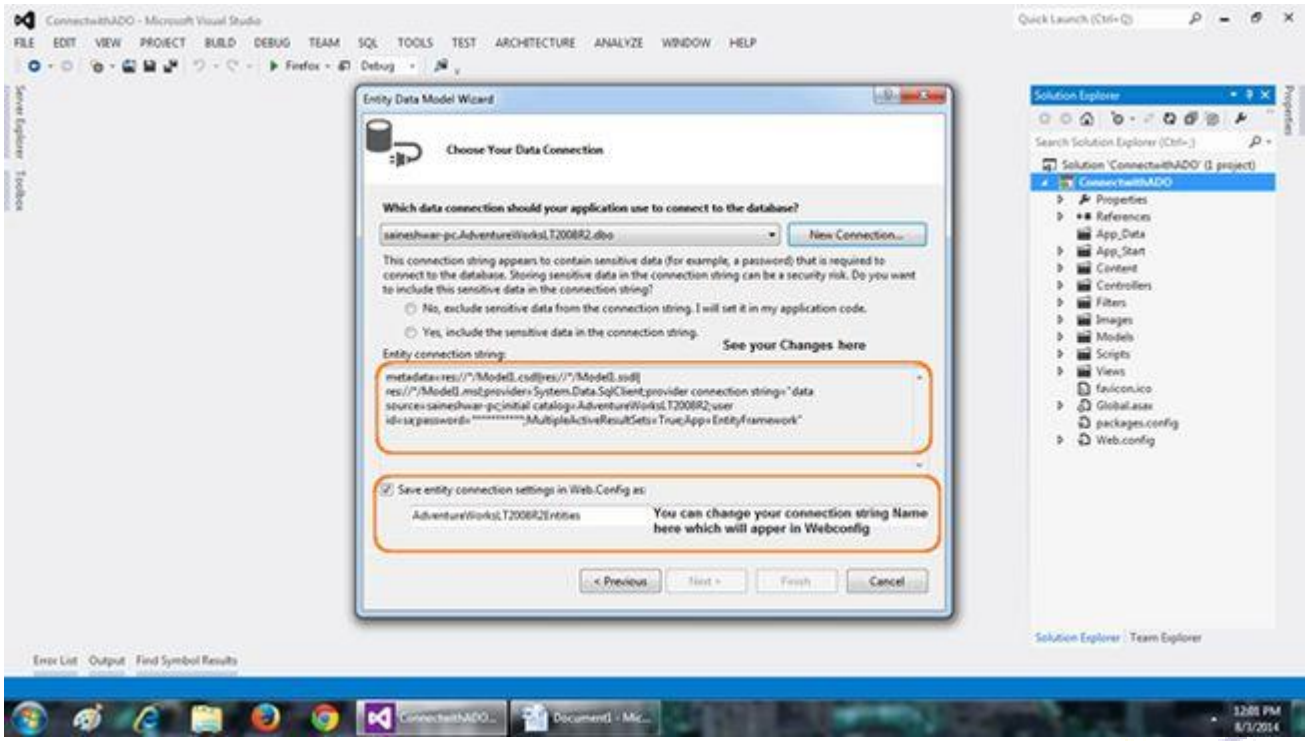
Select the database, whatever you have .

But in this I will select AdventureWorksLT2008R2.

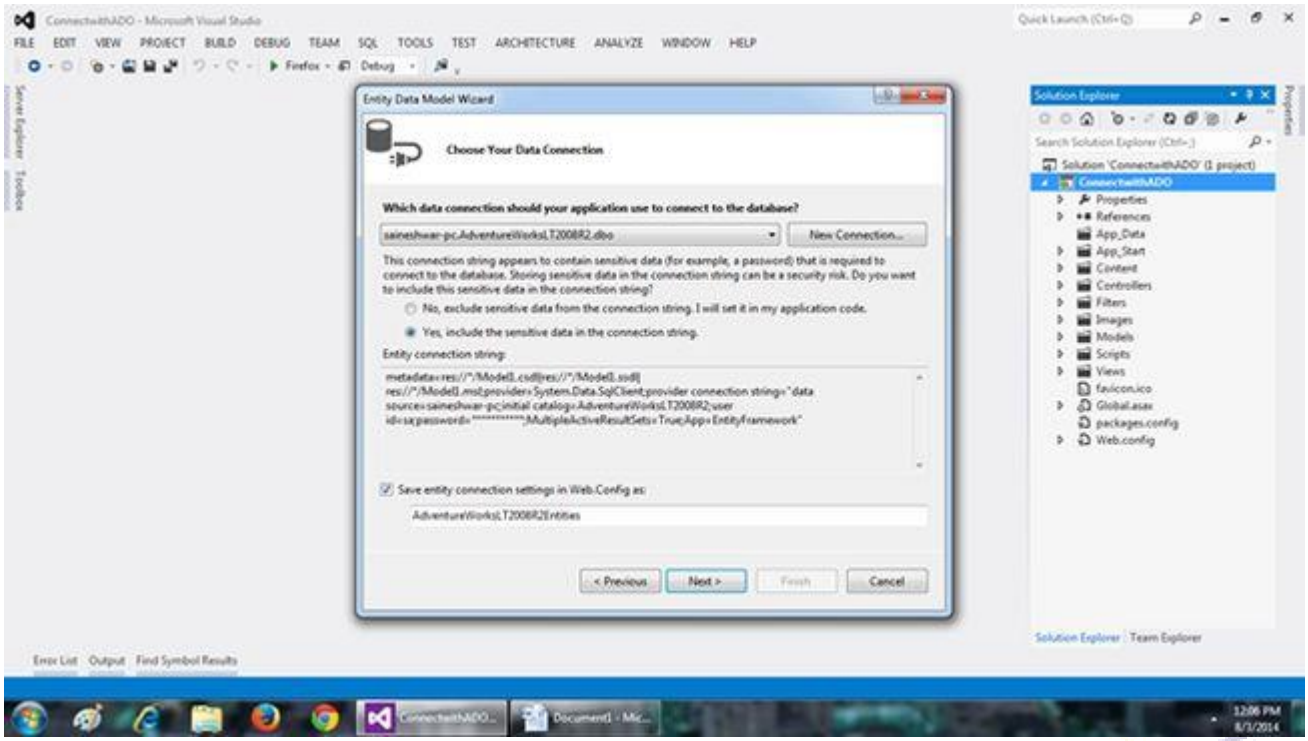
Click on Test Connection and check it then click OK.

Then you can see all Changes in the Entity Connection String that have you chosen.

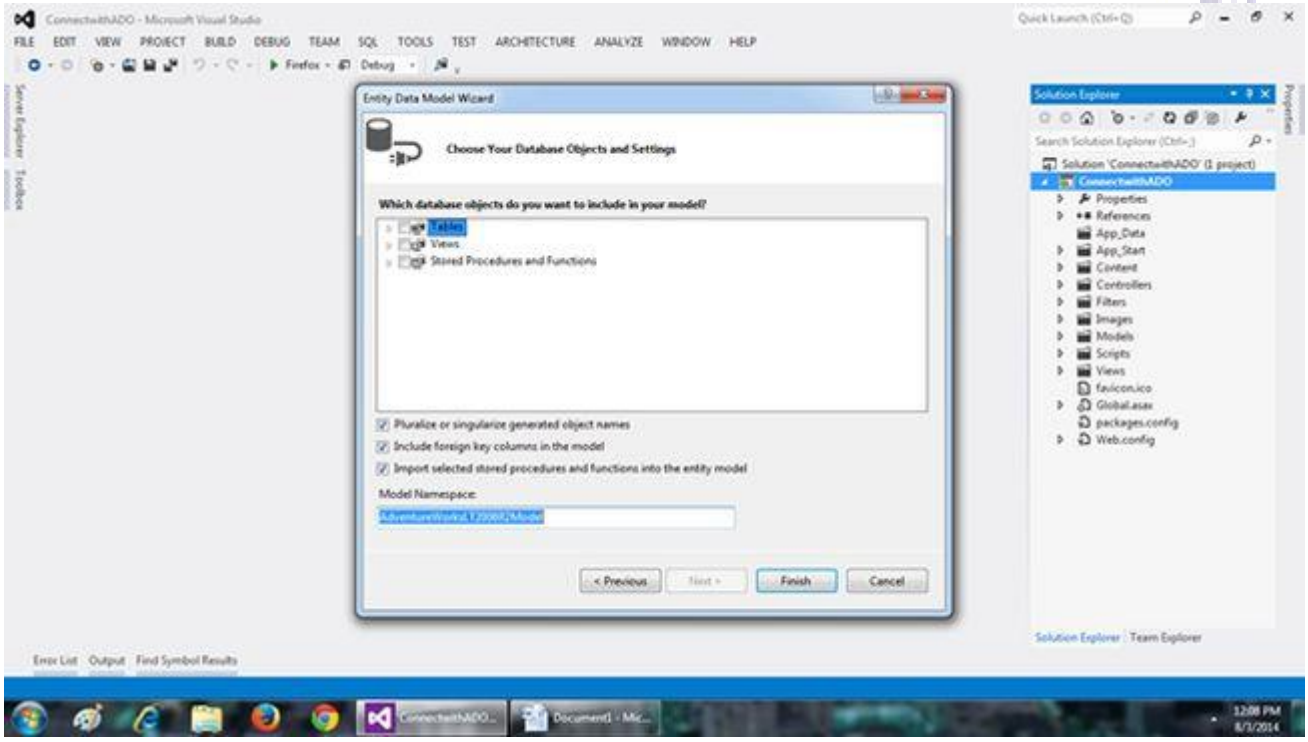
You can also configure the webConfig Connection String Name.



The last option to choose is to show the sensitive data in the Connection string; click Yes.

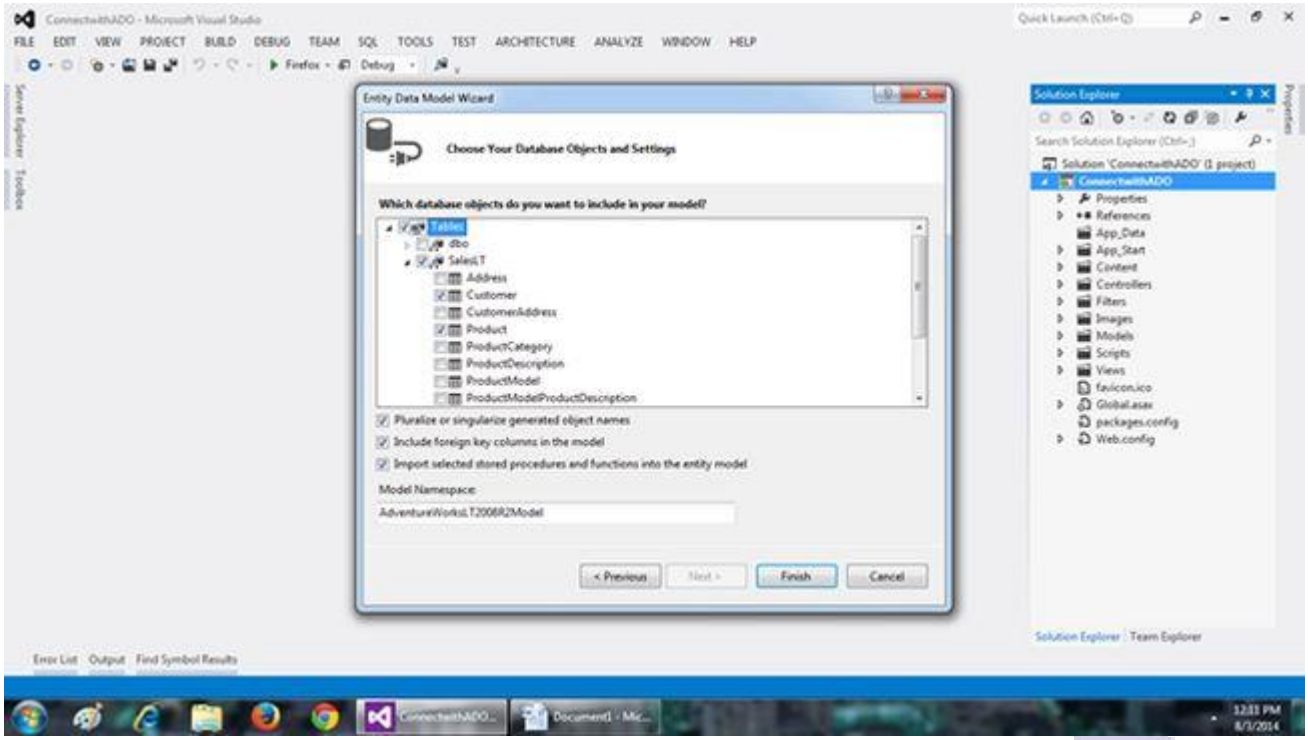


Step 4: Select the object to use.

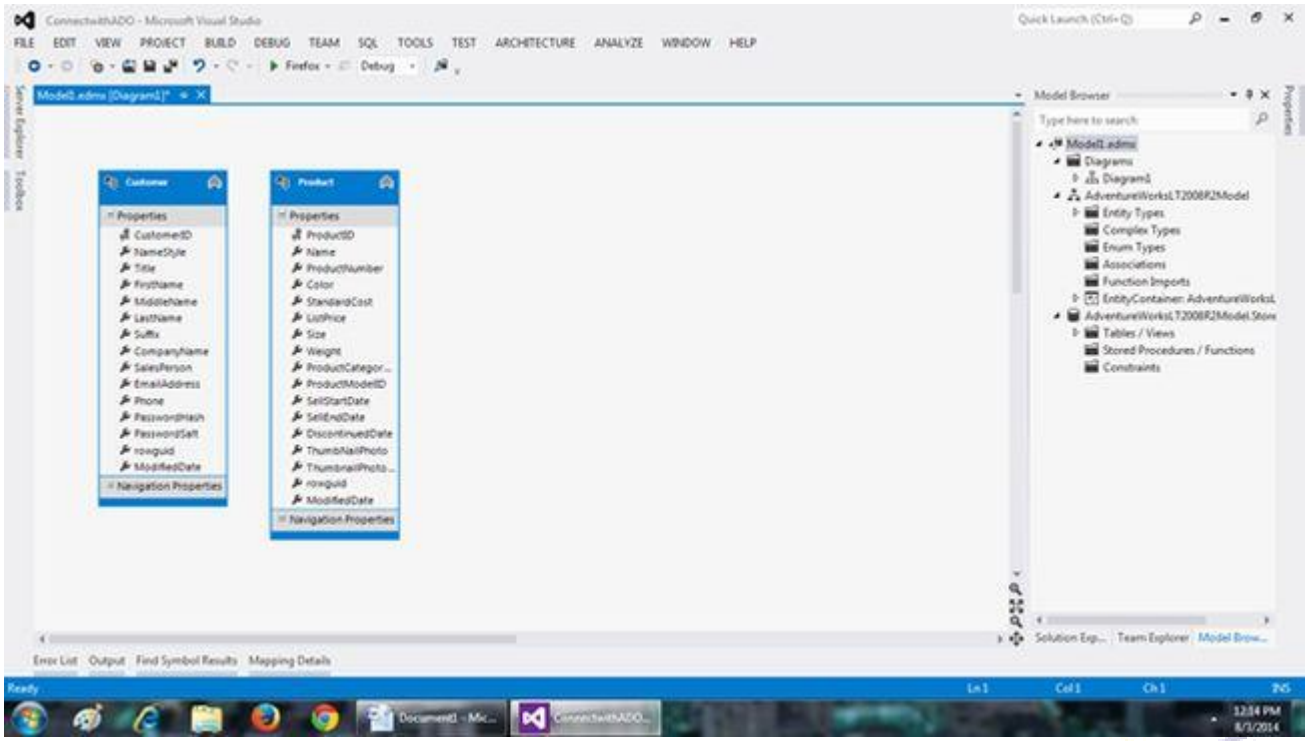


Select Tables and Expand it to see all the tables.

In that I will select 2 tables, customer and Product, and finally click on Finish.

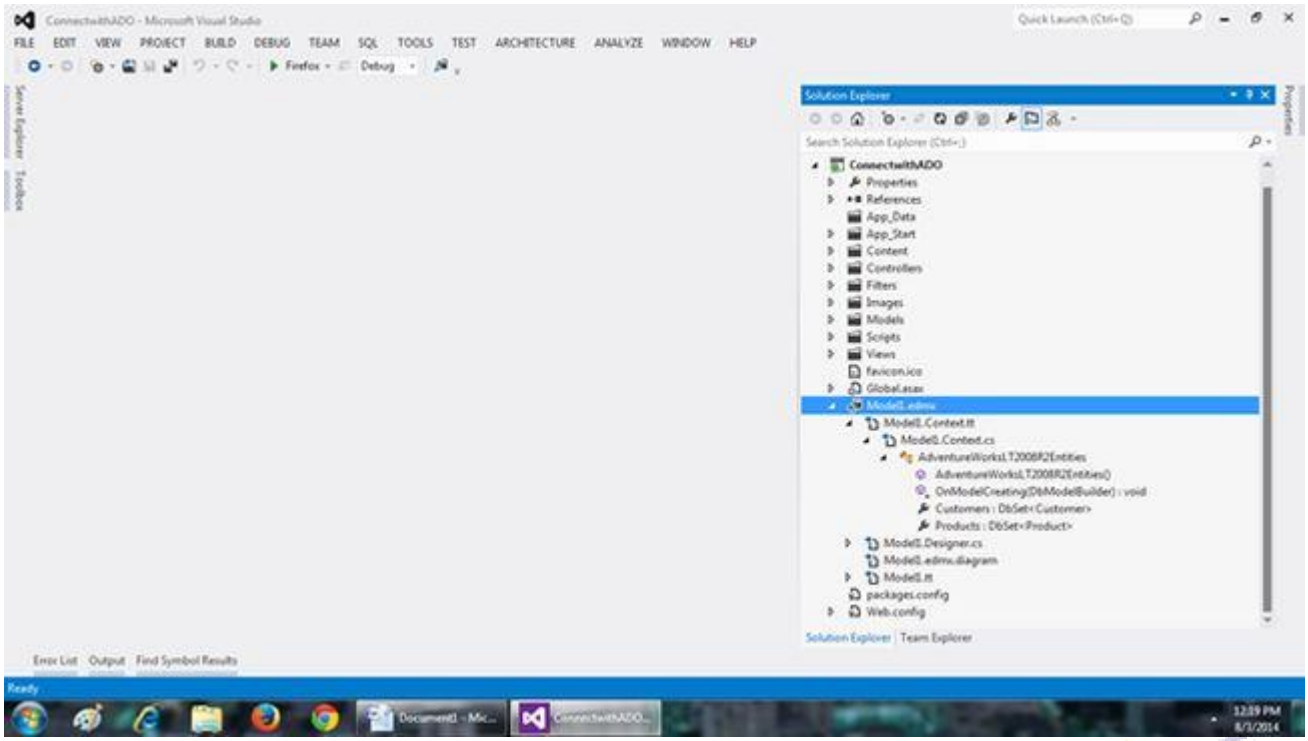


After this Visual Studio will generate a database diagram for the table that we chose.

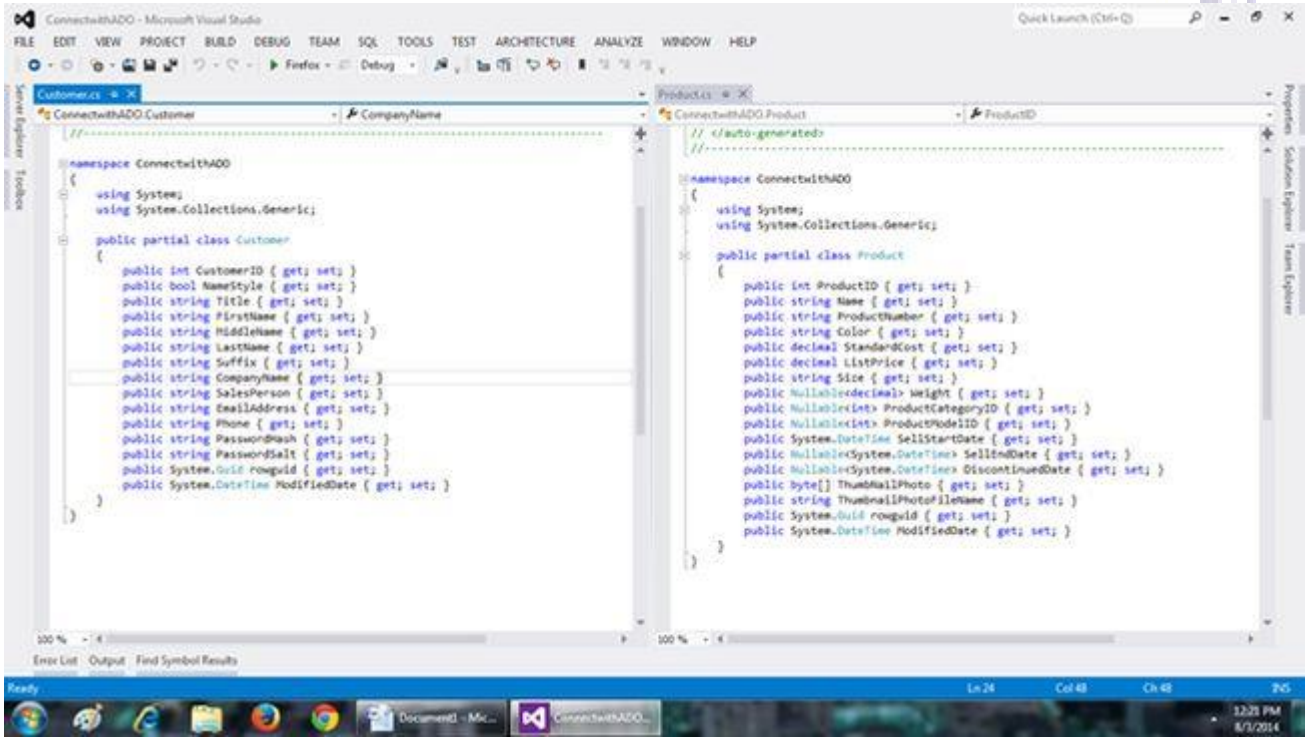


Now save your project.

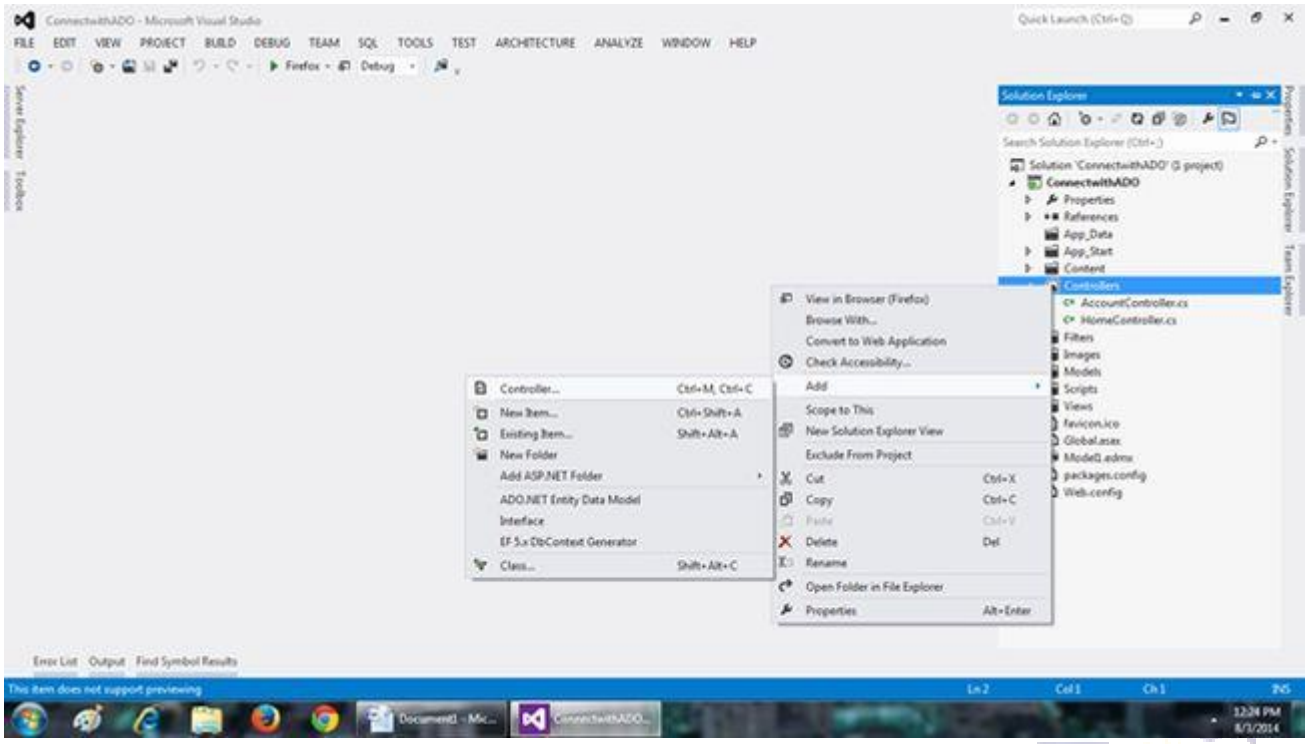
In Solution Explorer you will find the model and your tables.



Add your table class will be created.

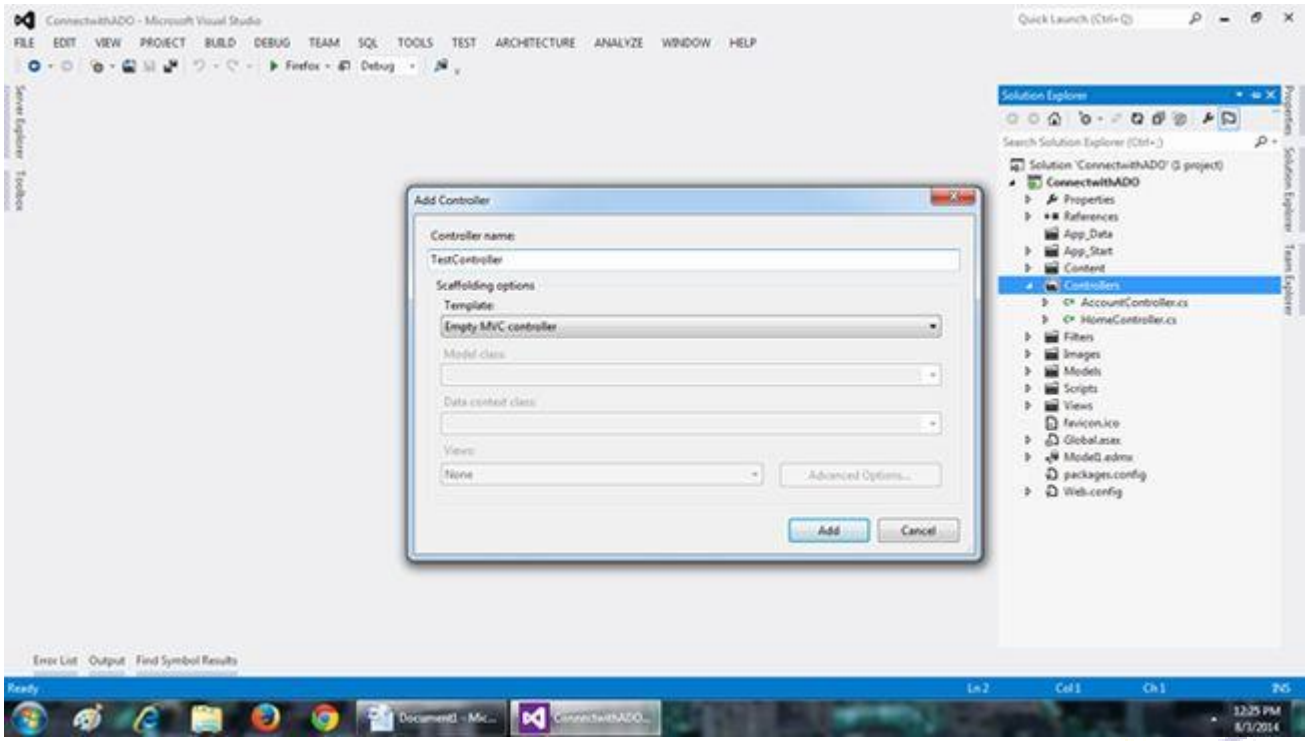


Step 5: Build the project and add the Controller for the model that we created.

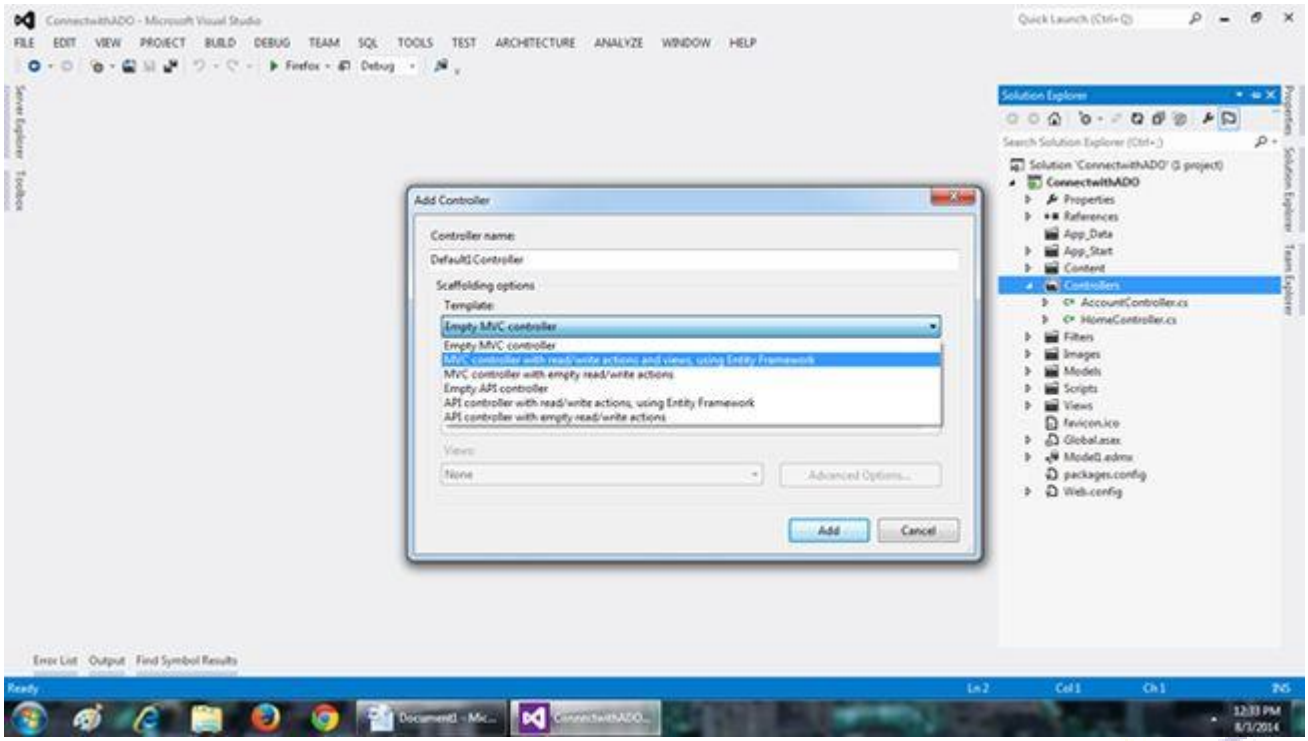


After clicking on the Controller, the Add Controller Wizard will pop up.

I will name the controller TestController.

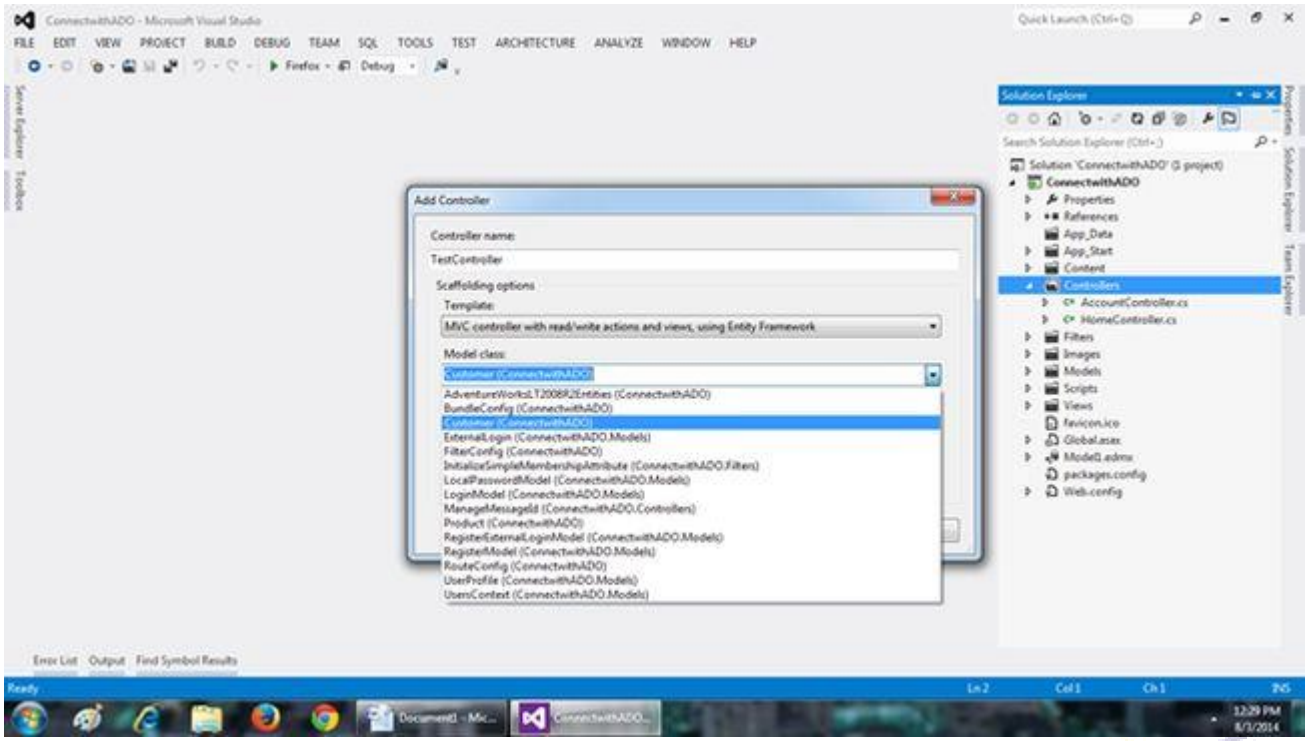


In the template select MVC Controller with read write action and views using Entity Framework.



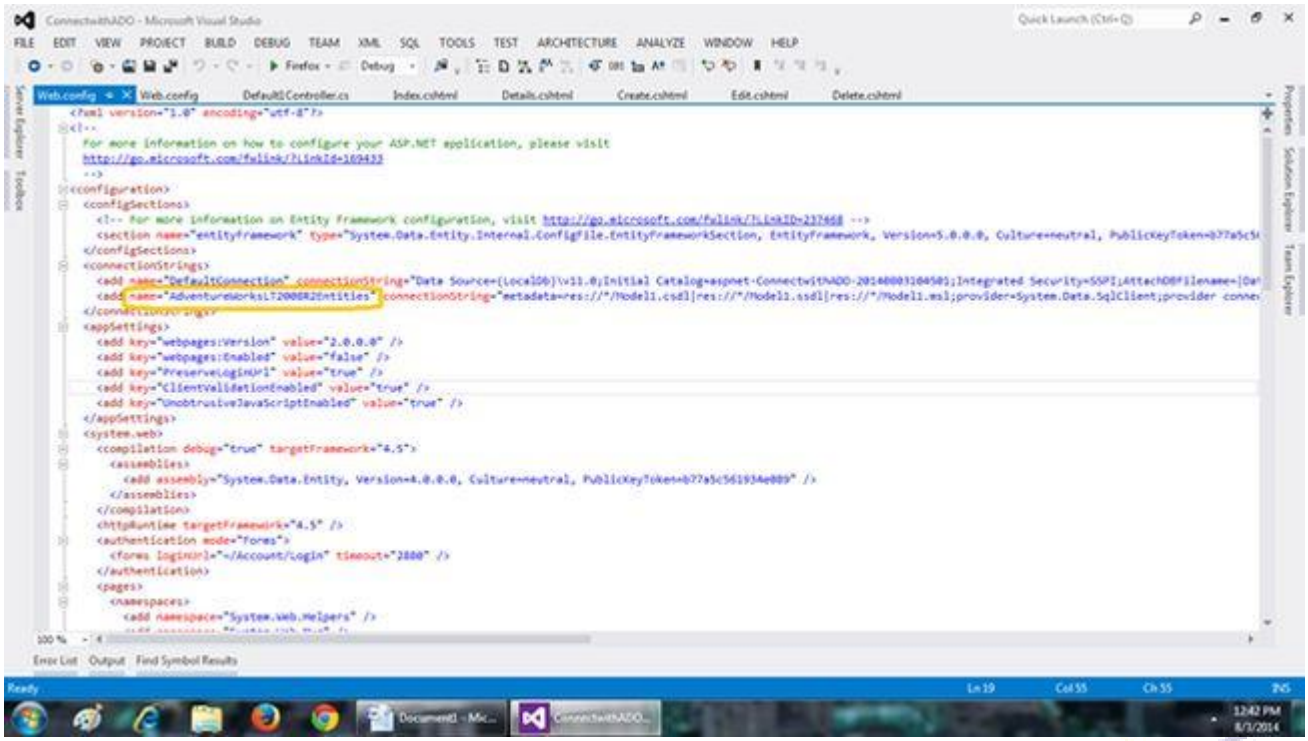
Select the Model Class for generating your CRUD for that model.

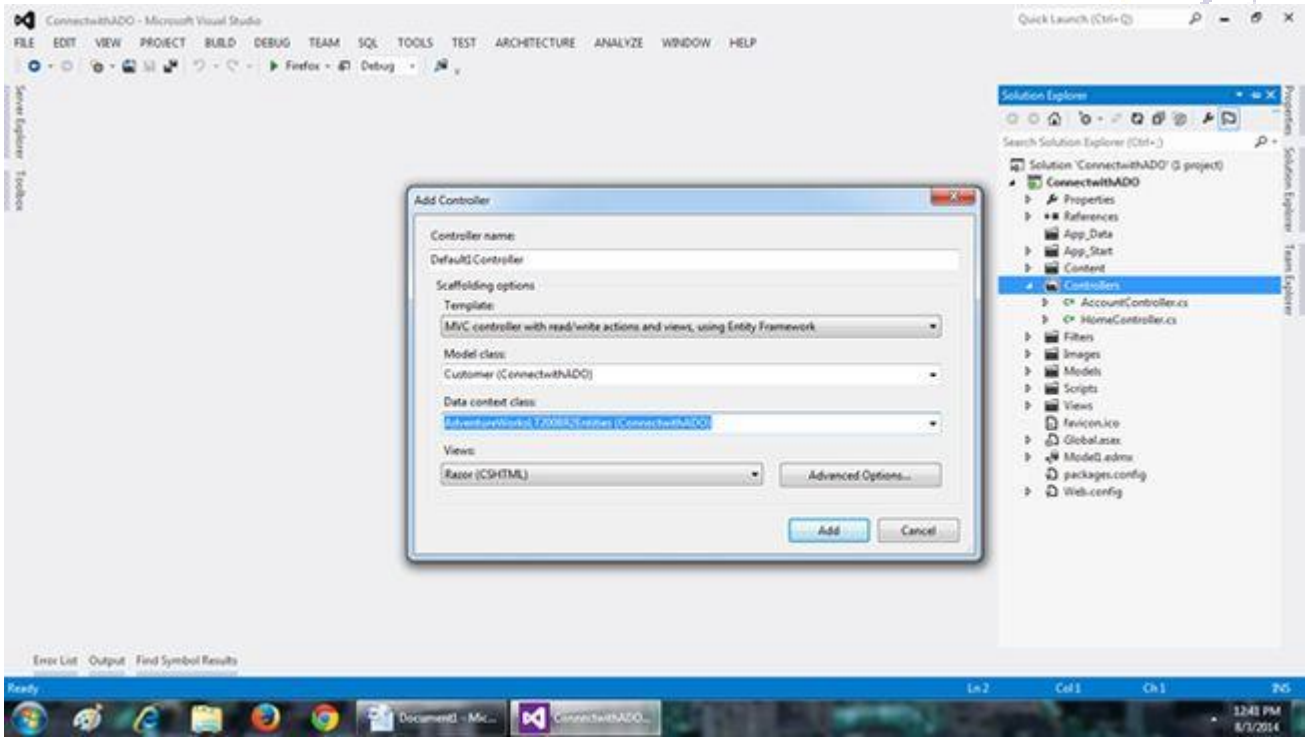
Here I will choose Customer.



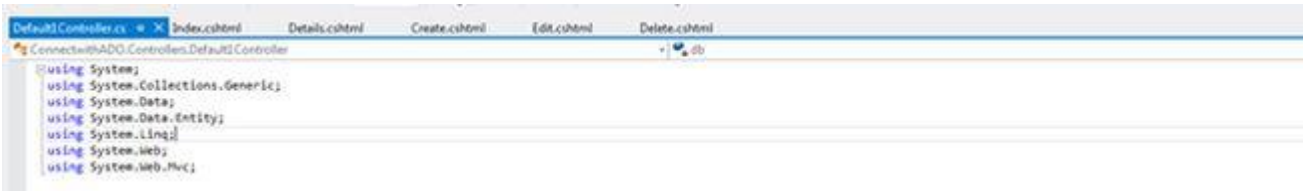
Finally I will select Data Context class that I created in Step 4.

You will also find the Data Context class in the webconfig.

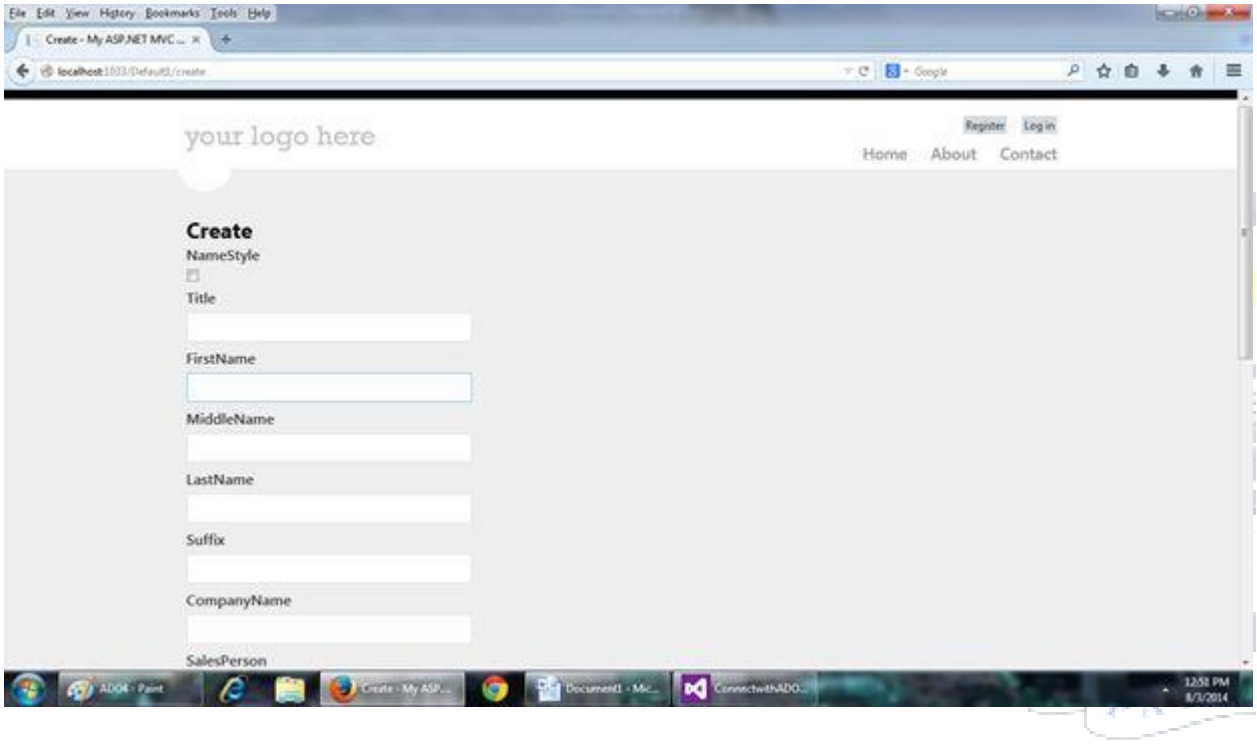




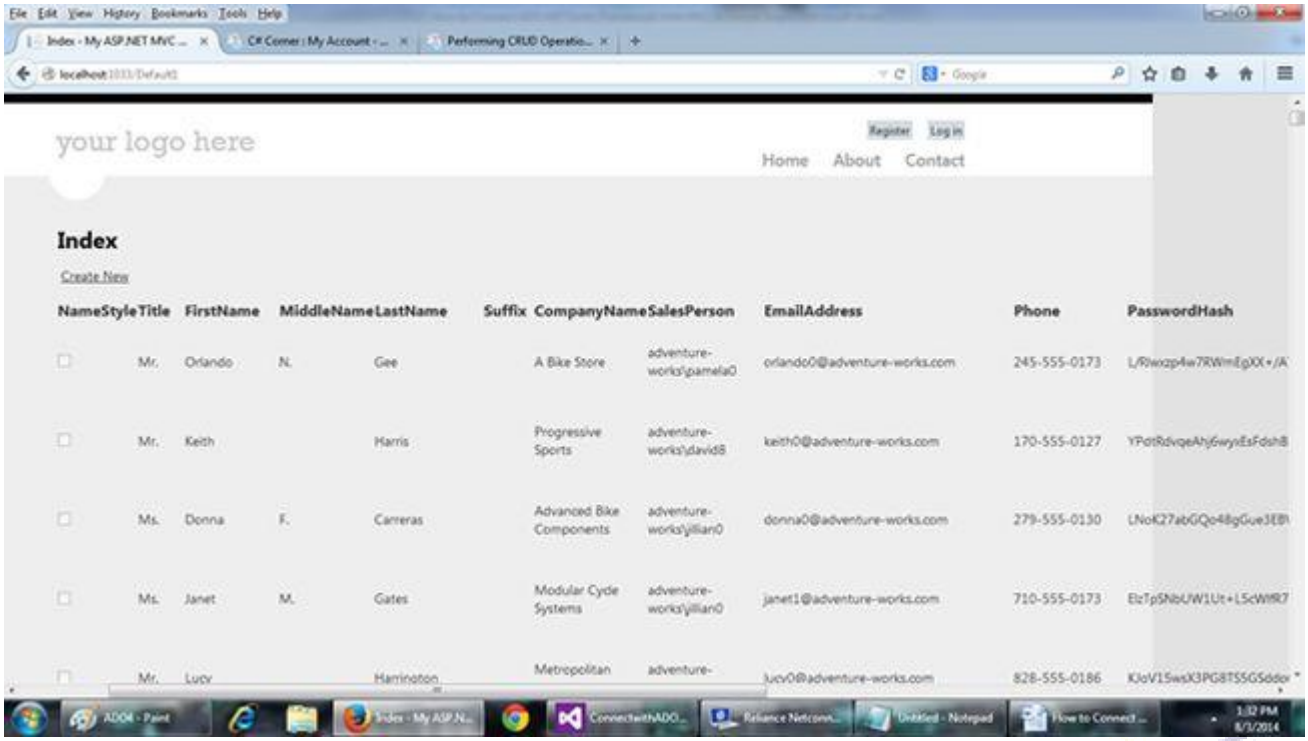
After clicking add you will find all CRUD views and the Controller is generated.



Step 6: Now just Run Project.



Here all the table data for the customer has been populated in the list. It ensures that our ADO.NET is working.



Time Boxing

Activity Name	Activity Time	Total Time
Login Systems + Setting up Visual studio Environment	3 mints + 5 mints	8 mints
Walk through Theory & Tasks	60 mints	60 mints
Implement Tasks	80 mints	80 mints
Evaluation Time	30 mints	30 mints
	Total Duration	178 mints

Objectives

After completing this lab the student should be able to:

- Clearly understand the purpose and benefits that Cloud Computing has to offer.
- Understand the concept of Cloud Computing.
- Build a basic Cloud Computing base application.
- Use and learn Cloud Computing need.

Lab Tasks/Practical Work

1. Create login form using simple MVC net core
2. Create an employee profile and apply crud operations on it

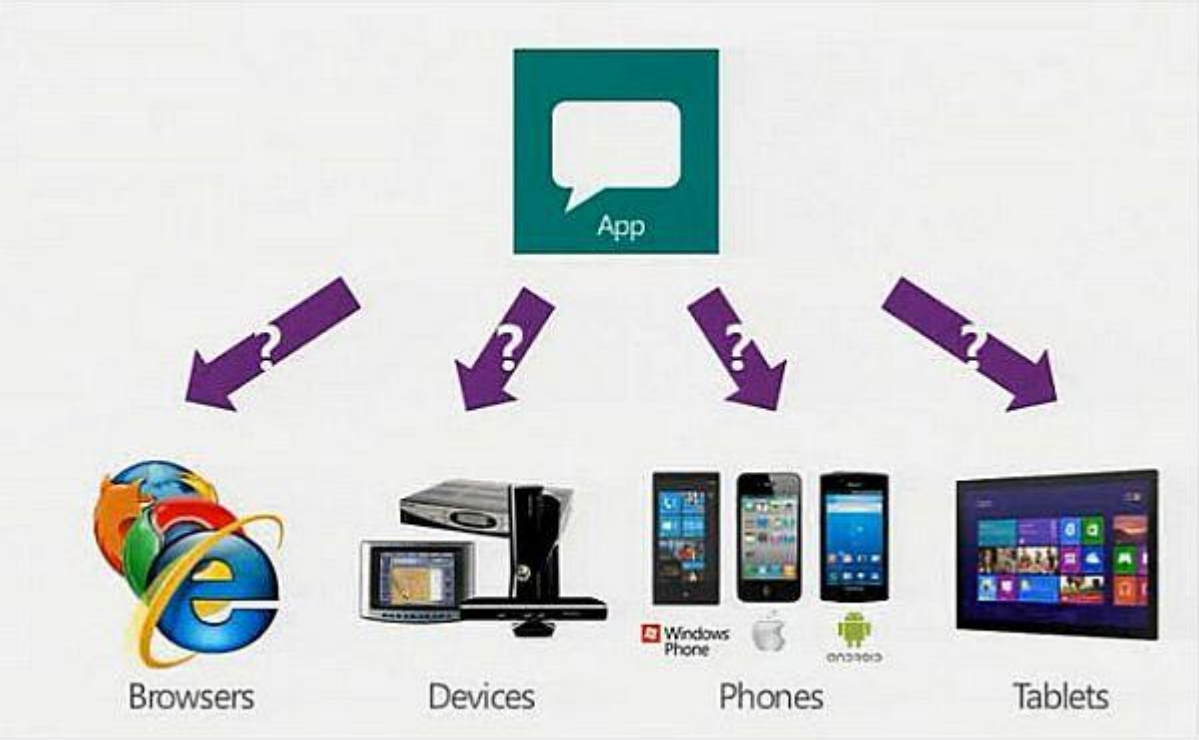
Lab No. 4

Creating ASP.Net Core Web API

LAB # 04

Creating ASP.Net Core Web API

Introduction



Web API

The ASP.NET Web API is a framework that makes it easy to build HTTP services that reach a broad range of clients, including browsers and mobile devices.

The ASP.NET Web API is an ideal platform for building Restful applications on the .NET Framework. Referred from “Microsoft.com”.

Why to use the Web API

Currently most mobile devices, browsers and tablets are the medium for accessing most of the internet and in this also people are using mobile apps the most and to provide data to apps we are now going to use the Microsoft new technology called Web API.

When to use it

If you want to expose the data/information of your application to your clients and other people then that other people can use your data and interact with the data/information you expose to them.

For example, a mobile application requires a service.

- HTML 5 requires a service.
- Desktop PC and tablets require services.
- Currently most device apps require Web API services.
- The ASP.Net Framework leverages both web standards such as HTTP, JSON and XML and it provides a simple way to build and expose REST based data services.
- Some core concepts of ASP.Net MVC are similar to the ASP.Net Web API such as routing and controllers.

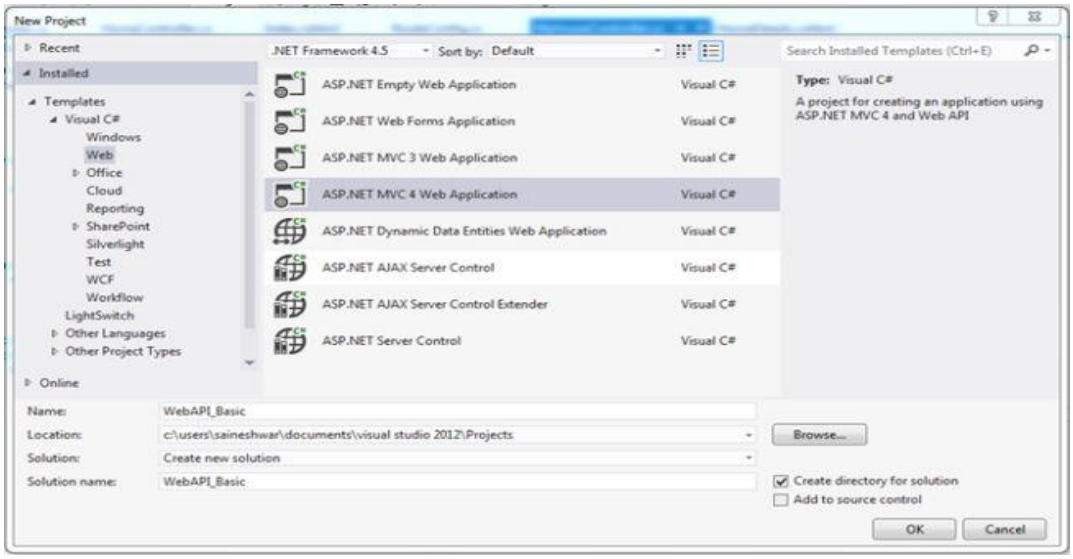
Requirements

We are using Visual Studio 2012 for a demo application.

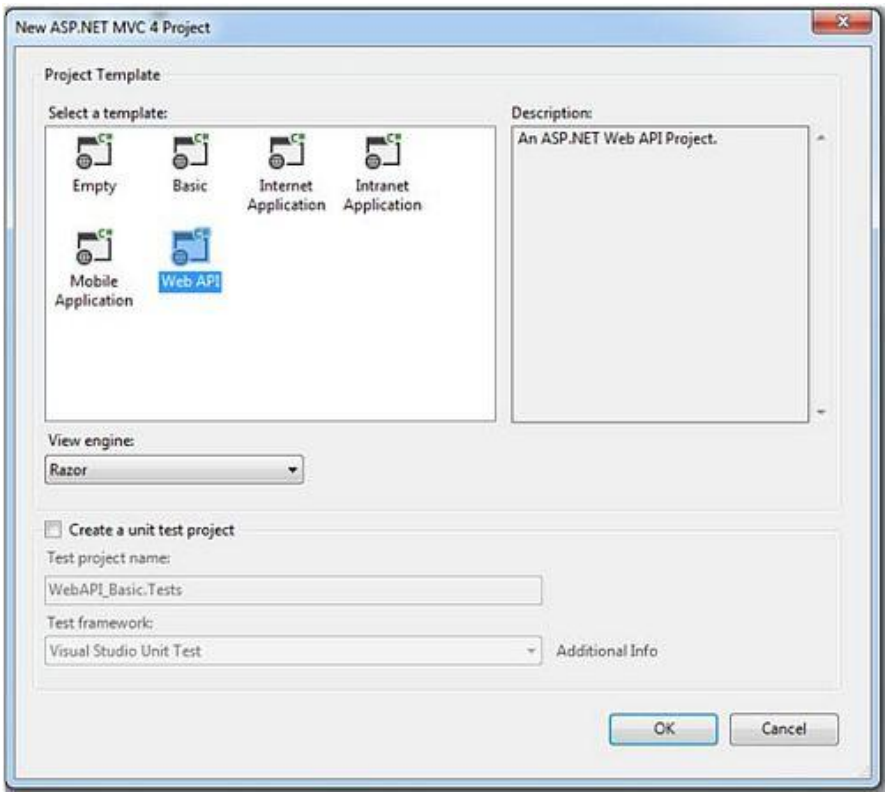
Building a Web API

Let's start with creating a Web API project.

- Start Visual Studio and select New project from the Start page or from the File menu select "File" -> "New" -> "Project...".
- In the template pane select Installed Templates and expand the Visual C# menu. Inside that Visual C# select Web. In the list of projects select ASP.Net MVC 4 Web A
- And name the project WebAPI_Basic.
- For reference see the following snapshot.



After adding, a new dialog will pop-up.



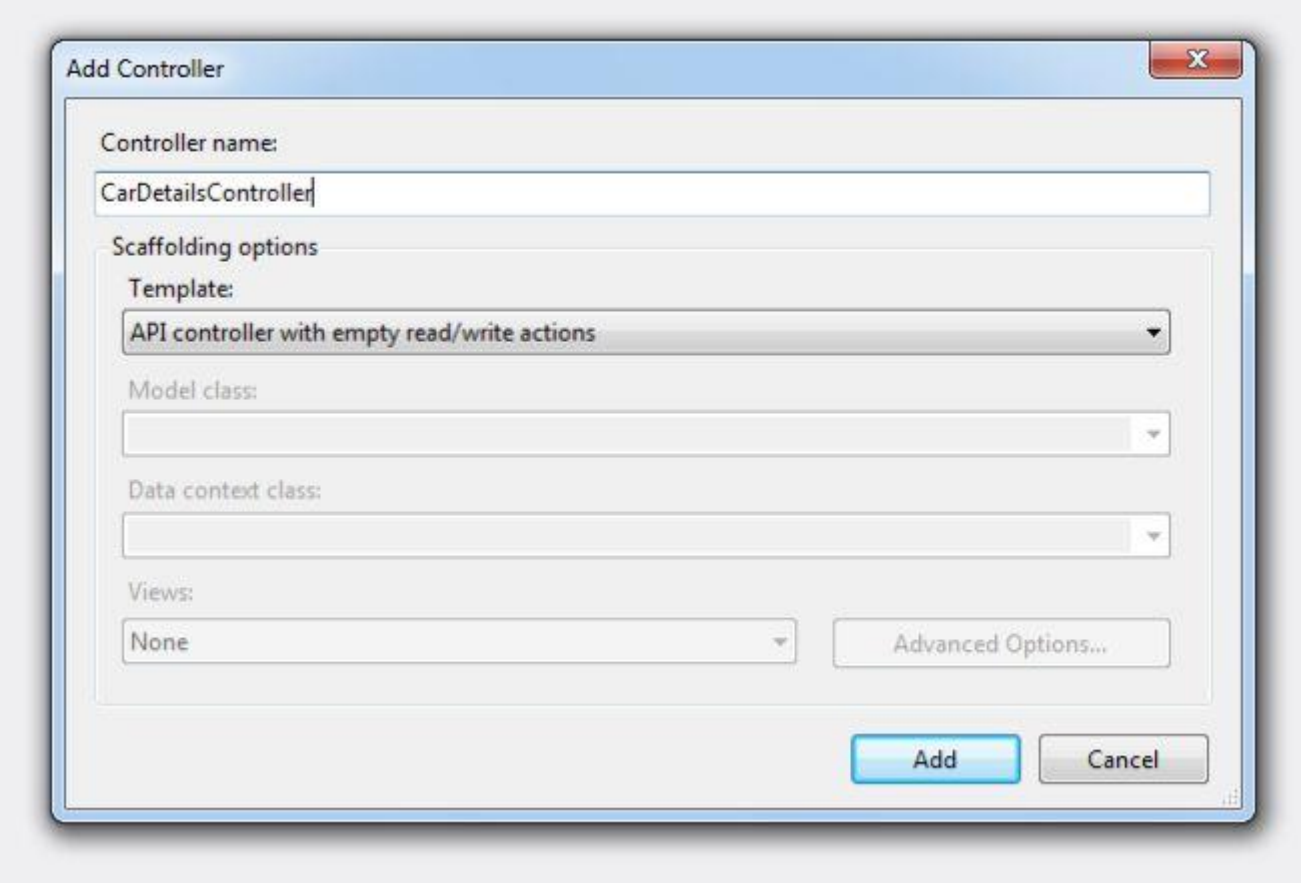
Inside the project template select Web API and in the view engine select Razor.

A new project is created now.

Let's begin with adding Web API Controller

Now let's begin with adding a Web API Controller. It is nearly similar to adding a Controller in ASP.NET MVC.

Right-click on the Controller folder and add a new Web API Controller with the name CarDetailsController and in the template select API Controller with an empty read / write action.



After adding the Controller you will see the code as in the following snapshot.

```
namespace WebAPI_Basic.Controllers
{
    public class CarDetailsController : ApiController
    {
        // GET api/cardetails
        public IEnumerable<string> Get()
        {
            return new string[] { "value1", "value2" };
        }

        // GET api/cardetails/5
        public string Get(int id)
        {
            return "value";
        }

        // POST api/cardetails
        public void Post([FromBody]string value)
        {
        }

        // PUT api/cardetails/5
        public void Put(int id, [FromBody]string value)
        {
        }

        // DELETE api/cardetails/5
        public void Delete(int id)
        {
        }
    }
}
```

You can keep this Web API controller anywhere in the application.

If you want to follow the convention then create the new folder in the root your of application with the name API.

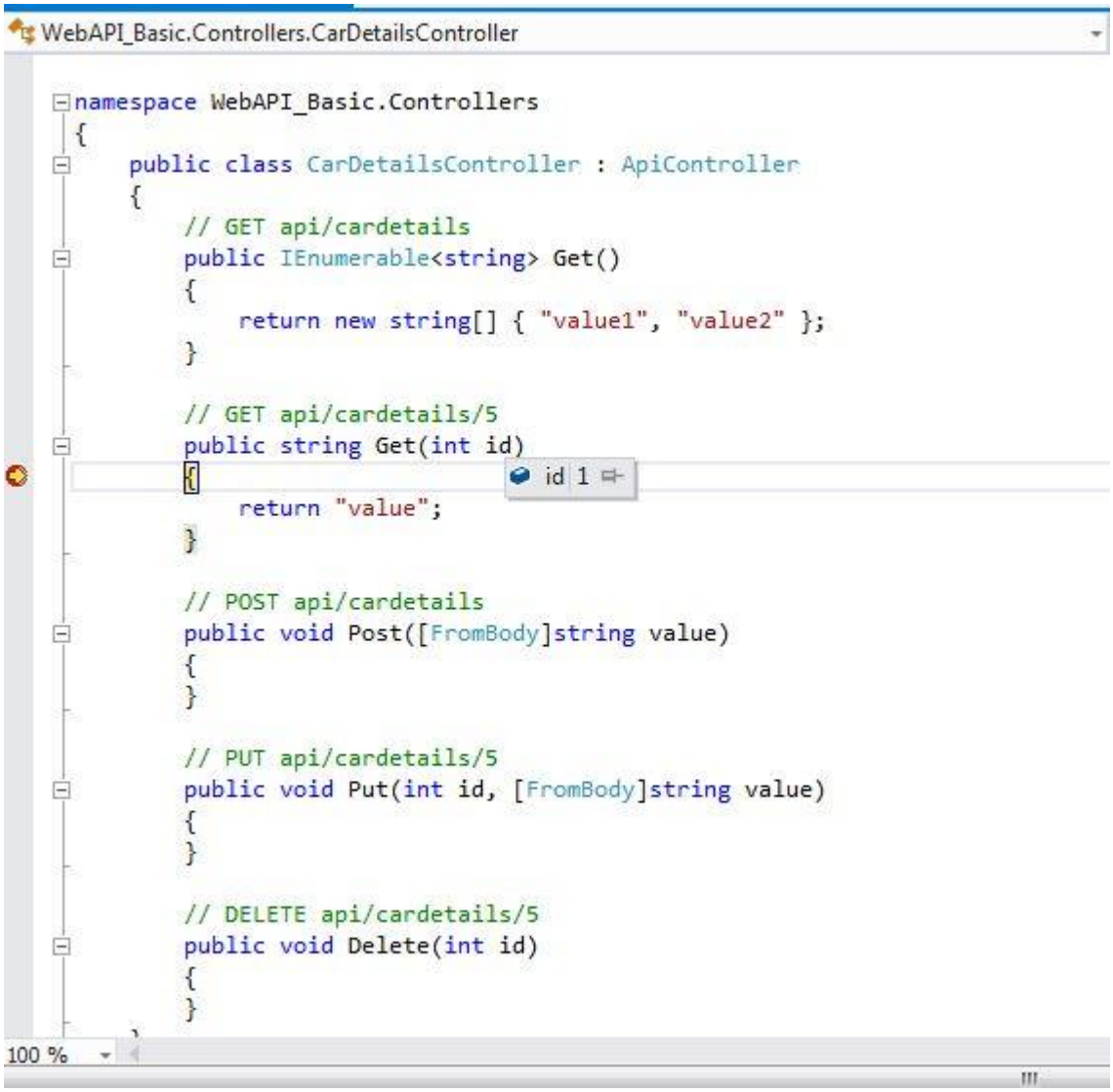
Inside that you can add a Web API controller.

You have successfully added a Web API controller to your application.

Now you can run the application and test it.

For testing I am passing http://localhost:32359/api/cardetails/1 for calling the method get.

Wow, it's working!



It's easy to configure it as a Web API.

The ASP.Net MVC and ASP.Net Web API makes heavy use of convention for configuration to lighten the work load for creating the services.

For example, add a decorating method with attributes to make it easy to do CRUD operations.

Else it will make it difficult to understand and code.

```
1. [HttpPut]
2.         public void Put(int id, [FromBody]string value)
3.         {
```

```
4.
5.              }
6.    [HttpPost]
7.          public void Post([FromBody]string value)
8.          {
9.
10.             }
11. [HttpDelete]
12.         public void Delete(int id)
13.         {
```

The HTTP actions and their corresponding CRUD operations are:

- GET (Read)
  Retrieves the representation of the resource.

- PUT(Update)
  Update an existing resource.

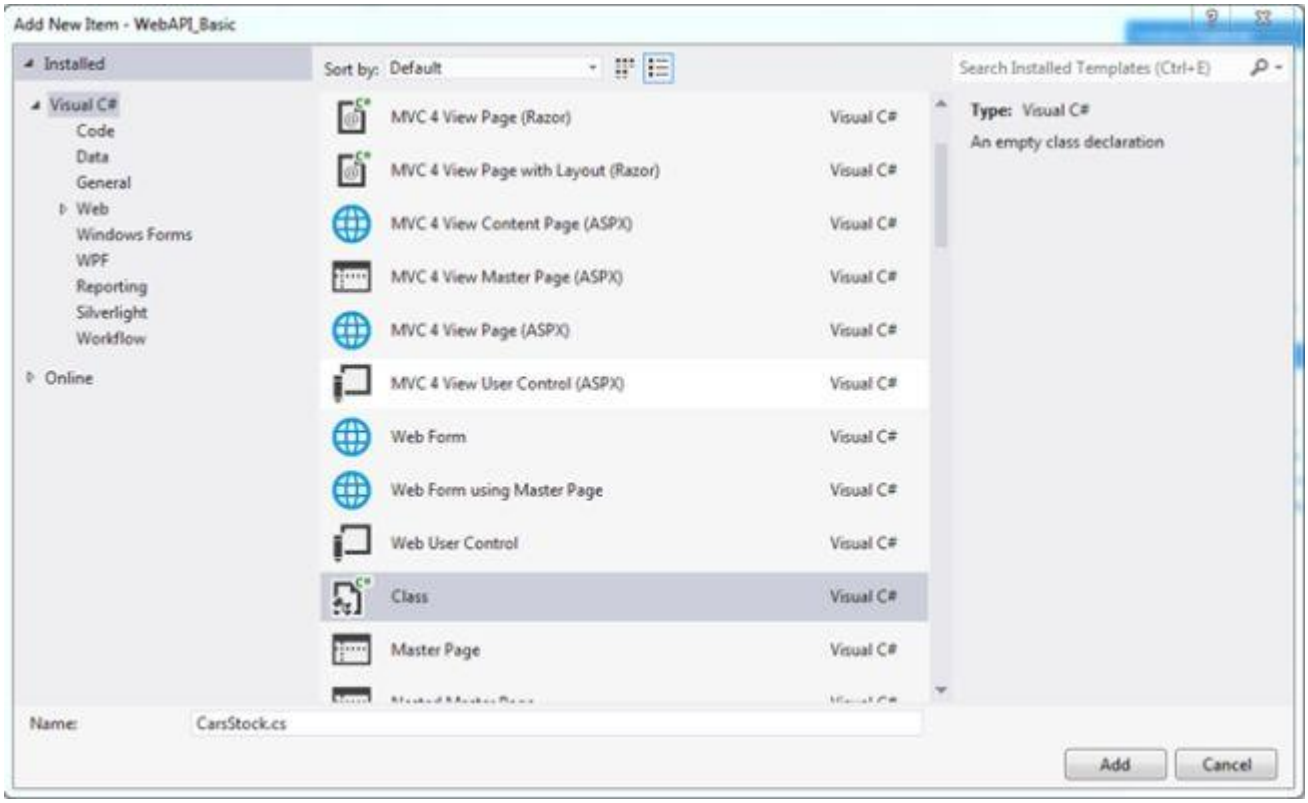
- POST (Create)
  Create new resource.

- DELETE (Delete)
  Delete an existing resource.

Now let's begin with how to create a CRUD operation with the WEB API.

Let's start by adding a Model.

To add the model right-click on the model folder and add a class with the name CarsStock.



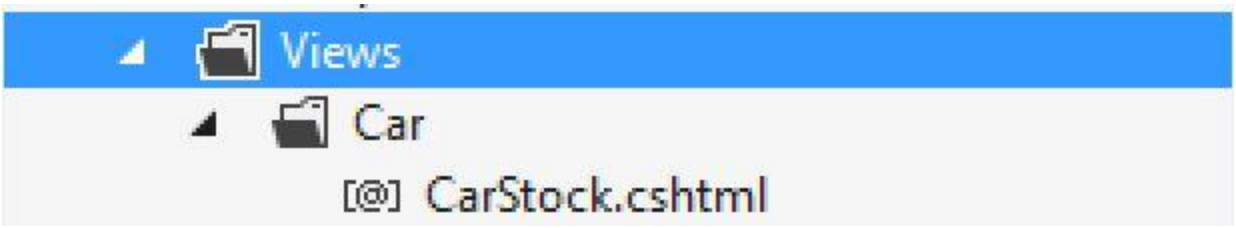
After adding the Model CarsStock.cs now let's start with adding properties to the class.

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

namespace WebAPI_Basic.Models
{
    public class CarsStock
    {
        public string CarName { get; set; }
        public string CarModel { get; set; }
        public string CarPrice { get; set; }
        public string CarColor { get; set; }

    }
}
```

After adding the model properties now I will consume the HTTP service developed using the ASP.NET Web API in a simple cshtml page with jQuery and Ajax.

For that in the View folder I will add a folder named Car and inside that folder will add a CarStock named view. To add it just right-click on the View folder and select View.

The following snapshot shows how I had added the view.



After adding the view you will get a blank view because we are not using any tightly coupled model here.

Then add a Controller with the name CarController. Call this view Carstock for the demo of consuming the Web API.

In this I called the view CarStock.

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

namespace WebAPI_Basic.Controllers
{
    public class CarController : Controller
    {
        //
        // GET: /Car/

        public ActionResult Car()
        {
            return View("CarStock");
        }

    }
}
```

After adding the Controller and View now let us move back towards the Web API and make some changes that we have already created with the name “CarDetailsController”.

Let's get to the first method in CarDetailsController.

1. **GET IEnumerable**

```
[HttpGet]
 public IEnumerable<CarsStock> GetAllcarDetails()
 {
    CarsStock ST = new CarsStock();
        CarsStock ST1 = new CarsStock();
    List<CarsStock> li = new List<CarsStock>();

    ST.CarName = "Maruti Waganor";
    ST.CarPrice = "4 Lakh";
    ST.CarModel = "VXI";
    ST.CarColor = "Brown";

    ST1.CarName = "Maruti Swift";
    ST1.CarPrice = "5 Lakh";
    ST1.CarModel = "VXI";
    ST1.CarColor = "RED";

    li.Add(ST);
    li.Add(ST1);
    return li;
 }
```

This method is used to get a list of data.

In this method I have used the Model CarsStock and created a list of CarsStock “List<CarsStock>“.

And returning it.

GET by id

```
public IEnumerable<CarsStock> Get(int id)
{
   CarsStock ST = new CarsStock();
   CarsStock ST1 = new CarsStock();
```

```
25.     List<CarsStock> li = new List<CarsStock>();
26.     if (id == 1)
27.     {
28.         ST.CarName = "Maruti Waganor";
29.         ST.CarPrice = "4 Lakh";
30.         ST.CarModel = "VXI";
31.         ST.CarColor = "Brown";
32.         li.Add(ST);
33.     }
34.     else
35.     {
36.         ST1.CarName = "Maruti Swift";
37.         ST1.CarPrice = "5 Lakh";
38.         ST1.CarModel = "VXI";
39.         ST1.CarColor = "RED";
40.         li.Add(ST1);
41.     }
42.     return li;
43. }
```

In this GET method you can retrieve records for the database by passing an id.

POST

```
44. [HttpPost]
45.  public void PostCar([FromBody] CarsStock cs)
46.  {
47.
48.  }
```

In this POST method you can post data (CREATE) to the database. In this I am using the Carstock model to post the data.

PUT

```
2.   [HttpPut]
3.  public void Putcar(int id, [FromBody]CarsStock cs)
4.  {
5.
```

```
6.      }
```

In this PUT method you can UPDATE the data (UPDATE) to the database. I am using the Carstock model to update the data.

DELETE

```
7.  [HttpDelete]
8.      public void Deletecar(int id)
9.       {
10.
11.      }
```

In this DELETE method you can delete data (DELETE) from the database. I am using an id to delete the data.

Here is a snapshot of all the methods and models after adding the attributes to it.

```
using System.Web.Http;
using WebAPI_Basic.Models;
namespace WebAPI_Basic.Controllers
{
    public class CarDetailsController : ApiController
    {

        [HttpGet] // GET api/cardetails
        public IEnumerable<CarsStock> GetAllcarDetails()...

        // GET api/cardetails/5
        public string Get(int id)...


        [HttpPost] // POST api/cardetails
        public void PostCar([FromBody] CarsStock cs)...


        [HttpPut]  // put api/cardetails/5
        public void Putcar(int id, [FromBody]CarsStock cs)...


        [HttpDelete] // put api/cardetails/5
        public void Deletecar(int id)...

    }
}
```

Now let's move to the view and do CRUD operations from there.

For getting a list of data I have created a function in jQuery.

1. Calling GET IEnumerable List from Ajax and getting data from the Web API.

```
<script lang="ja" type="text/javascript">

    function AllcarDetails() {
        $.ajax({
            type: "GET",
            url: "http://localhost:32359/api/Cardetails", //URI

            dataType: "json",
            success: function (data) {
                debugger;
                var datadatavalue = data;
                var myJsonObject = datavalue;
```

```
            contentType: "application/json";
            $.each(myJsonObject, function (i, mobj) {
                $("#Cartbl").append('<tr><td width="50px">' + mobj.CarName +
                 '</td><td width="50px">' + mobj.CarModel +
                '</td><td width="50px">' + mobj.CarPrice +
                '</td>' + '</td><td width="50px">'
                + mobj.CarColor + '</td></tr>');

            });

        },
        error: function (xhr) {
            alert(xhr.responseText);
        }
    });

}
```

2. Calling PostCar Method using Ajax and posting data to the Web API.

```
function PostData()
 {

 var cardetails =
 {
CarName: "Ertiga",
CarModel: "LXI",
CarPrice: "5000000",
CarColor: "blue"
};

    $.ajax({
        type: "POST",
        data: JSON.stringify(cardetails),
        url: "http://localhost:32359/api/Cardetails",
        dataType: "json",
        contentType: "application/json",
    });

 }
```

3. Calling the PUTcar method using Ajax and updating the data of the Web API.

```
function PutData() {

```

```
var cardetails =
 {

 CarName: "Ertiga",
 CarModel: "LXI",
 CarPrice: "5000000",
 CarColor: "blue"

 };

      var t = JSON.stringify(cardetails);
      var id = "0";
      $.ajax({
          url: 'http://localhost:32359/api/Cardetails/' + id,
          type: "put",
          contentType: "application/json; charset=utf-8",
          data: t,
          dataType: "json",

      });
  }
```

4. Calling the Delete car method using Ajax and to delete data of the Web API.

```
function deleteData1()
   {
          var id = 0;
          $.ajax({
              url: 'http://localhost:32359/api/CarDetails/' + id,
              type: 'DELETE',
              success: function (data) {

              },
              error: function (data) {
                  alert('Problem in deleting car:' + data.responseText);
              }
          });
}
```

5. Calling GET by ID from Ajax and getting data from the Web API by id.

```
function GetCarById() {
      var id = 1;
      $.ajax({
          url: 'http://localhost:32359/api/CarDetails/' + id,
```

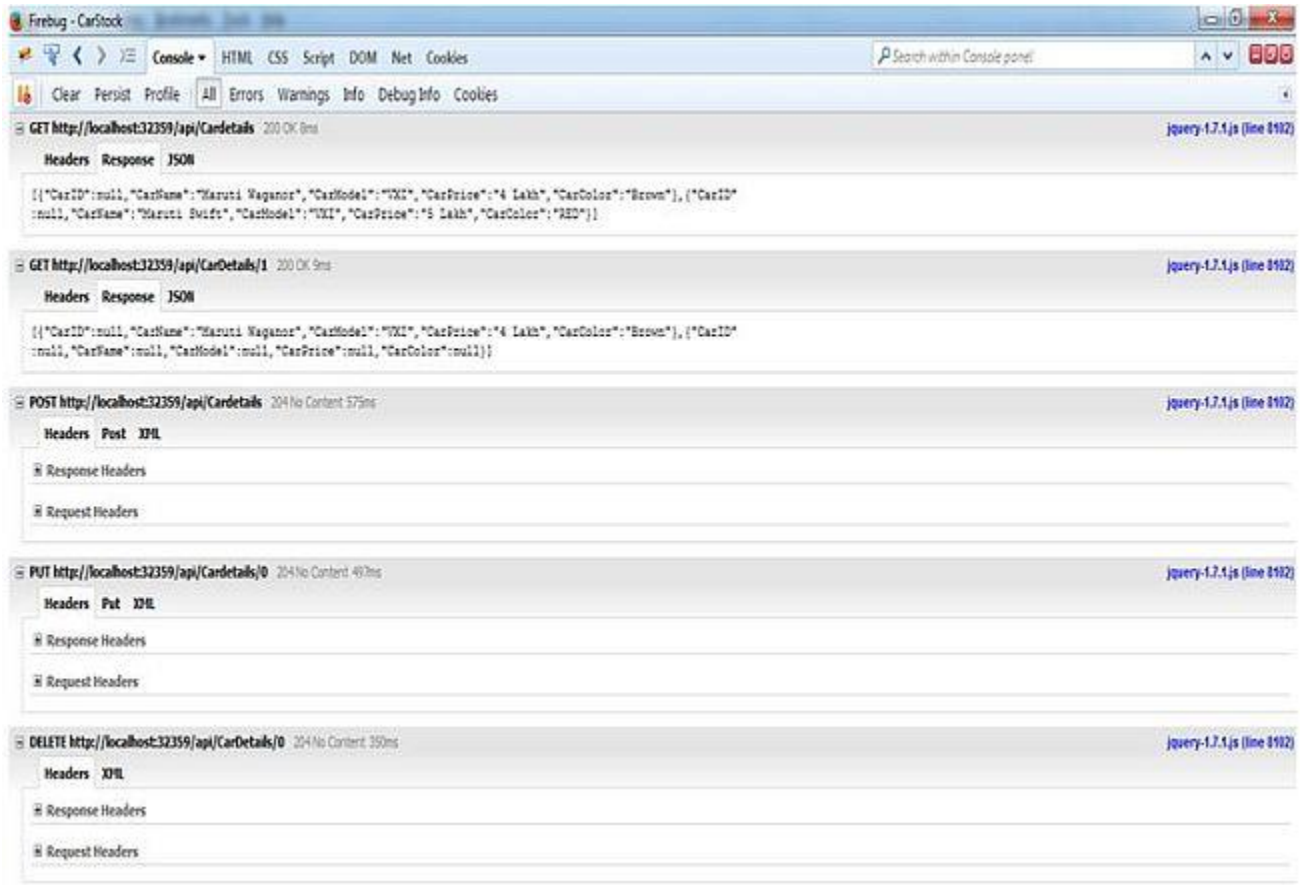
```
5.          type: 'GET',
6.          dataType: "json",
7.          success: function (data) {
8.
9.              var datavalue = data;
10.             var myJsonObject = datavalue;
11.
12.             var CarModel = myJsonObject[0].CarModel;
13.             var CarName = myJsonObject[0].CarName;
14.             var CarColor = myJsonObject[0].CarColor;
15.             var CarPrice = myJsonObject[0].CarPrice;
16.
17.             $('<tr><td>' + CarModel + '</td><td>' + CarName +
18.             '</td><td>' + CarColor + '</td>' + '</td><td>' + CarPrice + '</td></tr>')
    .appendTo('#Cartbl');
19.         },
20.         error: function (xhr) {
21.             alert(xhr.responseText);
22.         }
23.     });
24. }
```

After completing all the functions of Ajax I am now displaying the view “CarStock”.

```
@{ViewBag.Title = "CarStock";}
<h2>CarStock</h2>
<script src="~/Scripts/jquery-1.7.1.min.js"></script>
<script lang="ja" type="text/javascript">
    function GetCarById()...
    function AllcarDetails()...
    function PostData()...
    function PutData()...
    function deleteData1()...
</script>

@using (Html.BeginForm())
{
    <div style="text-align:center;">
     <table id="Cartbl" ...>...</table>
     <table border='1' width="70%" style="color: chocolate" id="Cartbl">
        <tr>
            <td>
                <input id="btnget" type="button" onclick="AllcarDetails();" value="Get_Data" />
                <input id="btngetbyid" type="button" onclick="GetCarById();" value="Get_BYID" />
                <input id="Btnpost" type="button" onclick="PostData();" value="Post_Data" />
                <input id="Btnput" type="button" onclick="PutData();" value="Put_Data" />
                <input id="BtnDelete" type="button" onclick="deleteData1();" value="Delete" />
            </td></tr>
    </table>
    </div>
}
```

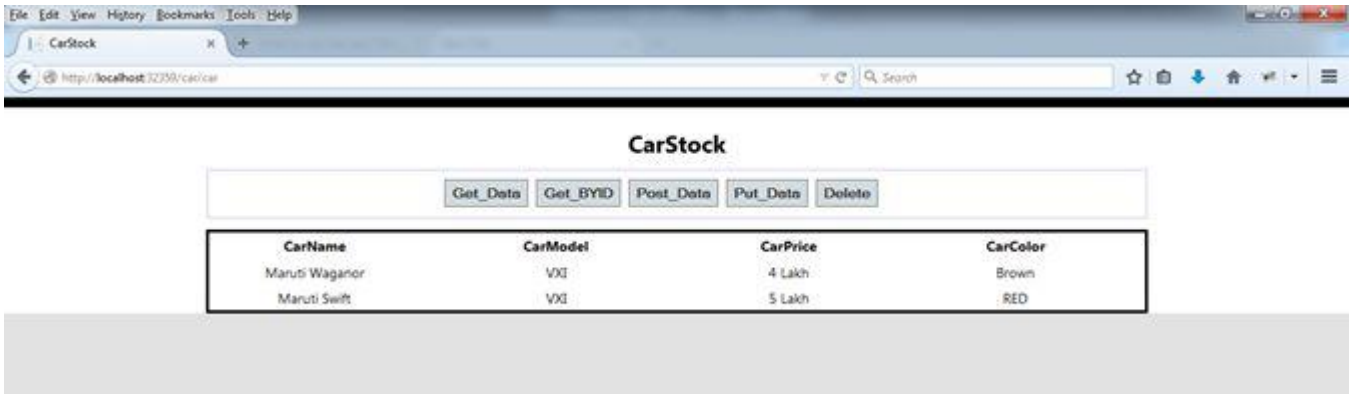
In the carstock view I have just added 5 buttons and on the click event of every button a different function of the Web API is called.

The following snippet is debugged in Firebug.



Final Output

Here in this view I have consumed a Web API for Demo.



Time Boxing

Activity Name	Activity Time	Total Time
Login Systems + Setting up Visual studio Environment	3 mints + 5 mints	8 mints
Walk through Theory & Tasks	60 mints	60 mints
Implement Tasks	80 mints	80 mints
Evaluation Time	30 mints	30 mints
	Total Duration	178 mints

Objectives

After completing this lab the student should be able to:

- Clearly understand the purpose and benefits that Cloud Computing has to offer.
- Understand the concept of Cloud Computing.
- Build a basic Cloud Computing base application.
- Use and learn Cloud Computing need.

Lab Tasks/Practical Work

1. Implement Car Stock system and search car data by car id using web api

Lab No.5

REST based CRUD operations with ASP.NET Web API

LAB # 05

REST based CRUD operations with ASP.NET Web API

Introduction

ASP.NET Web API is an ideal platform for building RESTful applications on the .NET Framework. ASP.NET Web API is a framework that makes it easy to build HTTP services that reach a broad range of clients, including browsers and mobile devices. Once we create a Web API, we can consume it in Windows, Web and Mobile devices easily. I believe that is the big advantage of Web APIs.

In this lab, we will see how to create a simple Web API with all CRUD operations and will connect with an existing MS SQL database. After that, we will create an MVC application and consume this Web API for CRUD actions.

Create "Employees" table in MSSQL database

In this Lab, we will see how to create an Employee data entry application. So, we need to create an “Employees” table first. If you have any existing database, please create this table under that database. Otherwise, create a new database as well.

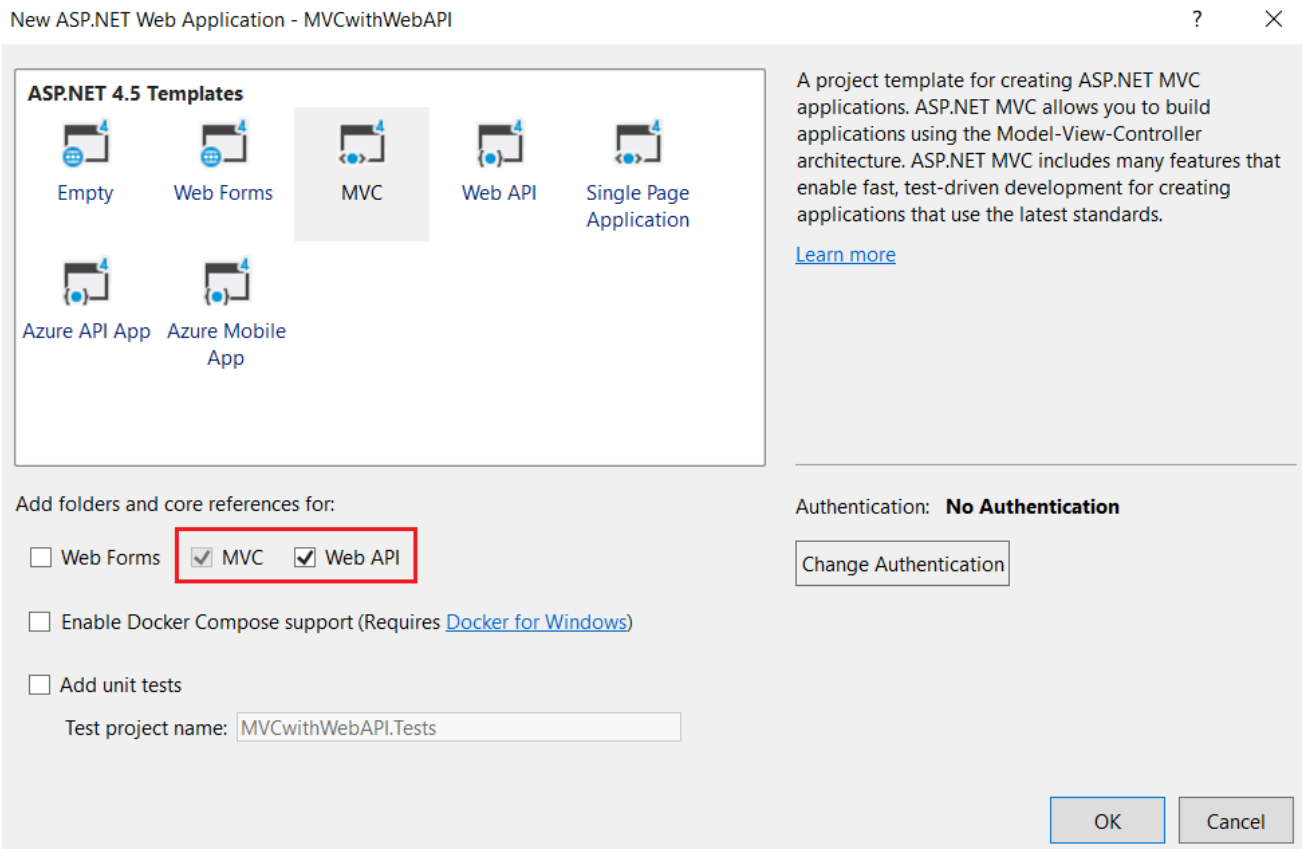
You can use below SQL statement to create a new table.

```
USE [SarathlalDB]
GO

CREATE TABLE [dbo].[Employees](
    [Id] [nvarchar](50) NOT NULL,
    [Name] [nvarchar](50) NULL,
    [Address] [nvarchar](50) NULL,
    [Gender] [nvarchar](10) NULL,
    [Company] [nvarchar](50) NULL,
    [Designation] [nvarchar](50) NULL,
 CONSTRAINT [PK_Employees] PRIMARY KEY CLUSTERED
    (
        [Id] ASC
    )
)
GO
```

Create a Visual Studio project with MVC and Web API templates

We can create a new project in VS 2015/2017. I am using a free 2017 community edition.

Please choose MVC template and choose the Web API option so, that we can get the bootstrapping for Web API in our project. No need to change any other default parameters.



Our new project will be ready in a few moments.

As I mentioned earlier, we are creating an Employee data entry application. Hence, please create an Employee class under “Models” folder.

Please note, I am creating all business classes inside this “Models” folder for simplicity. If you are following other design patterns, you can keep the files accordingly.

Employee.cs

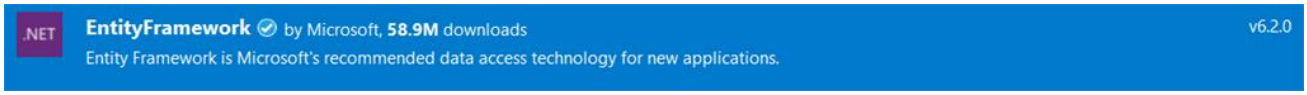
```
1.  namespace MVCwithWebAPI.Models
2.  {
3.      public class Employee
```

```
4.      {
5.          public string Id { get; set; }
6.          public string Name { get; set; }
7.          public string Address { get; set; }
8.          public string Gender { get; set; }
9.          public string Company { get; set; }
10.         public string Designation { get; set; }
11.     }
12. }
```

We can create a “DbContext” class for database connectivity.

The class that derives DbContext is called context class in entity framework. DbContext is an important class in Entity Framework API. It is a bridge between domain or entity classes and the database. DbContext is the primary class that is responsible for interacting with the database.

Before creating our DbContext class, we must install “EntityFramework” NuGet package in our project.

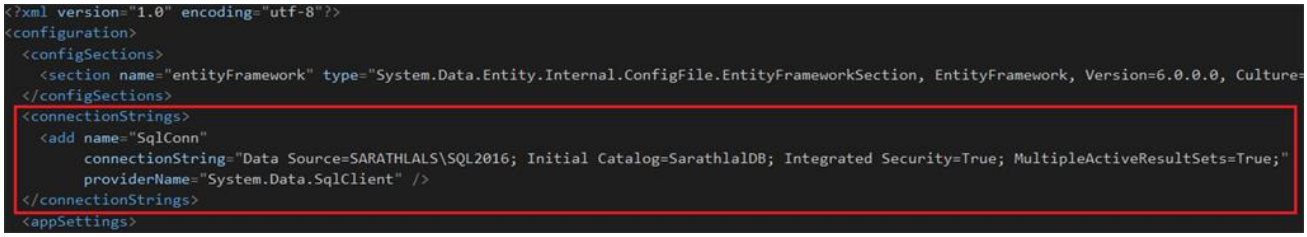


We can create “SqlDbContext” class and derives DbContext class insides this class.

SqlDbContext.cs

```
1.  using System.Data.Entity;
2.
3.  namespace MVCwithWebAPI.Models
4.  {
5.      public class SqlDbContext : DbContext
6.      {
7.          public SqlDbContext() : base("name=SqlConn")
8.          {
9.          }
10.         public DbSet<Employee> Employees { get; set; }
11.     }
12. }
```

Please note, we have used a connection “SqlConn” in above DbContext class. Hence, we can create the connection string in Web.Config file.




```
<connectionStrings>
    <add name="SqlConn"
        connectionString="Data Source=SARATHLALS\SQL2016; Initial Catalog=SarathlalDB; Integrated Security=True; MultipleActiveResultSets=True;"
        providerName="System.Data.SqlClient" />
  </connectionStrings>
```

We are following the repository pattern in this application. We can create a "IEmployeeRepository" interface and define all the functions there.

IEmployeeRepository.cs

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

namespace MVCwithWebAPI.Models
{
    public interface IEmployeeRepository
    {
        Task Add(Employee employee);
        Task Update(Employee employee);
        Task Delete(string id);
        Task<Employee> GetEmployee(string id);
        Task<IEnumerable<Employee>> GetEmployees();
    }
}
```

We can implement the exact logic for CRUD actions in "EmployeeRepository" class. We will implement IEmployeeRepository interface in this class.

EmployeeRepository.cs

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
```

```
using System.Threading.Tasks;
using System.Web;

namespace MVCwithWebAPI.Models
{
    public class EmployeeRepository : IEmployeeRepository
    {
        private readonly SqlDbContext db = new SqlDbContext();
        public async Task Add(Employee employee)
        {
            employee.Id = Guid.NewGuid().ToString();
            db.Employees.Add(employee);
            try
            {
                await db.SaveChangesAsync();
            }
            catch
            {
                throw;
            }
        }
        public async Task<Employee> GetEmployee(string id)
        {
            try
            {
                Employee employee = await db.Employees.FindAsync(id);
                if (employee == null)
                {
                    return null;
                }
                return employee;
            }
            catch
            {
                throw;
            }
        }
        public async Task<IEnumerable<Employee>> GetEmployees()
        {
            try
            {
                var employees = await db.Employees.ToListAsync();
                return employees.AsQueryable();
```

```
48.         }
49.         catch
50.         {
51.             throw;
52.         }
53.     }
54.     public async Task Update(Employee employee)
55.     {
56.         try
57.         {
58.             db.Entry(employee).State = EntityState.Modified;
59.             await db.SaveChangesAsync();
60.         }
61.         catch
62.         {
63.             throw;
64.         }
65.     }
66.     public async Task Delete(string id)
67.     {
68.         try
69.         {
70.             Employee employee = await db.Employees.FindAsync(id);
71.             db.Employees.Remove(employee);
72.             await db.SaveChangesAsync();
73.         }
74.         catch
75.         {
76.             throw;
77.         }
78.     }
79.
80.     private bool EmployeeExists(string id)
81.     {
82.         return db.Employees.Count(e => e.Id == id) > 0;
83.     }
84.
85.   }
86. }
```

I have implemented all 5 methods (for CRUD) in this class. All are self-explanatory. If you need further clarification on any terms, please feel free to contact me.

We can create our API Controller now.

EmployeesApiController.cs

```
1.  using MVCwithWebAPI.Models;
2.  using System.Collections.Generic;
3.  using System.Threading.Tasks;
4.  using System.Web.Http;
5.
6.  namespace MVCwithWebAPI.Controllers
7.  {
8.      public class EmployeesApiController : ApiController
9.      {
10.         private readonly IEmployeeRepository _iEmployeeRepository = new EmployeeRe
    pository();
11.
12.         [HttpGet]
13.         [Route("api/Employees/Get")]
14.         public async Task<IEnumerable<Employee>> Get()
15.         {
16.             return await _iEmployeeRepository.GetEmployees();
17.         }
18.
19.         [HttpPost]
20.         [Route("api/Employees/Create")]
21.         public async Task CreateAsync([FromBody]Employee employee)
22.         {
23.             if (ModelState.IsValid)
24.             {
25.                 await _iEmployeeRepository.Add(employee);
26.             }
27.         }
28.
29.         [HttpGet]
30.         [Route("api/Employees/Details/{id}")]
31.         public async Task<Employee> Details(string id)
32.         {
33.             var result = await _iEmployeeRepository.GetEmployee(id);
34.             return result;
35.         }
36.
37.         [HttpPut]
38.         [Route("api/Employees/Edit")]
```

```
39.     public async Task EditAsync([FromBody]Employee employee)
40.     {
41.         if (ModelState.IsValid)
42.         {
43.             await _iEmployeeRepository.Update(employee);
44.         }
45.     }
46.
47.     [HttpDelete]
48.     [Route("api/Employees/Delete/{id}")]
49.     public async Task DeleteConfirmedAsync(string id)
50.     {
51.         await _iEmployeeRepository.Delete(id);
52.     }
53.   }
54. }
```

All the CRUD actions are derived in this API class. We have created an instance for EmployeeRepository class and with the help of this instance, we have accessed all the methods from EmployeeRepository class in our API class.

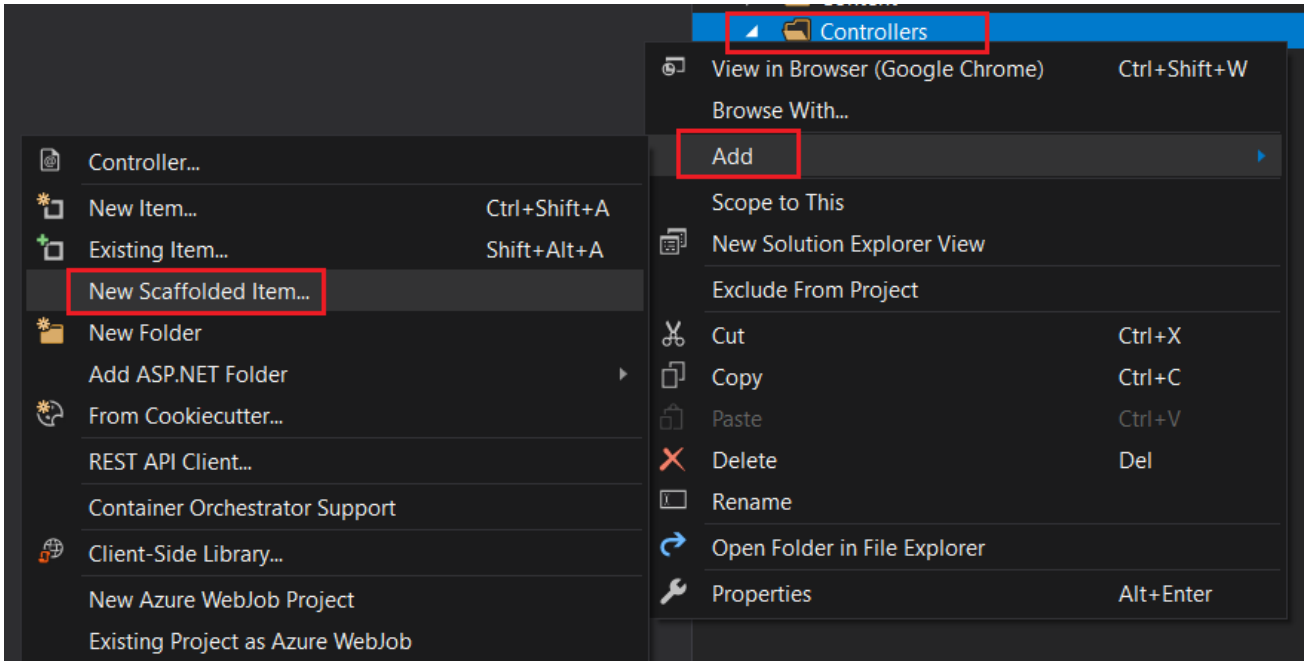
We have completed our Web API design. If needed, you can check the Web API using Postman or any other tool. Please note down the URL of the application. We need to add this base URL in our Web.Config file because we will use this base URL in MVC controllers.
We can create a new key-value pair in Web.Config file under “appSettings” section.

```
<appSettings>
  <add key="webpages:Version" value="3.0.0.0" />
  <add key="webpages:Enabled" value="false" />
  <add key="ClientValidationEnabled" value="true" />
  <add key="UnobtrusiveJavaScriptEnabled" value="true" />
  <add key="apiBaseAddress" value="http://localhost:2019/api/"/>
</appSettings>
<system.web>
```

We have named the key as “apiBaseAddress” and gave the project URL as value.

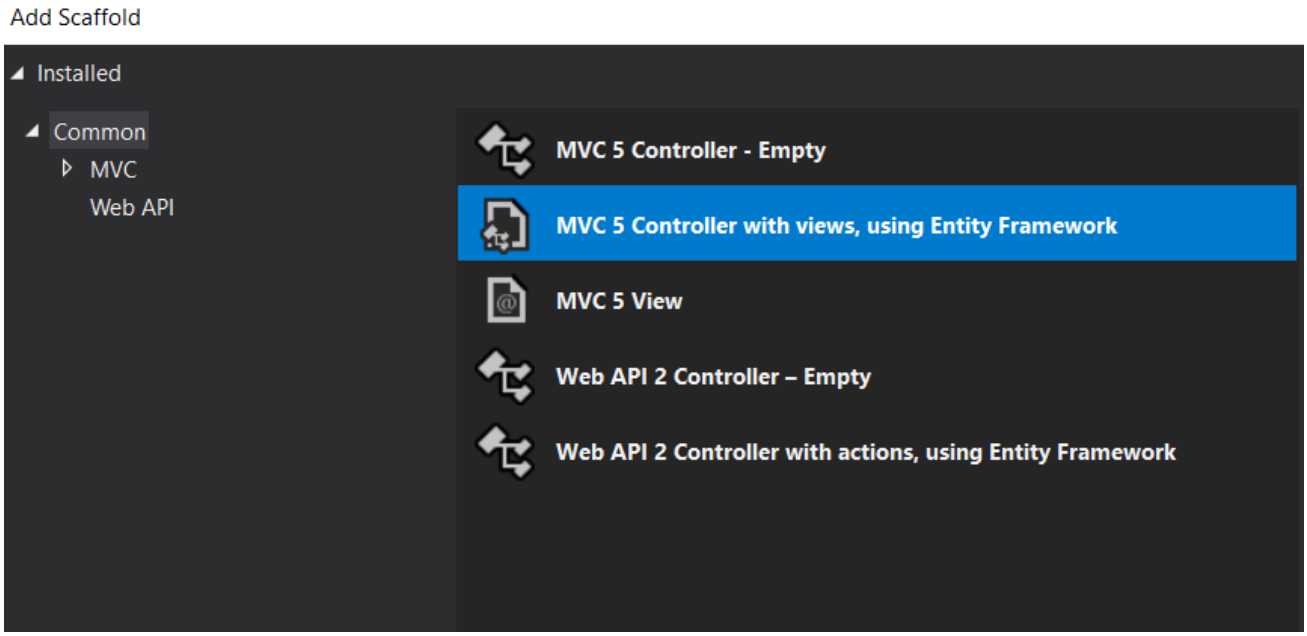
Create the MVC Controller using Scaffolding

We can create the MVC Controller using scaffolding template so that we will get all the views for CRUD actions easily. We will modify these views later.

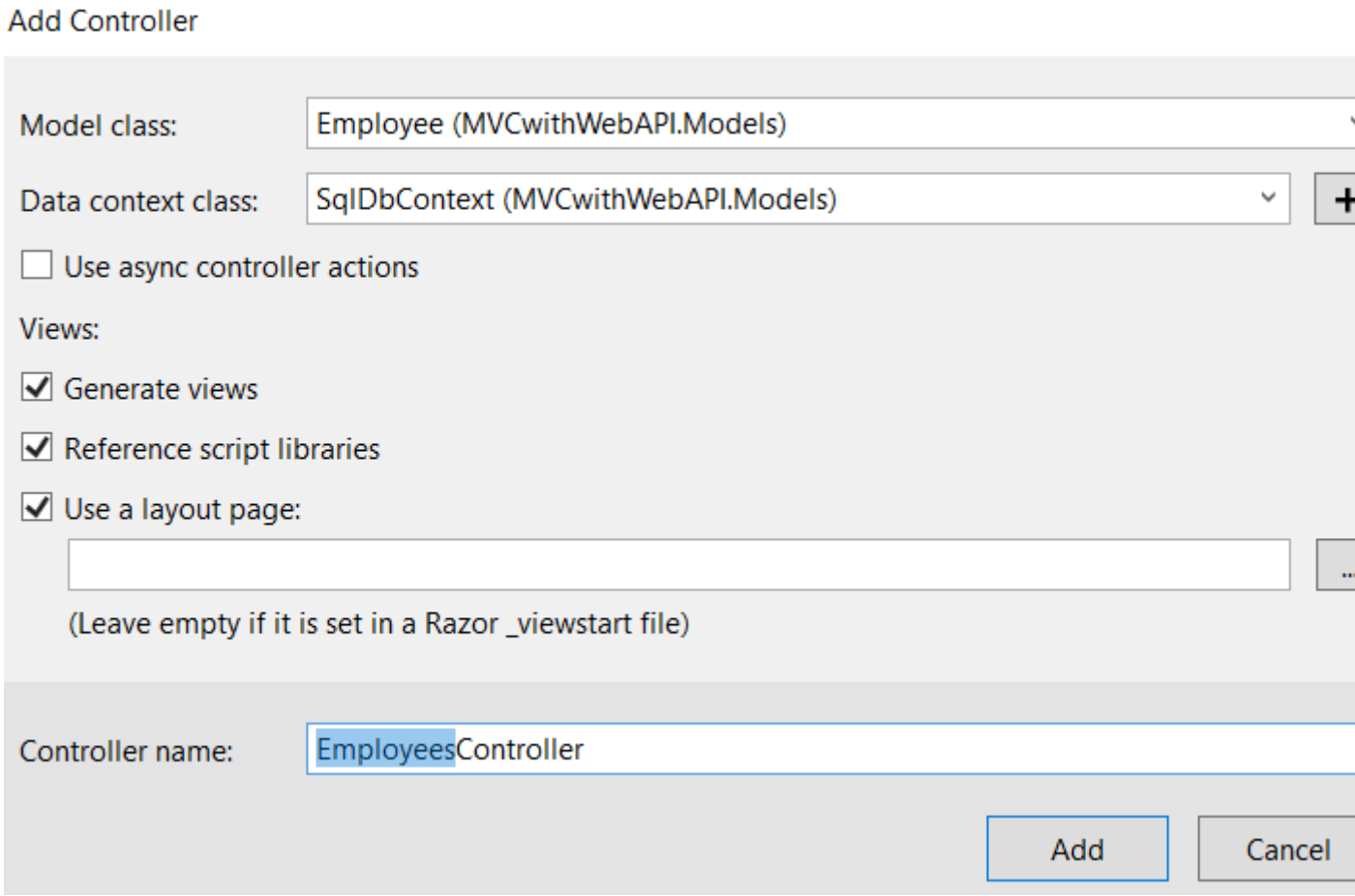


We can right-click the "Controller" folder and click "Add" and choose "New Scaffolded Item" to create a new MVC controller.

You can choose, "MVC 5 Controller with views, using Entity Framework" option. This will create all views for CRUD operations.

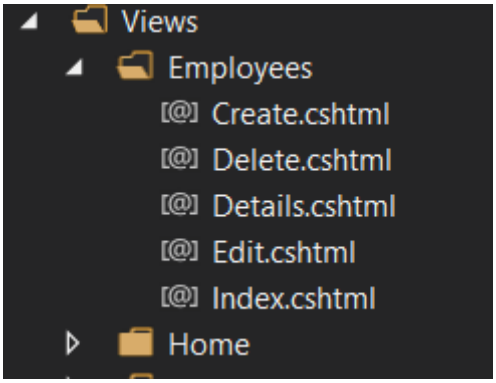


We can choose the model class, data context class and give controller name as well. Please note, the system will automatically give a name for a controller. If needed, you can change it.



You can click the “Add” button to proceed further. After a few moments, our Employees controller is created with all views.

You can see the views under “Views” -> “Employees” folder.



We will modify some of the view files later. Let us modify the "EmployeesController" class now. Please note, by default MVC controller does not create actions as async. We have converted all these actions asynchronously. You can copy the below code and paste it inside the controller class.

EmployeesController.cs

```
1.  using MVCwithWebAPI.Models;
2.  using System;
3.  using System.Collections.Generic;
4.  using System.Configuration;
5.  using System.Linq;
6.  using System.Net;
7.  using System.Net.Http;
8.  using System.Threading.Tasks;
9.  using System.Web.Mvc;
10.
11. namespace MVCwithWebAPI.Controllers
12. {
13.     public class EmployeesController : Controller
14.     {
15.         readonly string apiBaseAddress = ConfigurationManager.AppSettings["apiBaseAd
    dress"];
16.         public async Task<ActionResult> Index()
17.         {
18.             IEnumerable<Employee> employees = null;
19.
20.             using (var client = new HttpClient())
21.             {
22.                 client.BaseAddress = new Uri(apiBaseAddress);
23.
24.                 var result = await client.GetAsync("employees/get");
```

```
25.
26.             if (result.IsSuccessStatusCode)
27.             {
28.                 employees = await result.Content.ReadAsAsync<IList<Employee>>();
29.             }
30.             else
31.             {
32.                 employees = Enumerable.Empty<Employee>();
33.                 ModelState.AddModelError(string.Empty, "Server error try after some time.
    ");
34.             }
35.         }
36.         return View(employees);
37.     }
38.
39.     public async Task<ActionResult> Details(string id)
40.     {
41.         if (id == null)
42.         {
43.             return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
44.         }
45.
46.         Employee employee = null;
47.         using (var client = new HttpClient())
48.         {
49.             client.BaseAddress = new Uri(apiBaseAddress);
50.
51.             var result = await client.GetAsync($"employees/details/{id}");
52.
53.             if (result.IsSuccessStatusCode)
54.             {
55.                 employee = await result.Content.ReadAsAsync<Employee>();
56.             }
57.             else
58.             {
59.                 ModelState.AddModelError(string.Empty, "Server error try after some time.
    ");
60.             }
61.         }
62.
63.         if (employee == null)
64.         {
65.             return HttpNotFound();
```

```
66.             }
67.             return View(employee);
68.         }
69.
70.         public ActionResult Create()
71.         {
72.             return View();
73.         }
74.
75.         [HttpPost]
76.         [ValidateAntiForgeryToken]
77.         public async Task<ActionResult> Create([Bind(Include = "Name,Address,Gender,C
    ompany,Designation")] Employee employee)
78.         {
79.             if (ModelState.IsValid)
80.             {
81.                 using (var client = new HttpClient())
82.                 {
83.                     client.BaseAddress = new Uri(apiBaseAddress);
84.
85.                     var response = await client.PostAsJsonAsync("employees/Create", employee
    );
86.                     if (response.IsSuccessStatusCode)
87.                     {
88.                         return RedirectToAction("Index");
89.                     }
90.                     else
91.                     {
92.                         ModelState.AddModelError(string.Empty, "Server error try after some ti
    me.");
93.                     }
94.                 }
95.             }
96.             return View(employee);
97.         }
98.
99.         public async Task<ActionResult> Edit(string id)
100.                {
101.                    if (id == null)
102.                    {
103.                        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
104.                    }
105.                    Employee employee = null;
```

```
106.                using (var client = new HttpClient())
107.                {
108.                    client.BaseAddress = new Uri(apiBaseAddress);
109.
110.                    var result = await client.GetAsync($"employees/details/{id}");
111.
112.                    if (result.IsSuccessStatusCode)
113.                    {
114.                        employee = await result.Content.ReadAsAsync<Employee>();
115.                    }
116.                    else
117.                    {
118.                        ModelState.AddModelError(string.Empty, "Server error try after so
    me time.");
119.                    }
120.                }
121.                if (employee == null)
122.                {
123.                    return HttpNotFound();
124.                }
125.                return View(employee);
126.            }
127.
128.            [HttpPost]
129.            [ValidateAntiForgeryToken]
130.            public async Task<ActionResult> Edit([Bind(Include = "Id,Name,Address,G
    ender,Company,Designation")] Employee employee)
131.            {
132.                if (ModelState.IsValid)
133.                {
134.                    using (var client = new HttpClient())
135.                    {
136.                        client.BaseAddress = new Uri(apiBaseAddress);
137.                        var response = await client.PutAsJsonAsync("employees/edit", emplo
    yee);
138.                        if (response.IsSuccessStatusCode)
139.                        {
140.                            return RedirectToAction("Index");
141.                        }
142.                        else
143.                        {
144.                            ModelState.AddModelError(string.Empty, "Server error try after s
    ome time.");
```

```
145.                     }
146.                 }
147.                 return RedirectToAction("Index");
148.             }
149.             return View(employee);
150.         }
151.
152.         public async Task<ActionResult> Delete(string id)
153.         {
154.             if (id == null)
155.             {
156.                 return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
157.             }
158.             Employee employee = null;
159.             using (var client = new HttpClient())
160.             {
161.                 client.BaseAddress = new Uri(apiBaseAddress);
162.
163.                 var result = await client.GetAsync($"employees/details/{id}");
164.
165.                 if (result.IsSuccessStatusCode)
166.                 {
167.                     employee = await result.Content.ReadAsAsync<Employee>();
168.                 }
169.                 else
170.                 {
171.                     ModelState.AddModelError(string.Empty, "Server error try after so
    me time.");
172.                 }
173.             }
174.
175.             if (employee == null)
176.             {
177.                 return HttpNotFound();
178.             }
179.             return View(employee);
180.         }
181.
182.         [HttpPost, ActionName("Delete")]
183.         [ValidateAntiForgeryToken]
184.         public async Task<ActionResult> DeleteConfirmed(string id)
185.         {
186.             using (var client = new HttpClient())
```

```
187.                 {
188.                     client.BaseAddress = new Uri(apiBaseAddress);
189.
190.                     var response = await client.DeleteAsync($"employees/delete/{id}");
191.                     if (response.IsSuccessStatusCode)
192.                     {
193.                         return RedirectToAction("Index");
194.                     }
195.                     else
196.                         ModelState.AddModelError(string.Empty, "Server error try after so
    me time.");
197.                 }
198.                 return View();
199.             }
200.
201.         }
202.     }
```

You can see, we have defined an “apiBaseAddress” variable globally and got the value for apiBaseAddress from Web.Config file. We will use this value in all our controller actions.

Index Action

```
1.  public async Task<ActionResult> Index()
2.      {
3.          IEnumerable<Employee> employees = null;
4.
5.          using (var client = new HttpClient())
6.          {
7.              client.BaseAddress = new Uri(apiBaseAddress);
8.
9.              var result = await client.GetAsync("employees/get");
10.
11.             if (result.IsSuccessStatusCode)
12.             {
13.                 employees = await result.Content.ReadAsAsync<IList<Employee>>();
14.             }
15.             else
16.             {
17.                 employees = Enumerable.Empty<Employee>();
18.                 ModelState.AddModelError(string.Empty, "Server error try after some time.
    ");
19.             }
```

```
            }
            return View(employees);
        }
```

If you look at the index action, you can see, we have declared a “HttpClient” variable and using client.GetAsync method, we have got the employee data result from Web API and store in a “result” variable. We have again read the employee data from this result variable using “ReadAsync” method.

We have used the same approach in other action methods also. All the methods are self-explanatory. If you have any queries, please feel free to contact me.

We can now modify the Index view. Please copy the below and paste to Index view file.

Index.cshtml

```
@model IEnumerable<MVCwithWebAPI.Models.Employee>

@{
    ViewBag.Title = "Employee List";
}

<h3>Employee List</h3>

<p>
    @Html.ActionLink("New Employee", "Create")
</p>
<table class="table">
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.Name)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Address)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Gender)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Company)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Designation)
```

```
        </th>
        <th></th>
    </tr>

    @foreach (var item in Model)
    {
        <tr>
            <td>
                @Html.ActionLink(item.Name, "Details", new { id = item.Id })
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Address)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Gender)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Company)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Designation)
            </td>
            <td>
                @Html.ActionLink("Edit", "Edit", new { id = item.Id }) |
                @Html.ActionLink("Delete", "Delete", new { id = item.Id })
            </td>
        </tr>
    }

</table>
```

We have modified the existing “Index” view. We have removed the “Details” link from this view and instead, we have given a hyperlink in the employee name itself for details.

We can modify the “Create” view by removing the Id field. Because for us, employee Id will be created automatically while inserting new data. We have used system GUID for this.

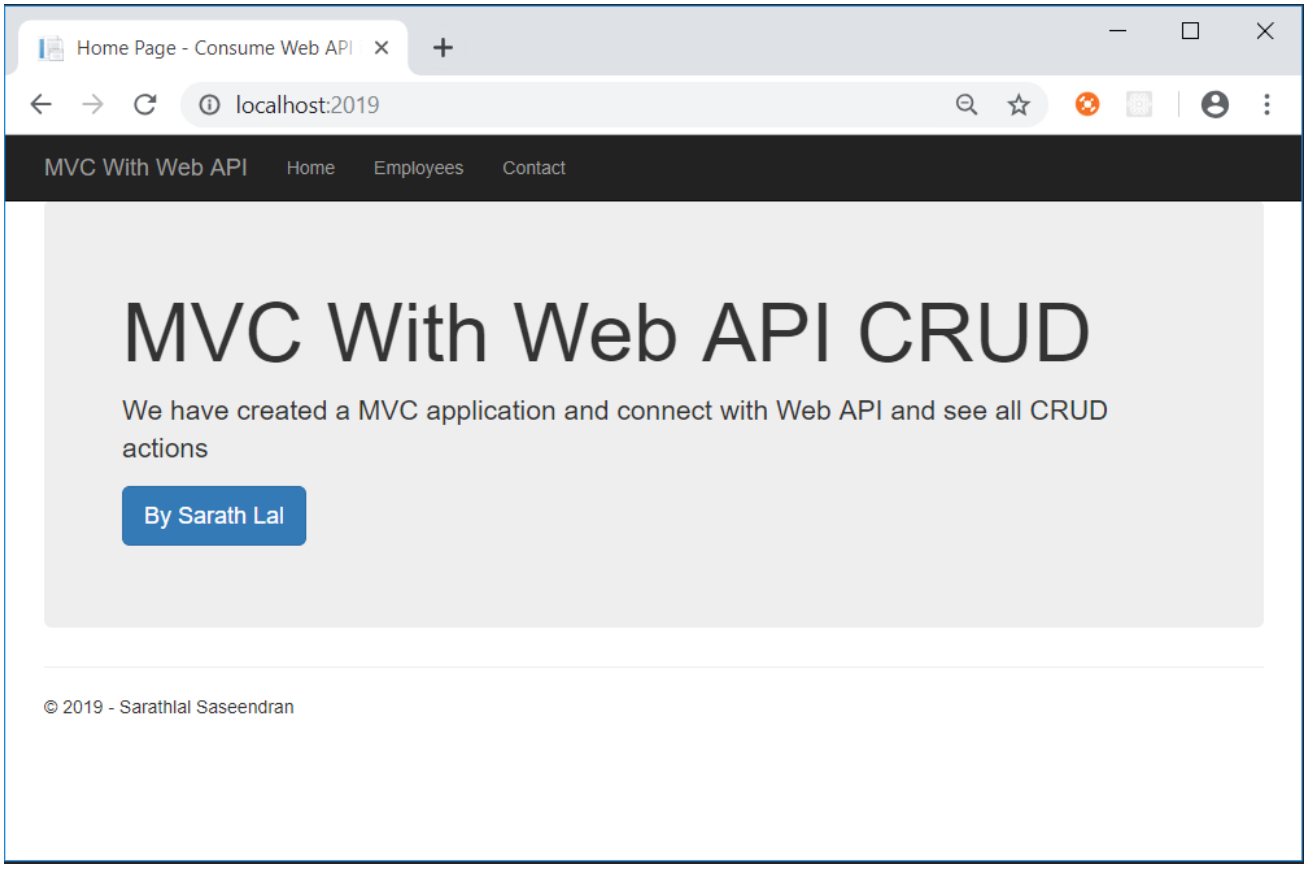
Create.cshtml

```
@model MVCwithWebAPI.Models.Employee

@{
    ViewBag.Title = "Create Employee";
```

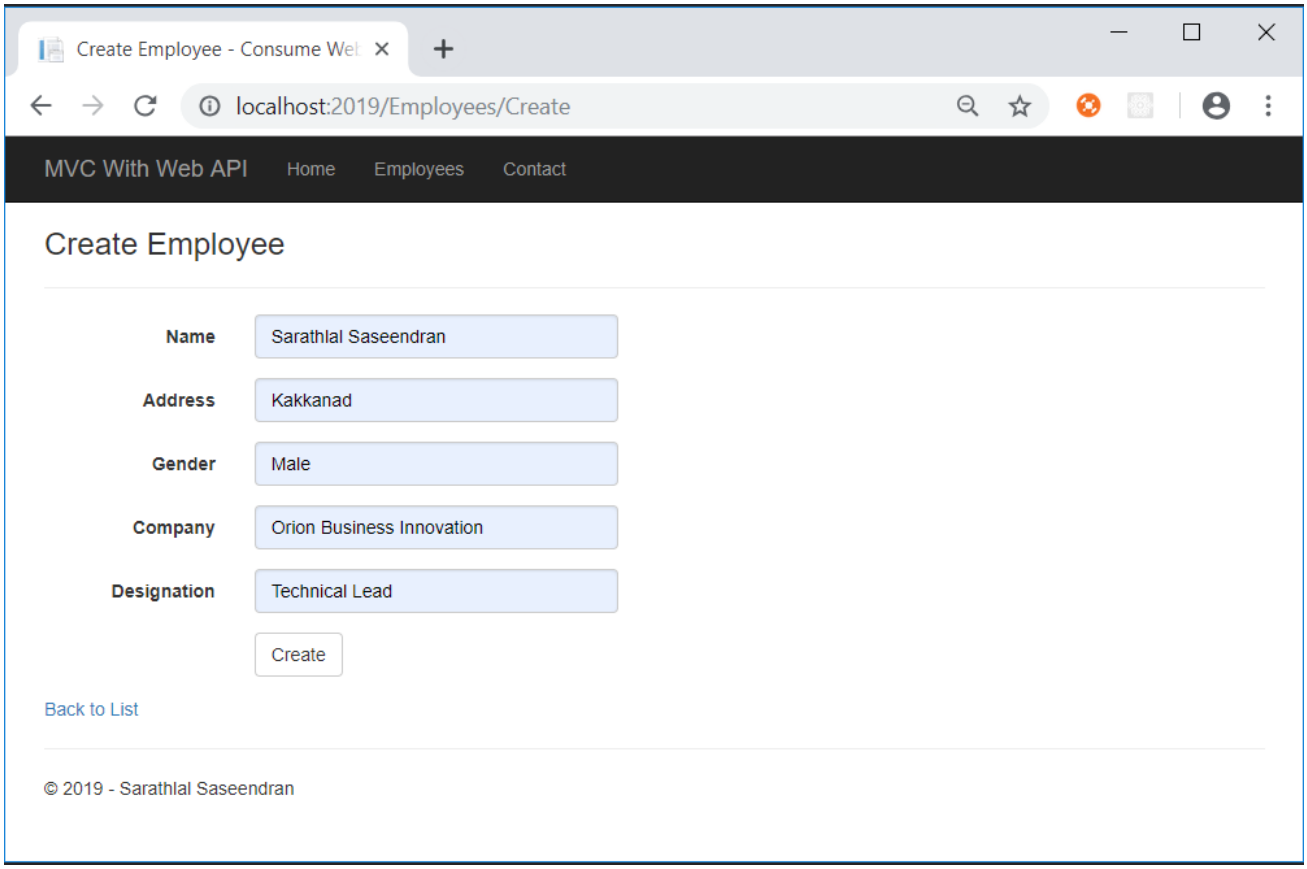
```
5.  }
6.
7.  <h3>Create Employee</h3>
8.
9.
10. @using (Html.BeginForm())
11. {
12.     @Html.AntiForgeryToken()
13.
14.     <div class="form-horizontal">
15.         <hr />
16.         @Html.ValidationSummary(true, "", new { @class = "text-danger" })
17.
18.         <div class="form-group">
19.             @Html.LabelFor(model => model.Name, htmlAttributes: new { @class = "contro
    l-label col-md-2" })
20.             <div class="col-md-10">
21.                 @Html.EditorFor(model => model.Name, new { htmlAttributes = new { @clas
    s = "form-control" } })
22.                 @Html.ValidationMessageFor(model => model.Name, "", new { @class = "tex
    t-danger" })
23.             </div>
24.         </div>
25.
26.         <div class="form-group">
27.             @Html.LabelFor(model => model.Address, htmlAttributes: new { @class = "cont
    rol-label col-md-2" })
28.             <div class="col-md-10">
29.                 @Html.EditorFor(model => model.Address, new { htmlAttributes = new { @cl
    ass = "form-control" } })
30.                 @Html.ValidationMessageFor(model => model.Address, "", new { @class = "t
    ext-danger" })
31.             </div>
32.         </div>
33.
34.         <div class="form-group">
35.             @Html.LabelFor(model => model.Gender, htmlAttributes: new { @class = "contr
    ol-label col-md-2" })
36.             <div class="col-md-10">
37.                 @Html.EditorFor(model => model.Gender, new { htmlAttributes = new { @cl
    ass = "form-control" } })
38.                 @Html.ValidationMessageFor(model => model.Gender, "", new { @class = "t
    ext-danger" })
```

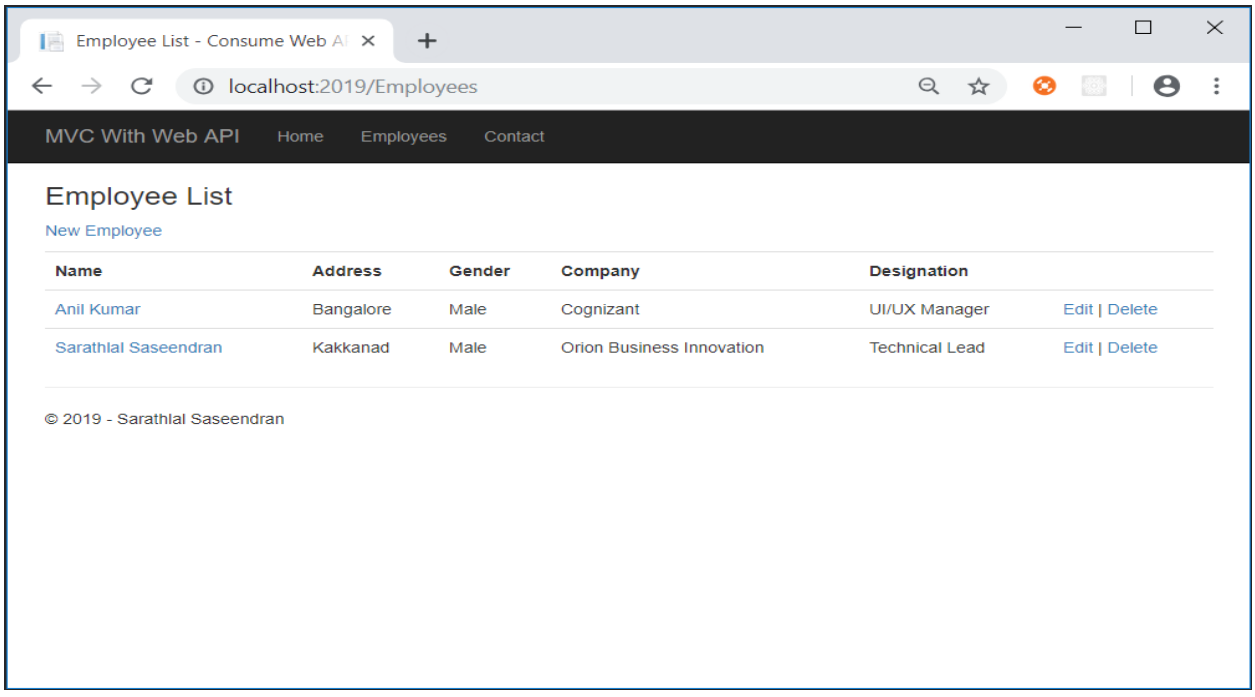
```
39.             </div>
40.         </div>
41.
42.         <div class="form-group">
43.             @Html.LabelFor(model => model.Company, htmlAttributes: new { @class = "co
    ntrol-label col-md-2" })
44.             <div class="col-md-10">
45.                 @Html.EditorFor(model => model.Company, new { htmlAttributes = new { @
    class = "form-control" } })
46.                 @Html.ValidationMessageFor(model => model.Company, "", new { @class =
    "text-danger" })
47.             </div>
48.         </div>
49.
50.         <div class="form-group">
51.             @Html.LabelFor(model => model.Designation, htmlAttributes: new { @class = "
    control-label col-md-2" })
52.             <div class="col-md-10">
53.                 @Html.EditorFor(model => model.Designation, new { htmlAttributes = new {
    @class = "form-control" } })
54.                 @Html.ValidationMessageFor(model => model.Designation, "", new { @class
    = "text-danger" })
55.             </div>
56.         </div>
57.
58.         <div class="form-group">
59.             <div class="col-md-offset-2 col-md-10">
60.                 <input type="submit" value="Create" class="btn btn-default" />
61.             </div>
62.         </div>
63.     </div>
64. }
65.
66. <div>
67.     @Html.ActionLink("Back to List", "Index")
68. </div>
69.
70. @section Scripts {
71.     @Scripts.Render("~/bundles/jqueryval")
72. }
```

We can run the application now. The landing page looks like the below screenshot.



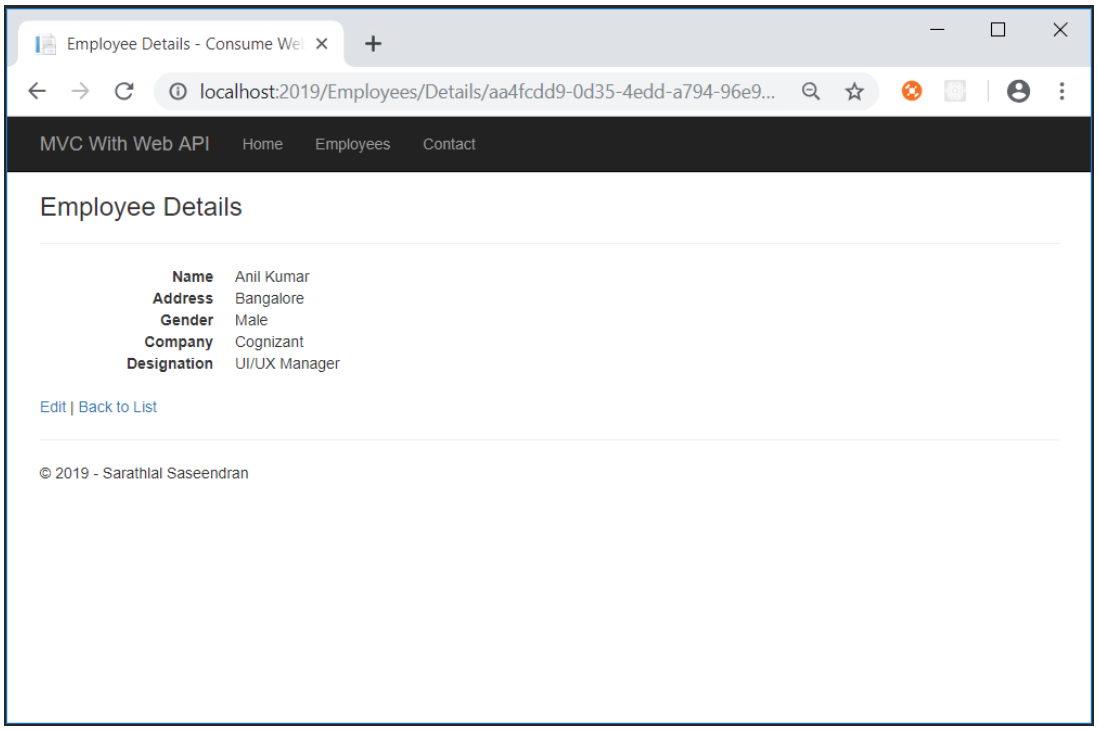
We can click the “Employees” link and click the “New Employee” link to create a new employee.


Create Employee - Consume Wel
localhost:2019/Employees/Create
MVC With Web API
Home
Employees
Contact
Create Employee
Name
Sarathlal Saseendran
Address
Kakkanad
Gender
Male
Company
Orion Business Innovation
Designation
Technical Lead
Create
Back to List
© 2019 - Sarathlal Saseendran

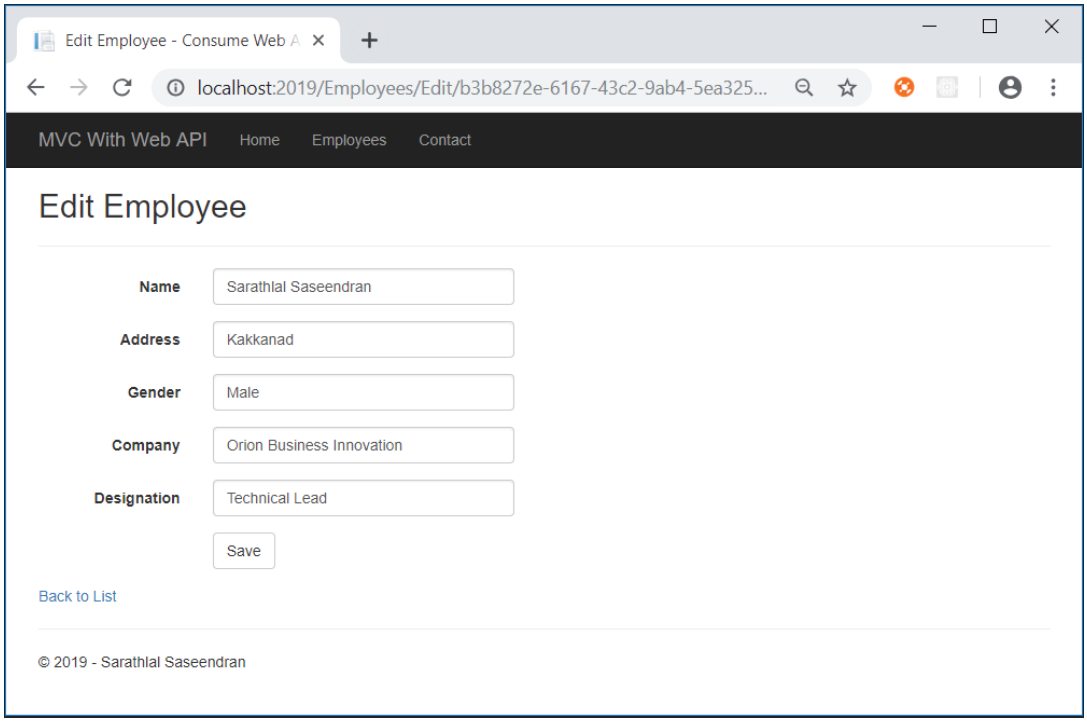

Employee List - Consume Web Al
localhost:2019/Employees
MVC With Web API
Home
Employees
Contact
Employee List
New Employee
Name
Address
Gender
Company
Designation
Anil Kumar
Bangalore
Male
Cognizant
UI/UX Manager
Edit | Delete
Sarathlal Saseendran
Kakkanad
Male
Orion Business Innovation
Technical Lead
Edit | Delete
© 2019 - Sarathlal Saseendran

As I mentioned earlier in this Lab, I have removed the default “Details” link from this view and added the hyperlink in employee name itself.

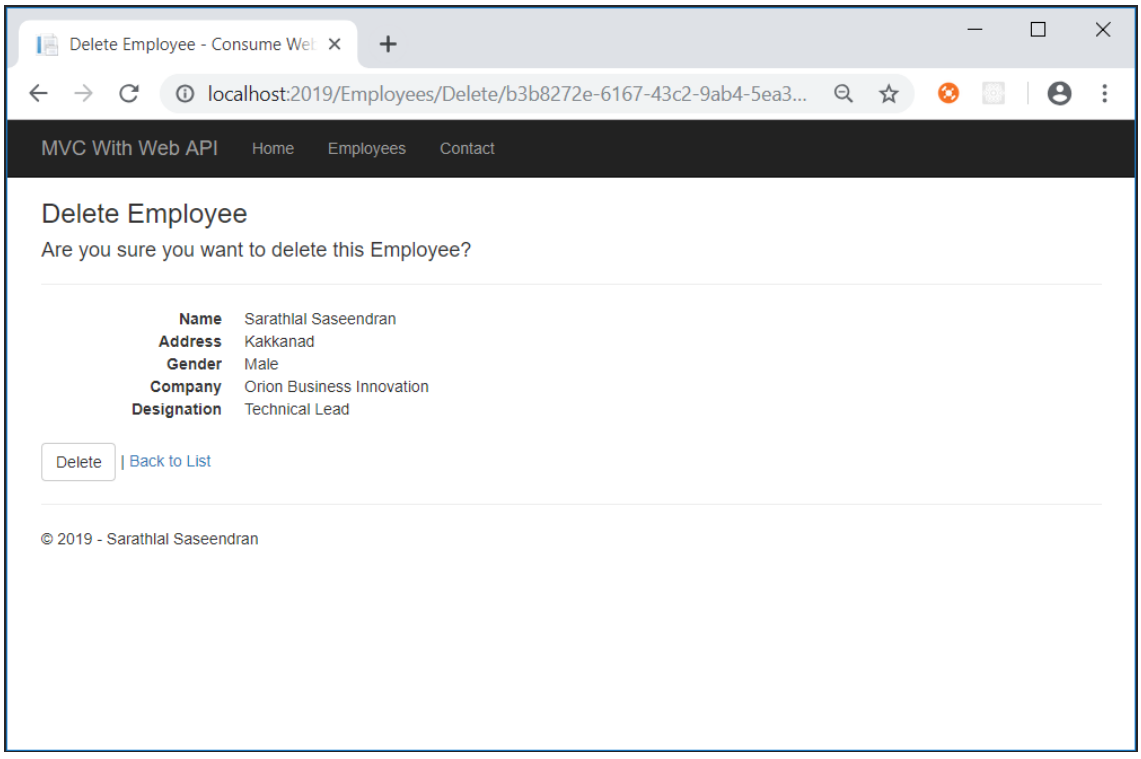
You can click the employee name to show the details.



We can click “Edit” link to edit the employee details.



We can use “Delete” link to delete the record as well.



We have successfully seen all the CRUD actions with this application.

Time Boxing

Activity Name	Activity Time	Total Time
Login Systems + Setting up Visual studio Environment	3 mints + 5 mints	8 mints
Walk through Theory & Tasks	60 mints	60 mints
Implement Tasks	80 mints	80 mints
Evaluation Time	30 mints	30 mints
	Total Duration	178 mints

Objectives

After completing this lab the student should be able to:

- Clearly understand the purpose and benefits that Cloud Computing has to offer.
- Understand the concept of Cloud Computing.
- Build a basic Cloud Computing base application.
- Use and learn Cloud Computing need.

Lab Tasks/Practical Work

1. Perform crud operation with mvc web api on employee form..

Lab No. 6

ASP.NET Web Service Basics

LAB # 06

ASP.NET Web Service Basics

We can now use ASP.NET to create Web Services based on industrial standards including XML, SOAP, and WSDL.

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

A web service is

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

Web Service History

- Microsoft coined the term "Web services" in June 2000, when the company introduced Web services as a key component of its .Net initiative, a broad new vision for embracing the Internet in the development, engineering and use of software.
- As others began to investigate Web services, it became clear that the technology could revolutionize (be the next stage in) distributed computing.
- Web services encom a set of related standards that can enable any two computers to communicate and exchange data via a network, such as the Internet.
- The primary standard used in Web services is the Extensible Markup Language (XML) developed by the World Wide Web Consortium (W3C).
- Developers use XML tags to describe individual pieces of data, forming XML documents, which are text-based and can be processed on any platform.
- XML provides the foundation for many core Web services standards (SOAP, WSDL, and UDDI) and vocabularies (XML-based markup for a specific industry or purpose).

- Almost every type of business can benefit from Web services such as expediting software development, integrating applications and databases, and automating transactions with suppliers, partners, and clients.

Key Web Service Technologies

- **XML**- Describes only data. So, any application that understands XML-regardless of the application's programming language or platform has the ability to format XML in a variety of ways (well-formed or valid).
- **SOAP**- Provides a communication mechanism between services and applications.
- **WSDL**- Offers a uniform method of describing web services to other programs.
- **UDDI**- Enables the creation of searchable Web services registries.

When these technologies are deployed together, they allow developers to package applications as services and publish those services on a network.

Web services advantages

- Use open, text-based standards, which enable components written in various languages and for different platforms to communicate.
- Promote a modular approach to programming, so multiple organizations can communicate with the same Web service.
- Comparatively easy and inexpensive to implement, because they employ an existing infrastructure and because most applications can be repackaged as Web services.
- Significantly reduce the costs of enterprise application (EAI) integration and B2B communications.
- Implemented incrementally, rather than all at once which lessens the cost and reduces the organizational disruption from an abrupt switch in technologies.
- The Web Services Interoperability Organization (WS-I) consisting of over 100 vendors promotes interoperability.

Web Services Limitations

- SOAP, WSDL, UDDI- require further development.
- Interoperability.
- Royalty fees.
- Too slow for use in high-performance situations.
- Increase traffic on networks.
- The lack of security standards for Web services.
- The standard procedure for describing the quality (i.e. levels of performance, reliability, security etc.) of particular Web services – management of Web services.
- The standards that drive Web services are still in draft form (always will be in refinement).

- Some vendors want to retain their intellectual property rights to certain Web services standards.

Web Service Example

A web service can perform almost any kind of task.

- **Web Portal-** A web portal might obtain top news headlines from an Associated press web service.
- Weather Reporting- You can use Weather Reporting web service to display weather information in your personal website.
- Stock Quote- You can display latest update of Share market with Stock Quote on your web site.
- News Headline: You can display latest news update by using News Headline Web Service in your website.
- You can make your own web service and let others use it. For example you can make Free SMS Sending Service with footer with your company's advertisement, so whosoever uses this service indirectly advertises your company. You can apply your ideas in N no. of ways to take advantage of it.

Time Boxing

Activity Name	Activity Time	Total Time
Login Systems + Setting up Visual studio Environment	3 mints + 5 mints	8 mints
Walk through Theory & Tasks	60 mints	60 mints
Implement Tasks	80 mints	80 mints
Evaluation Time	30 mints	30 mints
	Total Duration	178 mints

Objectives

After completing this lab the student should be able to:

- Clearly understand the purpose and benefits that Cloud Computing has to offer.
- Understand the concept of Cloud Computing.
- Build a basic Cloud Computing base application.
- Use and learn Cloud Computing need.

Lab Tasks/Practical Work

1. Create a web service of online add to cart system
2. Create a simple calculator web service to perform task

Lab No. 7

Consuming Web services using ASP.Net

LAB # 07

Consuming Web services using ASP.Net

What does consuming mean?

Many beginners are confused about what means consuming means, but its very simple, it means to use the Web Services in an application.

Example

I have created Web Services and now I want to use it in a real requirement so I used it in an ASP.Net Web Application. In other words, I am consuming the Web Service in an ASP.Net web application. Similarly you can use the same Web Service in Windows, console application, Java and other applications.

I hope you understand the word "consuming".

So let us create the simple ASP.Net Web Application as:

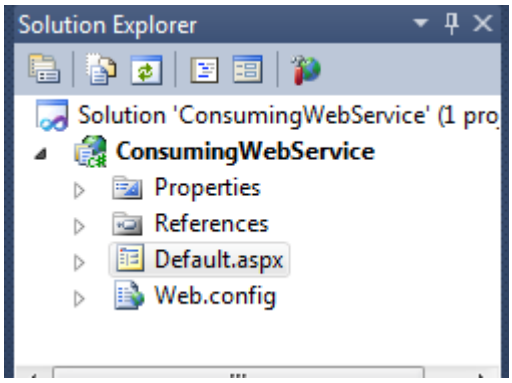
1. "Start" - "All Programs" - "Microsoft Visual Studio 2010".
2. "File" - "New" - "Project..." then in the New Project window "C#" - "Empty Project" (to avoid adding a master page).
3. Give the project a name, such as "Consumingwebservice" or another as you wish and specify the location.
4. Then right-click on Solution Explorer and select "Add New Item" - "Default.aspx" page.
5. Then drag three TextBoxes, one Button, and one Label onto the "Default.aspx" page.

Then the <body> section of the Default.aspx page looks as in the following:

```
<body>

    <form id="form1" runat="server">
    <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
    <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>
     <asp:TextBox ID="TextBox3" runat="server"></asp:TextBox>
    <asp:Button ID="Button1" runat="server" Text="Calculate" />
    <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
    </form>

</body>
```

I hope you understand it.

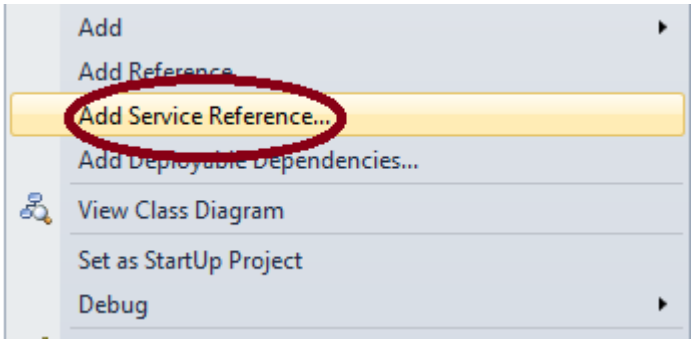
Now your Solution Explorer of ASP.Net Web Application will be as in the following:



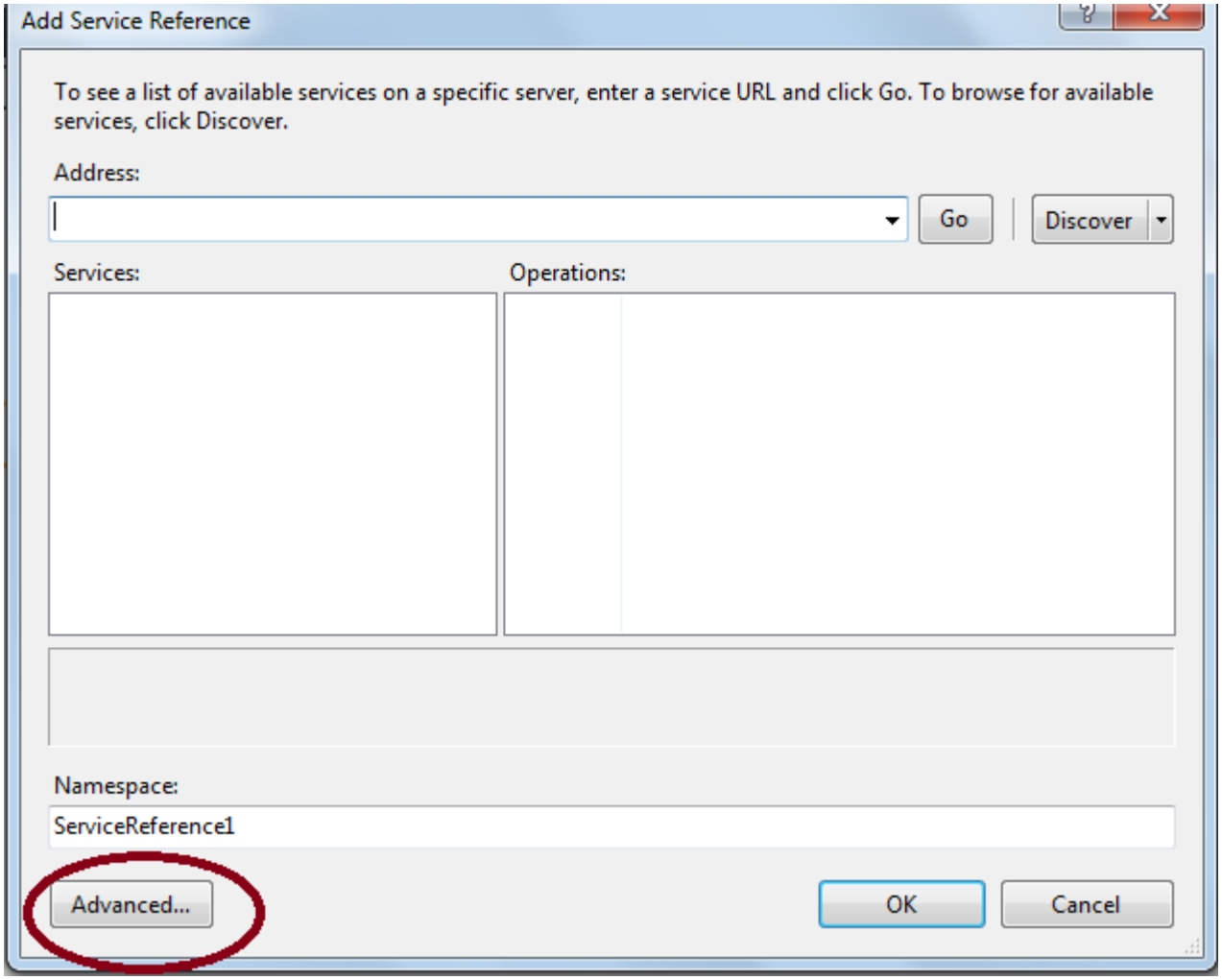
Adding a Web Service Reference in the ASP.Net Web Application

The most important task when consuming a Web Service in an ASP.Net Web Application is adding the Web Service reference into the ASP.Net web application. So how to add it? Let us see the procedure.

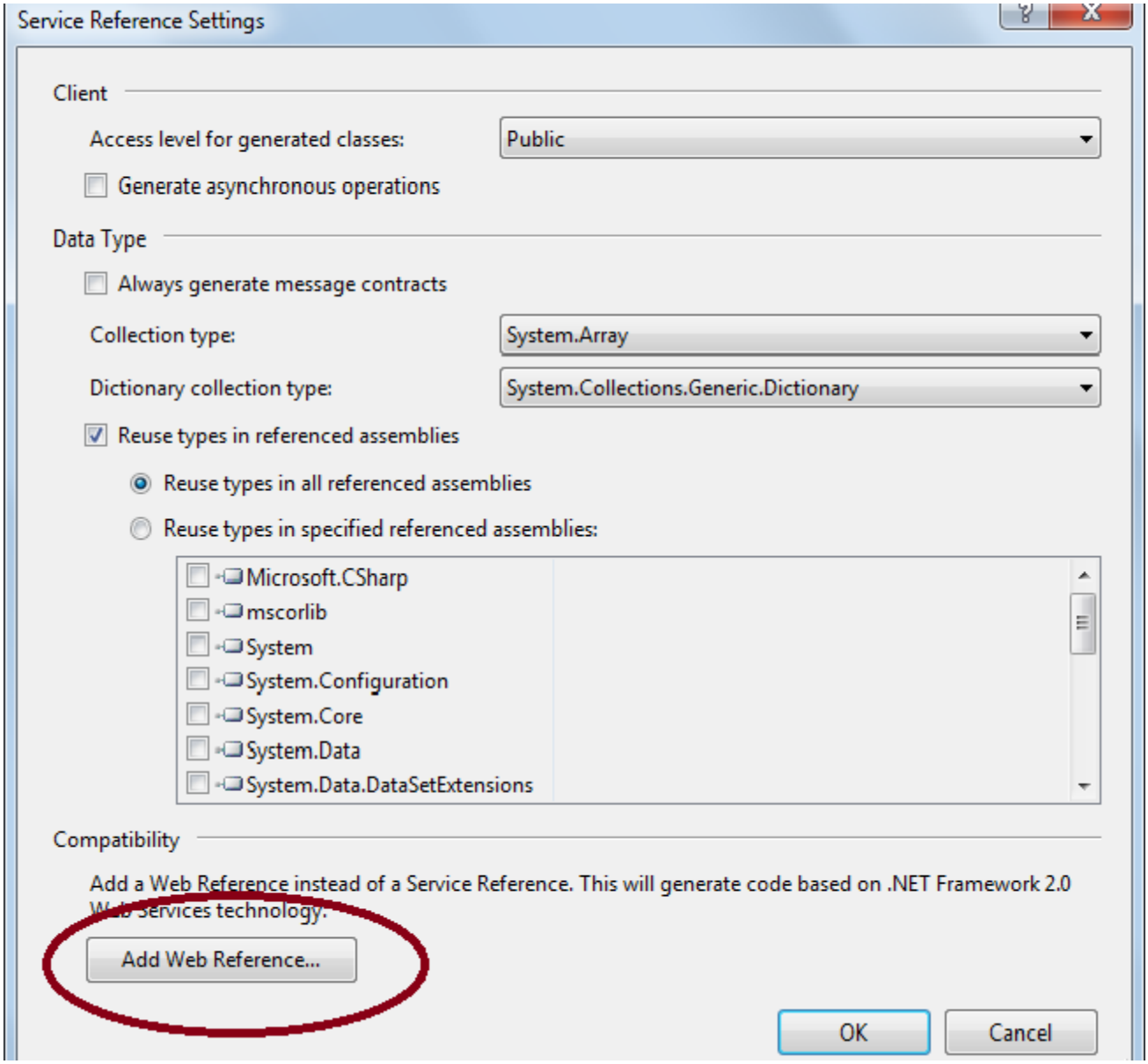
Right-click on the ASP.Net Web Application and click on "Add Service Reference" as in the following:



Then after clicking on the above option, the following window will appear, then click on the "Advanced" tab.

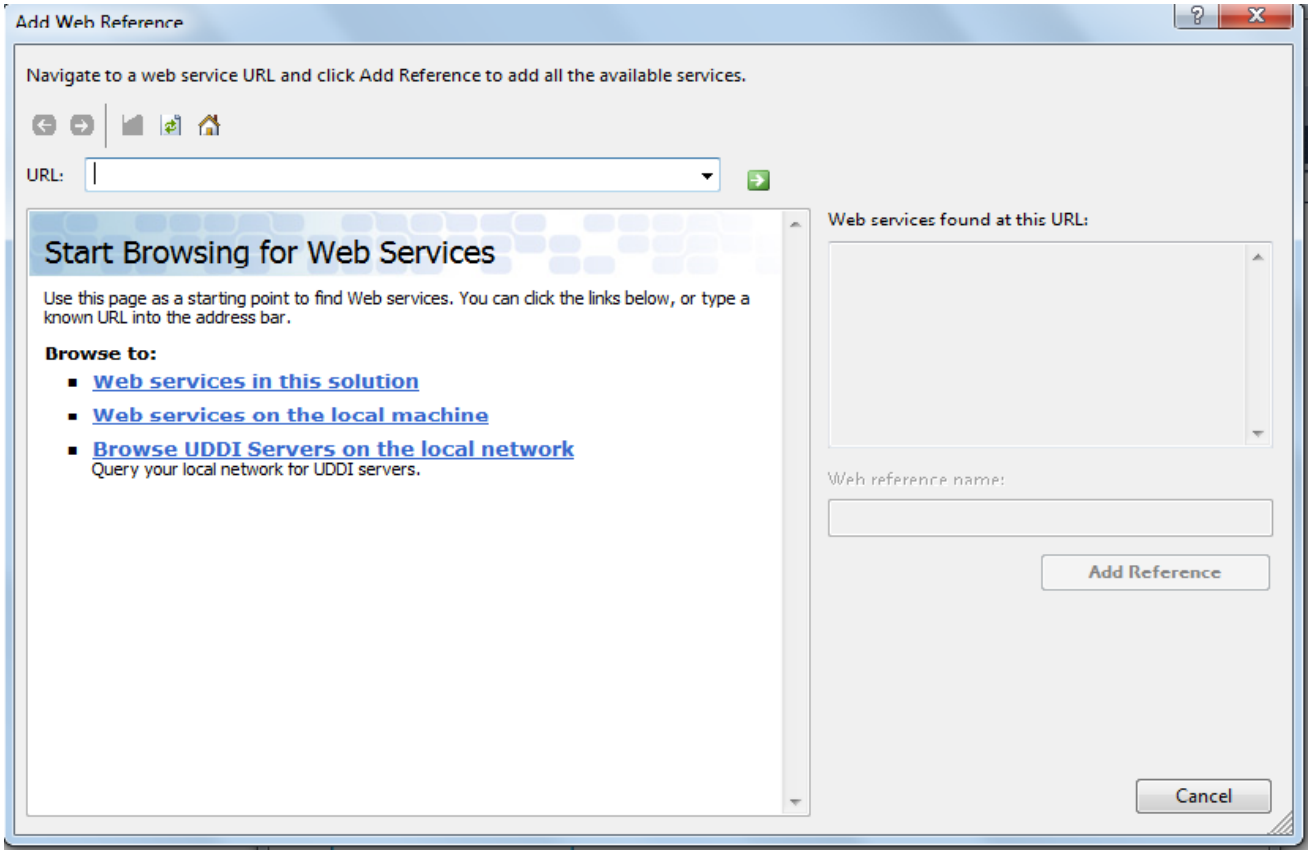


Now after clicking on the Advanced tab, it will show the following window then click on the "Add Web Reference" option as in the following in a circle:



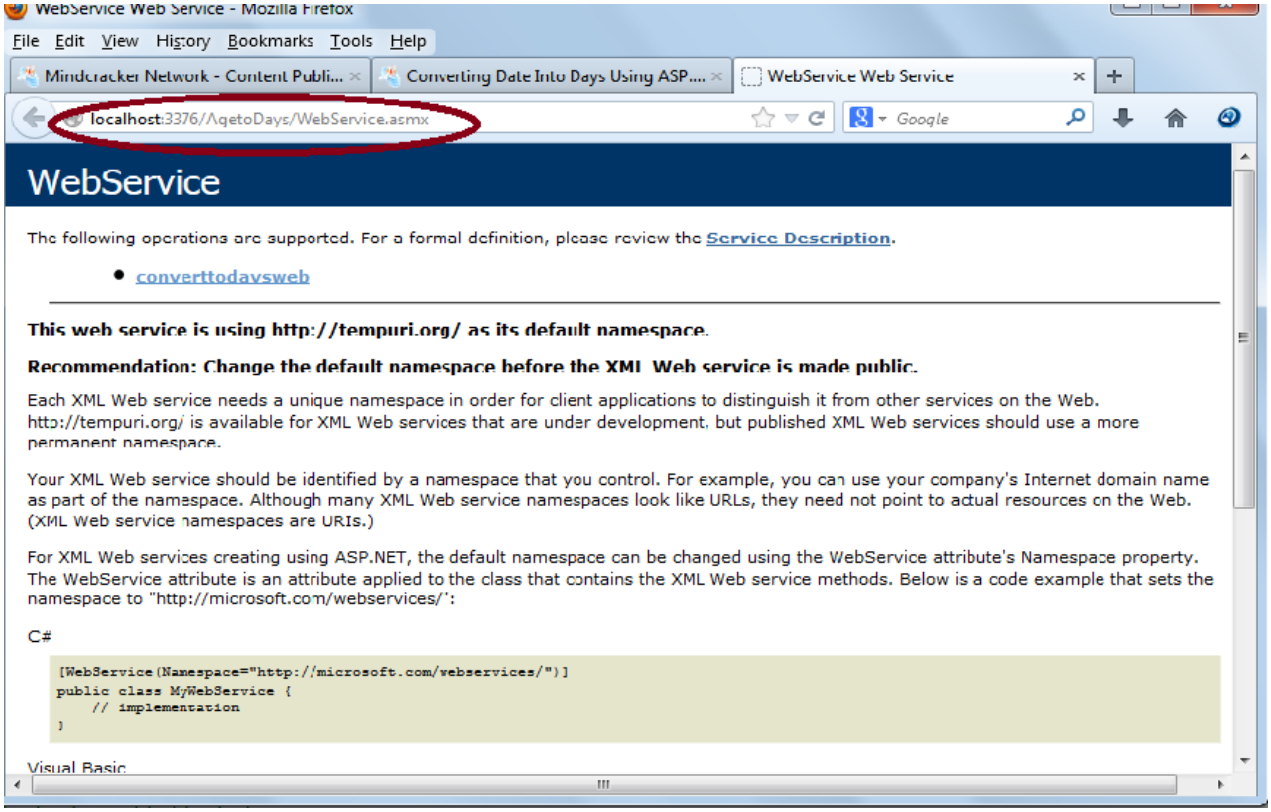
.

After clicking on the Add Web Reference tab, it will show the following window. Now this is a very important step, when adding the web reference to the ASP.Net web Application. Since you see "URL" option in the following window, on that window we need to paste or type the Web Service URL address.

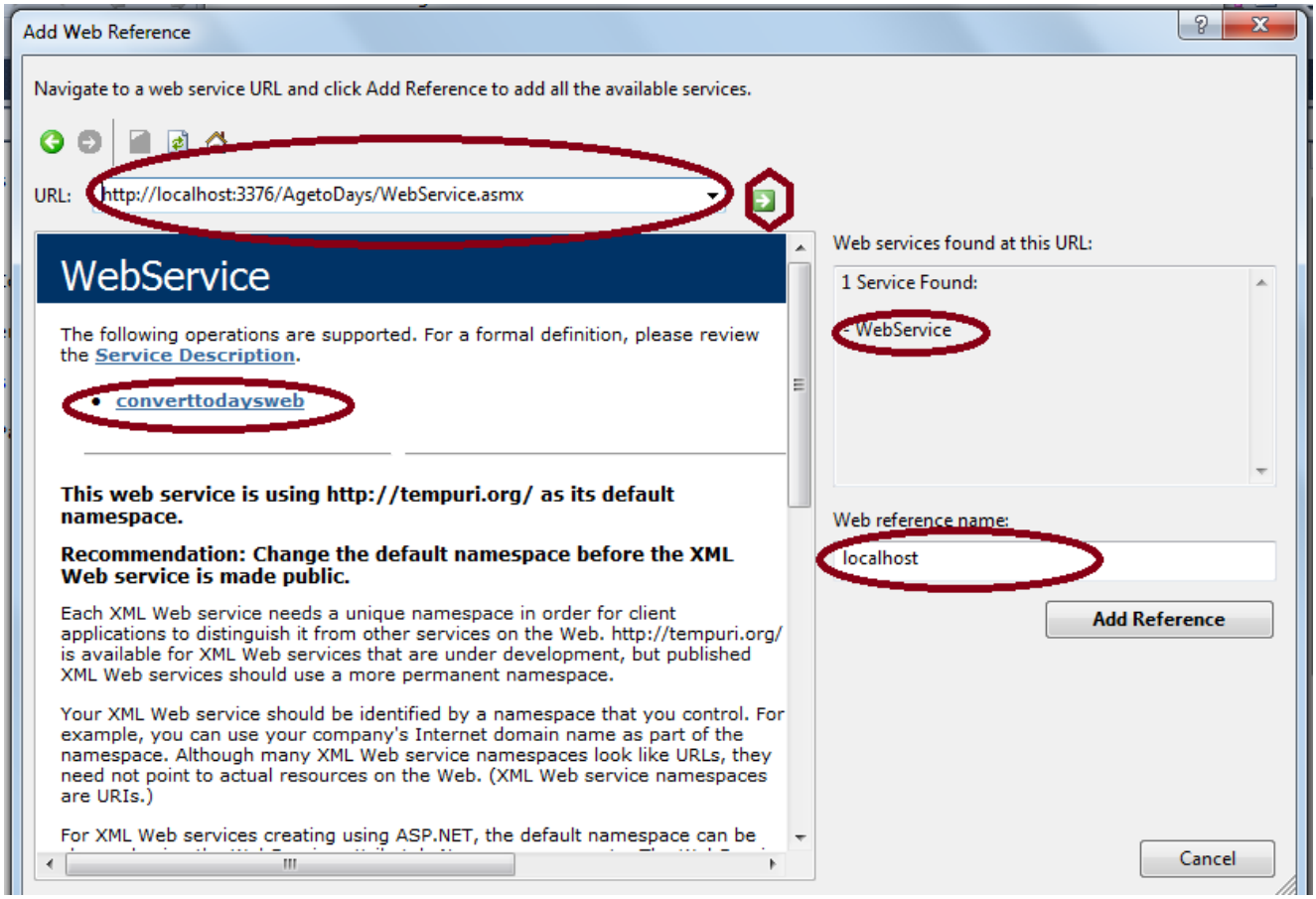


So how to add the URL Reference in the preceding URL box, let us see the procedure again.

- Run the Web Service we created in my article Introduction to Web Service with Example in ASP.Net by clicking on F5 or whatever other option you are familiar with, it will then show the following web page.



As you clearly see there, in the preceding window, it displays the method named "converttodaysweb" as we created in our Web Service, now just you need to copy the preceding URL that I have circled in red and paste it into the Step 4 window URL option, then the Step 4 window will look as in the following:



-----What After Pasting the URL in the preceding URL box

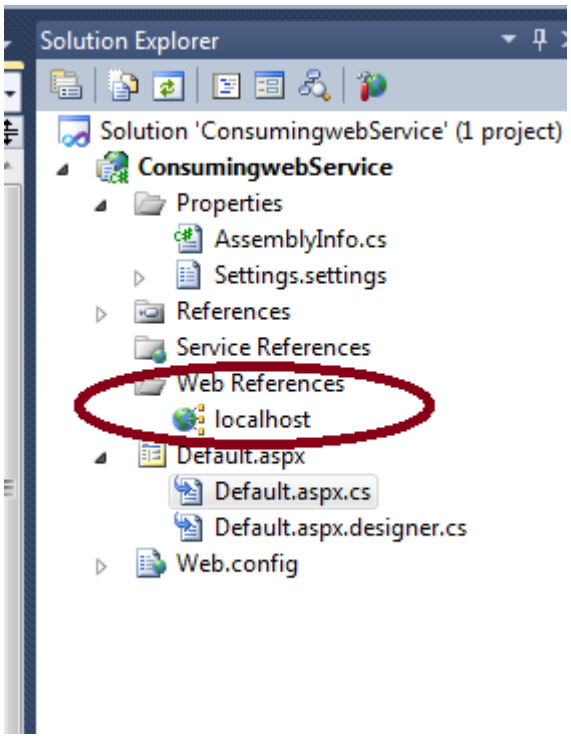
After pasting the URL in the preceding window box, click on the green right headed arrow button, it will discover the Web Services available related to that URL address and you see that in that related URL one Web Service is found message is displayed along with the Web Service name, "Web Services" in the preceding right hand side window.

The Name of the Web Service is "WebService" because I have given the class name as Web Service, that's why the name is Web Services, in your case it might be different or the class name is anything so you can use any name for Web Service so don't be confused about it.

- **Web Reference Name**

In the right hand corner of the window you have seen the option for the Web reference name; the web reference name is anything you wish and this name will be added in your ASP.Net Web Application as allies name for Web Service. In my article I have given the web reference name as "local host".

Then after adding the Web Service reference in the ASP.Net web application the Solution Explorer will look as in the following:



In the preceding window, you have clearly seen that the Web Service reference named "localhost" is added into the ASP.Net web Application. I hope you understand how to add the Web Service reference into the ASP.Net web application.

Calling the Web Service method from the ASP.Net Web Application

We have added the Web Service reference into our web application. Now next is how to call the Web Service method that we created in our Web Service Application from the ASP.Net Web Application.

The following is the procedure:

1. Go to the Default.aspx page of our ASP.Net Web application and double-click on the button that we have placed on the Default.aspx page.

2. Now write the following code in the button click to create the object of the Web Service class:

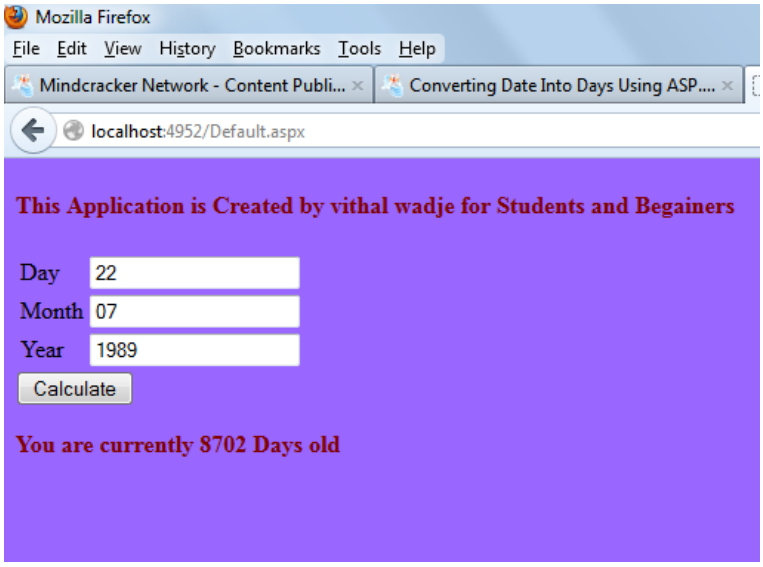
localhost.webservice age=new localhost.webservice();

In the code above, I have created the object of Web Service class named "age" followed by the Web reference name ("localhost") and Web Service class ("webservice"), I hope you understand how to create the object of the Web Service class.

The entire code of the Default.aspx.cs page will be as follows:

```
protected void Button1_Click(object sender, EventArgs e)
{

    localhost.WebService age = new localhost.WebService();

    int day =int.Parse(TextBox1.Text.ToString());
    int month = int.Parse(TextBox2.Text.ToString());
    int year = int.Parse(TextBox3.Text.ToString());

 int a=  age.converttodaysweb(day, month, year);


 Label1.Text ="You are currently"+""+ a.ToString()+" "+"Days old";


}
```

Output



Time Boxing

Activity Name	Activity Time	Total Time
Login Systems + Setting up Visual studio Environment	3 mints + 5 mints	8 mints
Walk through Theory & Tasks	60 mints	60 mints
Implement Tasks	80 mints	80 mints
Evaluation Time	30 mints	30 mints
	Total Duration	178 mints

Objectives

After completing this lab the student should be able to:

- Clearly understand the purpose and benefits that Cloud Computing has to offer.
- Understand the concept of Cloud Computing.
- Build a basic Cloud Computing base application.
- Use and learn Cloud Computing need.

Lab Tasks/Practical Work

1. Consume web Services in application for calculating DOB

Lab No. 8

Implementing Work Flows

LAB # 08

Implementing Work Flows

What is Workflow?

A workflow is a series of activities involved in a task or set of activities collectively forms a work. Except for the first and last step, steps will have a preceding and succeeding step.

Windows Workflow Foundation provides a very nice graphical interface to create the activities involved in a workflow. Windows Workflow Foundation is a great technology for creating workflows and can be used in combination with different technologies, like SharePoint, WCF, ASP.NET etc. Windows workflow foundation comes with a set of commonly used activities packaged along with it. They will be available in the Toolbox and you can just drag and drop them to your workflow. You can create your own custom activities and make them available in the Toolbox as well.

A workflow model when compiled, it can be executed inside any Windows process including console applications, forms-based applications, Windows Services, ASP.NET Web sites or Web services. Since a workflow is hosted in a process, a workflow can easily communicate with its host application. Though we use graphical interface to desgin workflows, they are storedsas XAML markup.

Windows Workflow Foundation can be used in following types of scenarios but not limited to this:

- Applications with User-interface page flows.
- Workflows that are document-centric.
- Human workflows.
- Service-oriented applications wtih composite workflows.
- Rule-driven workflows for business.
- Workflows for systems management.

Workflow framework can be used or are useful in the following situations:

- For creating business rule engines. This may include simple rules, such as if-then-else decisions based on an input argument to more complex rules, such as the potentially large set that is a combination of several other rules.
- Case where maintaining state for the workflow's lifetime is needed. For example, if the employee who is in charge of approving a task is on leave for a week, the task should be

bookmarked or stored for use at a later point of time and should be able to continue the workflow. The bookmarked or stored task should be able to re-activate when required.

- Interactive applications that connect with people. For example, if an HR must approve some policy documents, the workflow must be able to display a user interface or communicate to the HR through other software.
- Changing workflow behavior like changing a condition or editing an action which are common use cases.
- Multiple applications that work together. For example, a workflow rule engine gets input from an ASP.NET web application and does workflow activities and call another application for generating a travel request.
- Monitoring of a running workflow and examining its execution in real time.
- Support for graphical user interface tools for creating workflows easily. A workflow consists of different steps or activities and we should be able to re-use the activities if we create one.

Time Boxing

Activity Name	Activity Time	Total Time
Login Systems + Setting up Visual studio Environment	3 mints + 5 mints	8 mints
Walk through Theory & Tasks	60 mints	60 mints
Implement Tasks	80 mints	80 mints
Evaluation Time	30 mints	30 mints
	Total Duration	178 mints

Objectives

After completing this lab the student should be able to:

- Clearly understand the purpose and benefits that Cloud Computing has to offer.
- Understand the concept of Cloud Computing.
- Build a basic Cloud Computing base application.
- Use and learn Cloud Computing need.

Lab Tasks/Practical Work

1. Create a workflow for simple calculator
2. Create workflow for any real world problem and test asp.net mvc application.

Lab No. 9

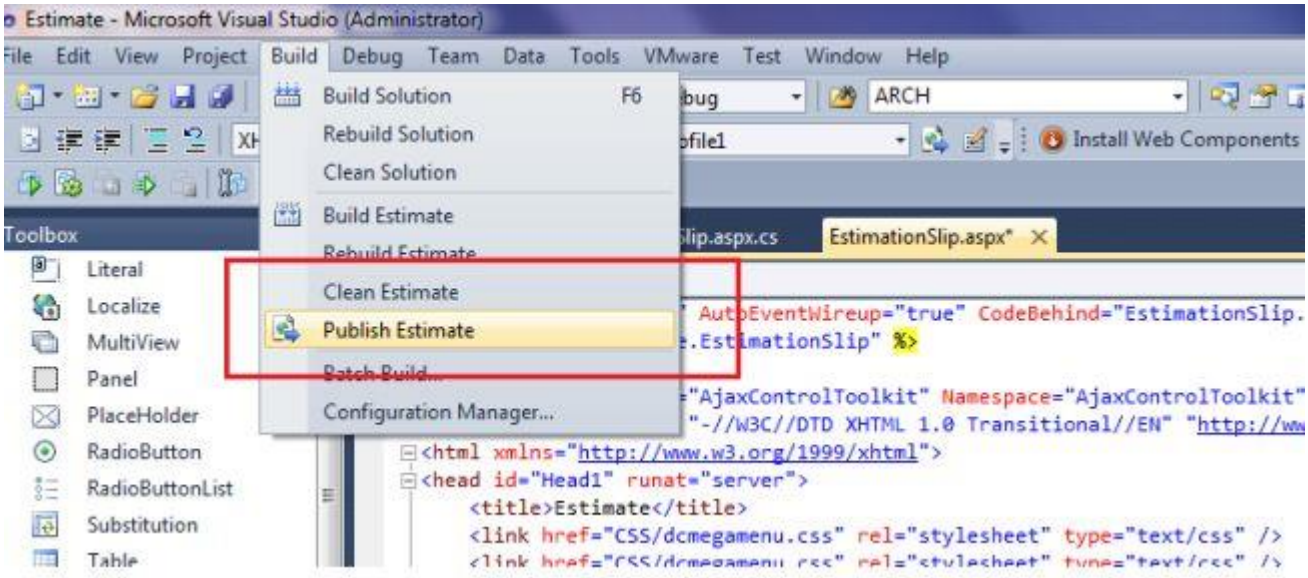
Deployment of Web Application using IIS

LAB # 09

Deployment of Web Application using IIS

First open your ASP.Net web application in Visual Studio.

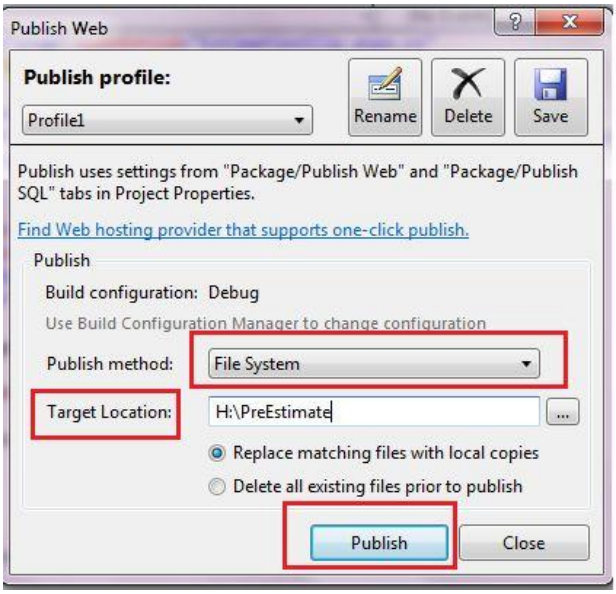
Now in the top we have the option **Build**. Click on that and under Build you will find Publish Website.



Click on Publish Website. Now open the publish web pop-up.

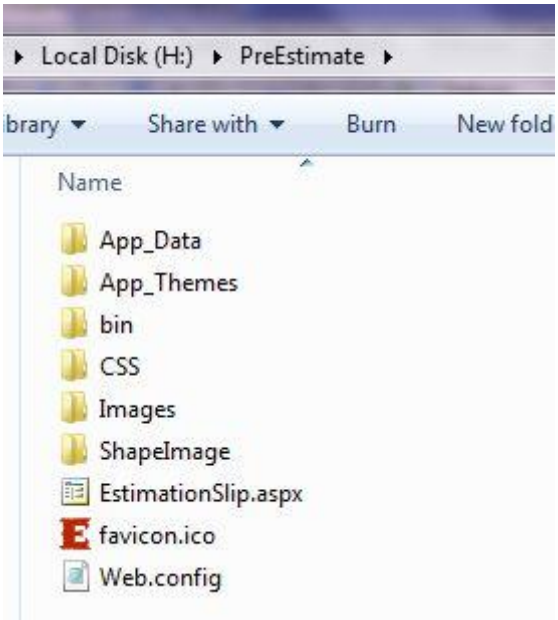
For Publish method select File System.

For Target location specify where to save your web application DLL file.



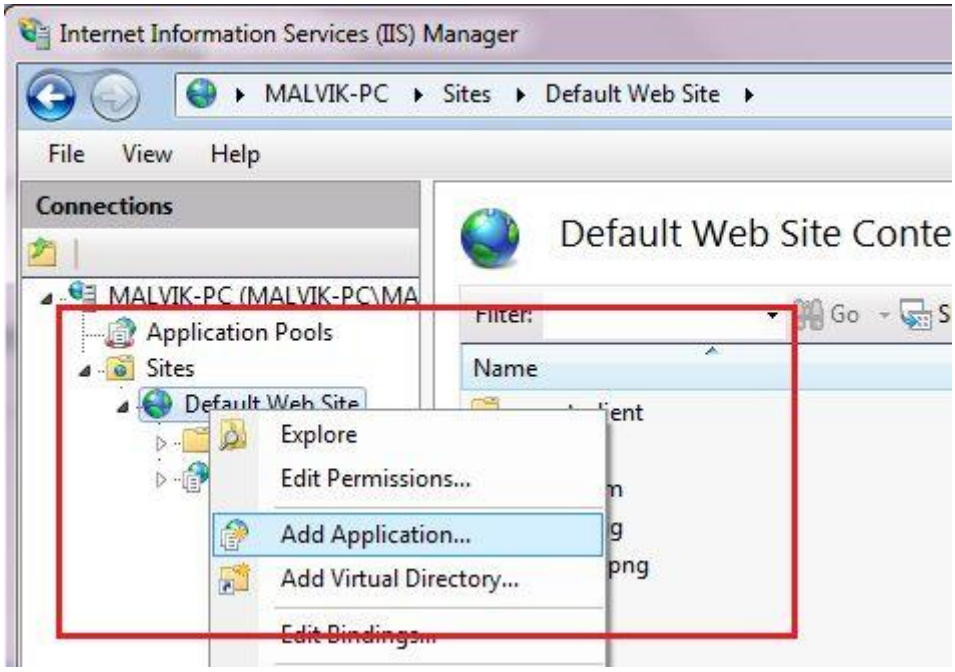
Then click on publish.

Go to the target folder and in that location, you will see your web application DLL file.

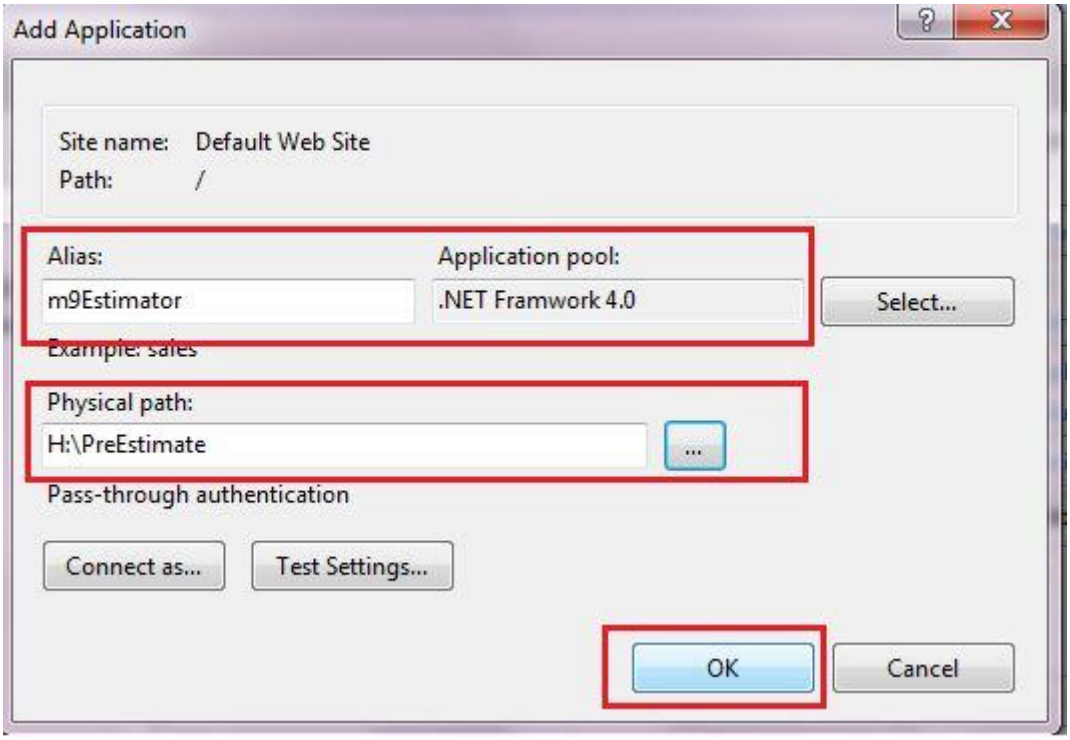


Now open IIS manager. (Type **inetmgr** in the Run command.)

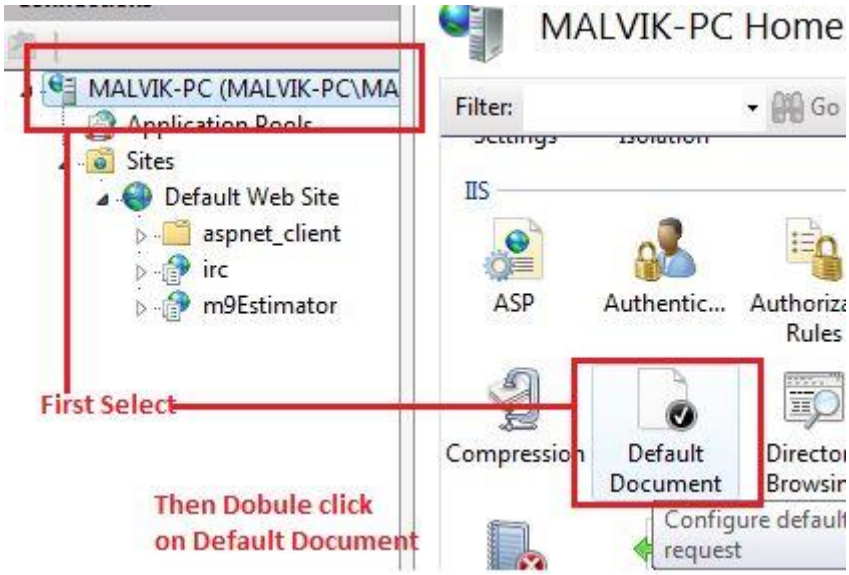
Right-click on Default Application and Add Application.



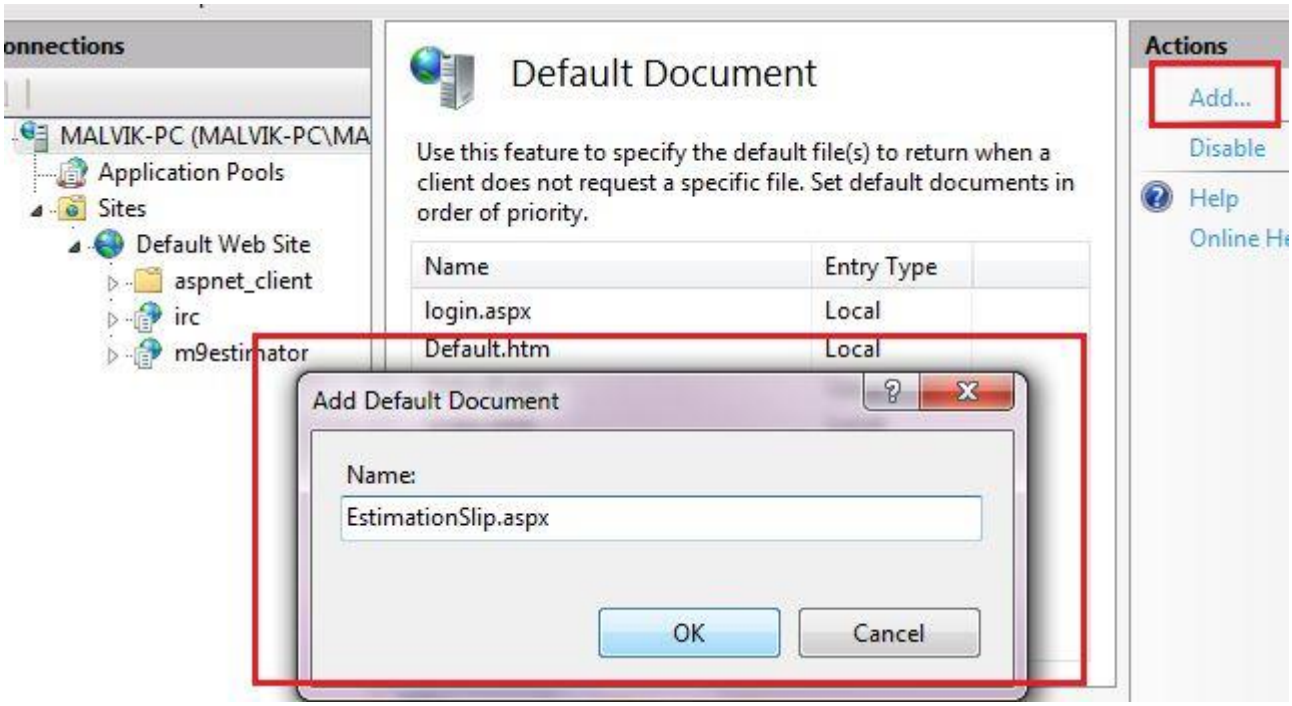
Enter Alias name then select an Application pool and Physical path.



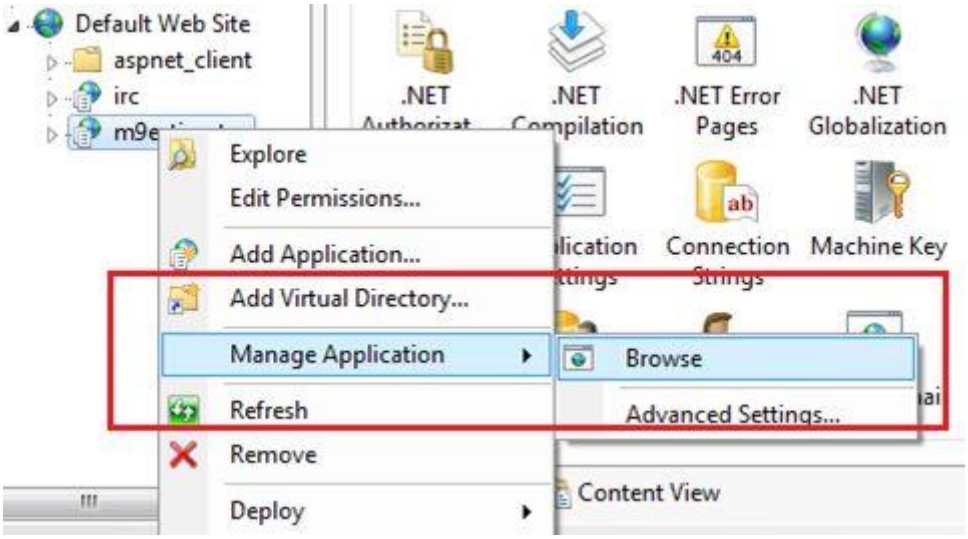
Now Double-click on default document.



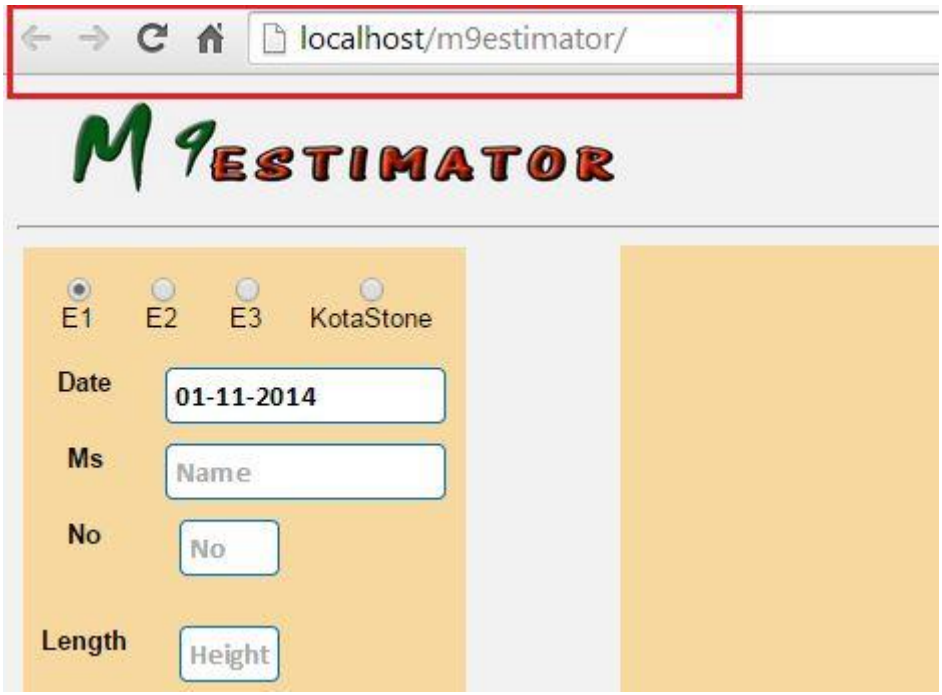
And add a start page of your web application. Here my application start page is EstimationSlip.aspx.



Now right-click on your application and browse.



See your application without Visual Studio.



Time Boxing

Activity Name	Activity Time	Total Time
Login Systems + Setting up Visual studio Environment	3 mints + 5 mints	8 mints
Walk through Theory & Tasks	60 mints	60 mints
Implement Tasks	80 mints	80 mints
Evaluation Time	30 mints	30 mints
	Total Duration	178 mints

Objectives

After completing this lab the student should be able to:

- Clearly understand the purpose and benefits that Cloud Computing has to offer.
- Understand the concept of Cloud Computing.
- Build a basic Cloud Computing base application.
- Use and learn Cloud Computing need.

Lab Tasks/Practical Work

1. Create a simple website and deploy it on IIS.
2. Consider a restaurant who wants to deploy their online billing system on local servers. Deploy that website using IIS

Lab No. 10

Two factor validation with Email and Phone using asp.net MVC

LAB # 10

Two factor validation with Email and Phone using asp.net MVC

Multi-factor authentication (MFA) is a process in which a user is requested during a sign-in event for additional forms of identification. This prompt could be to enter a code from a cellphone, use a FIDO2 key, or to provide a fingerprint scan. When you require a second form of authentication, security is enhanced. The additional factor isn't easily obtained or duplicated by an attacker.

MFA, 2FA

MFA requires at least two or more types of proof for an identity like something you know, something you possess, or biometric validation for the user to authenticate.

Two-factor authentication (2FA) is like a subset of MFA, but the difference being that MFA can require two or more factors to prove the identity.

MFA TOTP (Time-based One-time Password Algorithm)

MFA using TOTP is a supported implementation using ASP.NET Core Identity. This can be used together with any compliant authenticator app, including:

- Microsoft Authenticator App
- Google Authenticator App
- See the following link for implementation details:
- Enable QR Code generation for TOTP authenticator apps in ASP.NET Core

MFA FIDO2 or passwordless

FIDO2 is currently:

- The most secure way of achieving MFA.
- The only MFA flow that protects against phishing attacks.

At present, ASP.NET Core doesn't support FIDO2 directly. FIDO2 can be used for MFA or passwordless flows.

Azure Active Directory provides support for FIDO2 and passwordless flows. For more information, see Passwordless authentication options for Azure Active Directory.

Time Boxing

Activity Name	Activity Time	Total Time
Login Systems + Setting up Visual studio Environment	3 mints + 5 mints	8 mints
Walk through Theory & Tasks	60 mints	60 mints
Implement Tasks	80 mints	80 mints
Evaluation Time	30 mints	30 mints
	Total Duration	178 mints

Objectives

After completing this lab the student should be able to:

- Clearly understand the purpose and benefits that Cloud Computing has to offer.
- Understand the concept of Cloud Computing.
- Build a basic Cloud Computing base application.
- Use and learn Cloud Computing need..

Lab Tasks/Practical Work

1. Create a MVC website in which we have two authentication of email and phone no.

Lab No. 11

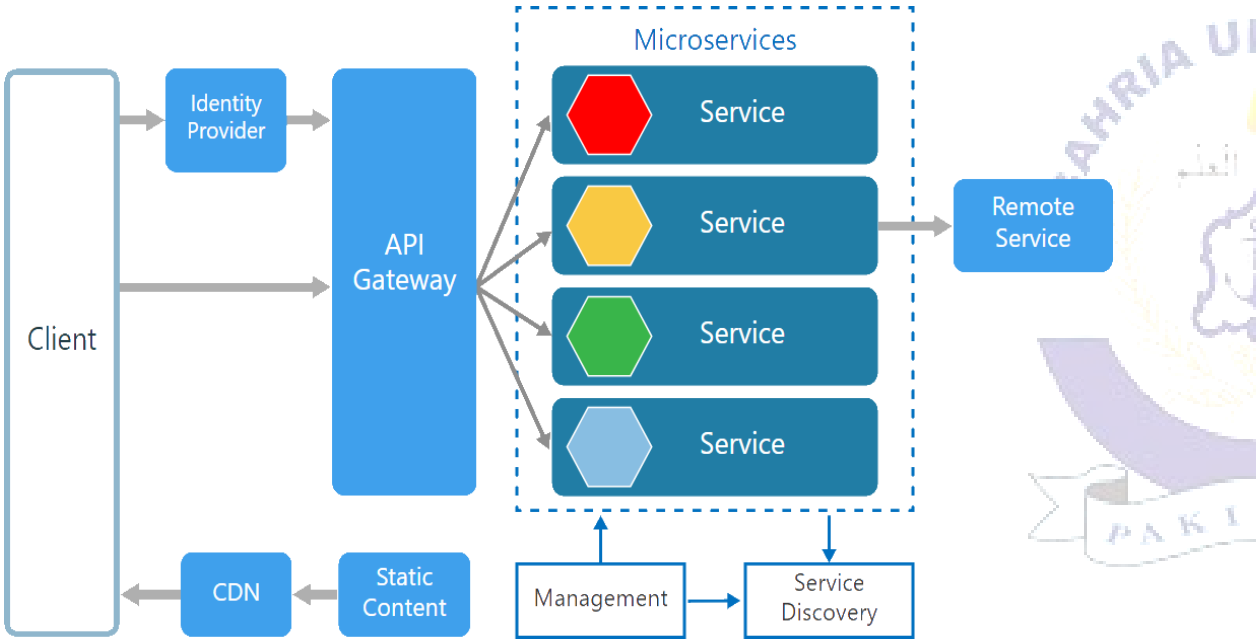
Introduction to Micro services

LAB # 11

Introduction to Micro services

The term micro services portrays a software development style that has grown from contemporary trends to set up practices those are meant to increase the speed and efficiency of developing and managing software solutions at scale. Micro services is more about applying a certain number of principles and architectural patterns an architecture. Each micro service lives independently, but on the other hand, also all rely on each other. All micro services in a project get deployed in production at their own pace, on- premise on the cloud, independently, living side by side.

Micro services Architecture

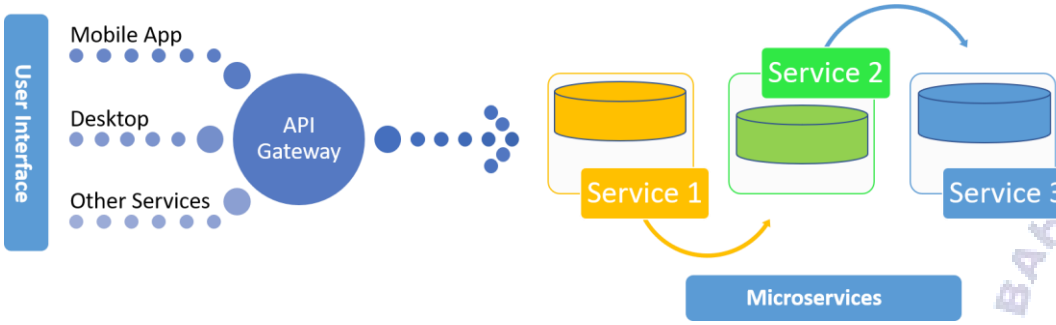


There are various components in a microservices architecture apart from microservices themselves.

- **Management**. Maintains the nodes for the service.
- **Identity Provider**. Manages the identity information and provides authentication services within a distributed network.
- **Service Discovery**. Keeps track of services and service addresses and endpoints.

- **API Gateway**. Serves as client's entry point. Single point of contact from the client which in turn returns responses from underlying microservices and sometimes an aggregated response from multiple underlying microservice.
- **CDN**. A content delivery network to serve static resources for e.g. pages and web content in a distributed network
- **Static Content** The static resources like pages and web content

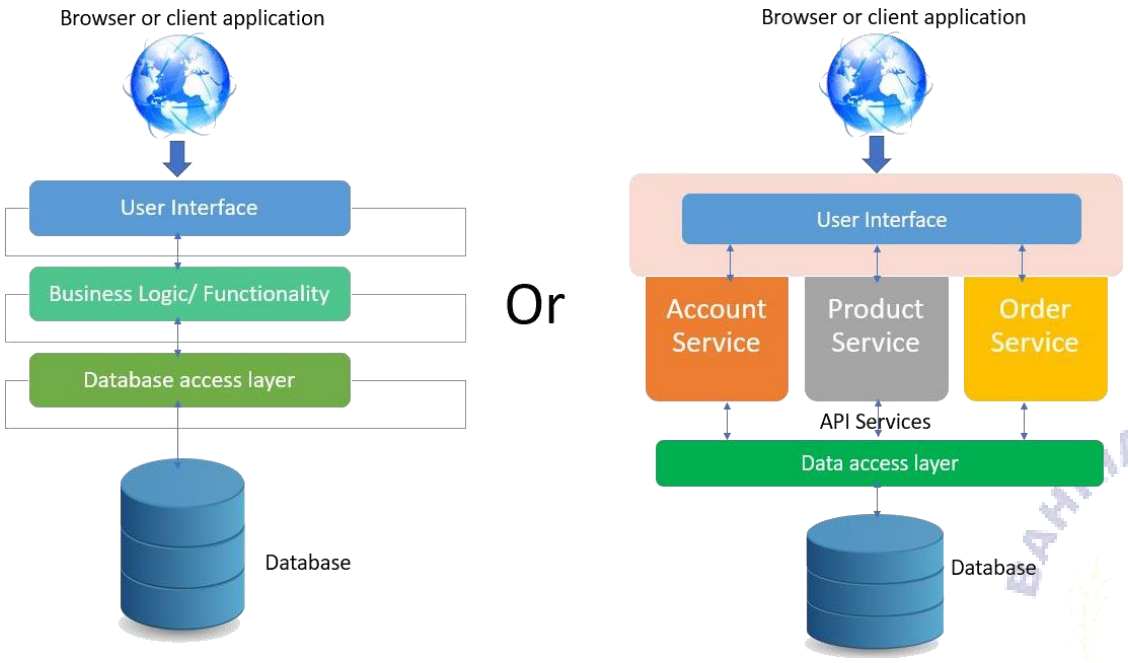
Micro services are deployed independently with their own database per service so the underlying micro services look as shown in the following picture.



Monolithic vs. Micro services Architecture

Monolithic applications are more of a single complete package having all the related needed components and services encapsulated in one package.

Following is the diagrammatic representation of monolithic architecture being package completely or being service based.



Dockers Containers and Dockers installation

Containers like Dockers and others slice the operating system resources for e.g. the network stack, processes namespace, file system hierarchy and the storage stack. Dockers are more like virtualizing the operating system.

Micro service using ASP.NET Core

This section will demonstrate how to create a Product micro service using ASP.NET Core step by step with the help of pictures. The service would be built using ASP.NET Core 2.1 and Visual Studio 2017. Asp.NET Core comes integrated with VS 2017. This service will have its own dB context and database with the isolated repository so that the service could be deployed independently

Time Boxing

Activity Name	Activity Time	Total Time
Login Systems + Setting up Visual studio Environment	3 mints + 5 mints	8 mints
Walk through Theory & Tasks	60 mints	60 mints

Implement Tasks	80 mints	80 mints
Evaluation Time	30 mints	30 mints
	Total Duration	178 mints

Objectives

After completing this lab the student should be able to:

- Clearly understand the purpose and benefits that Cloud Computing has to offer.
- Understand the concept of Cloud Computing.
- Build a basic Cloud Computing base application.
- Use and learn Cloud Computing need.

Lab Tasks/Practical Work

1. Create micro services for any website using asp.ne core.

.

Lab No. 12

Consuming Micro services in ASP.NET Core

LAB # 12

Consuming Micro services in ASP.NET Core

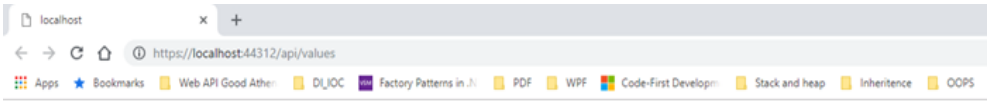
Run the Product Micro service

The service could be run via IIS Express i.e. Visual Studio default or via Docker container as well.

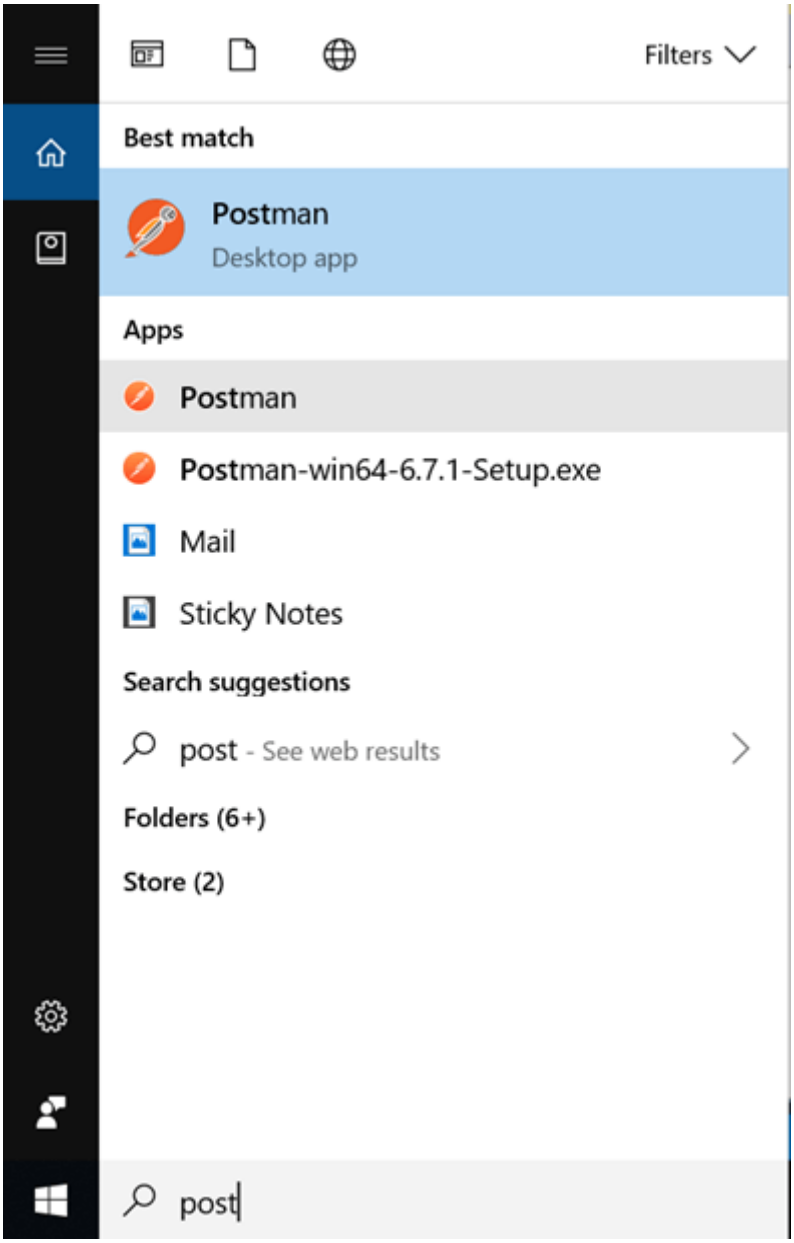
Via IIS Express

Choose IIS Express in the Visual Studio as shown below and press F5 or click that IIS Express button itself.

The application will be up once the browser page is launched. Since it has nothing to show, it will be blank, but the service could be tested via any API testing client. Here Postman is used to testing the service endpoints. Keep it opened and application running.

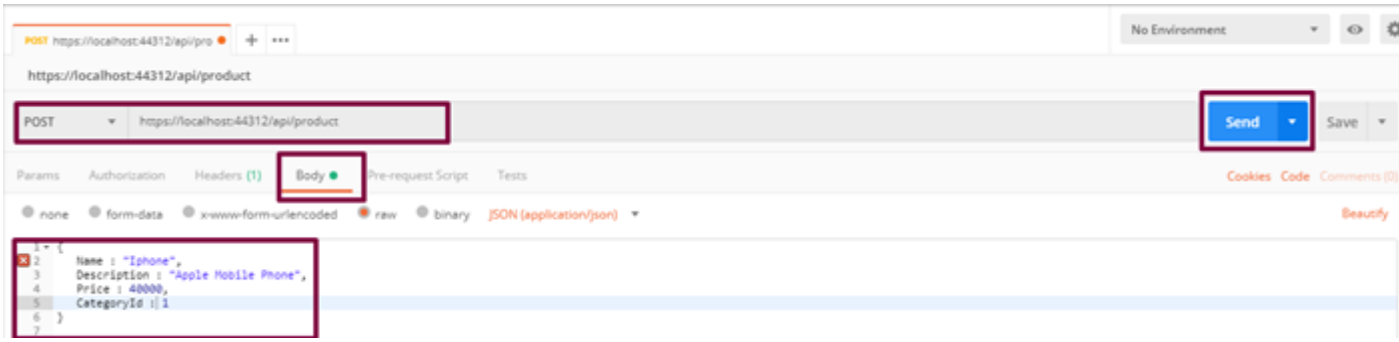


Install Postman if it is not on the machine and launch it.

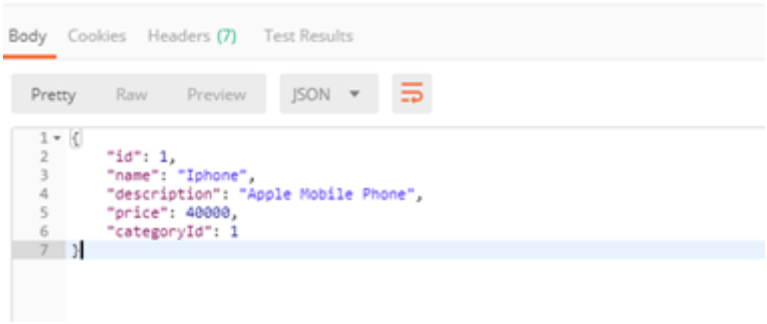


POST

To test the POST method; i.e. create a new resource, select the method as POST in postman and provide the endpoint, i.e. https://localhost:44312/api/product and in the Body section, add a JSON similar to having properties of Product model as shown below and click on Send.

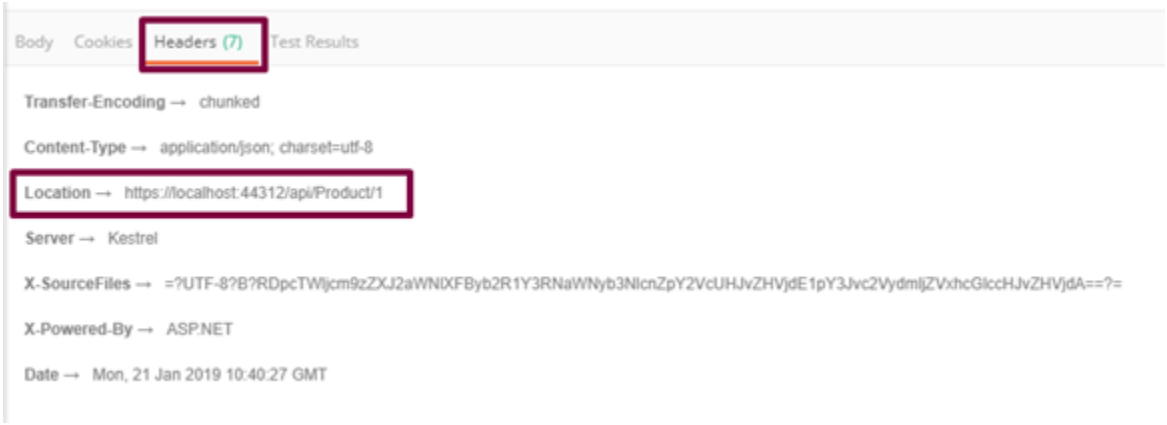


The response is returned with the Id of the product as well.

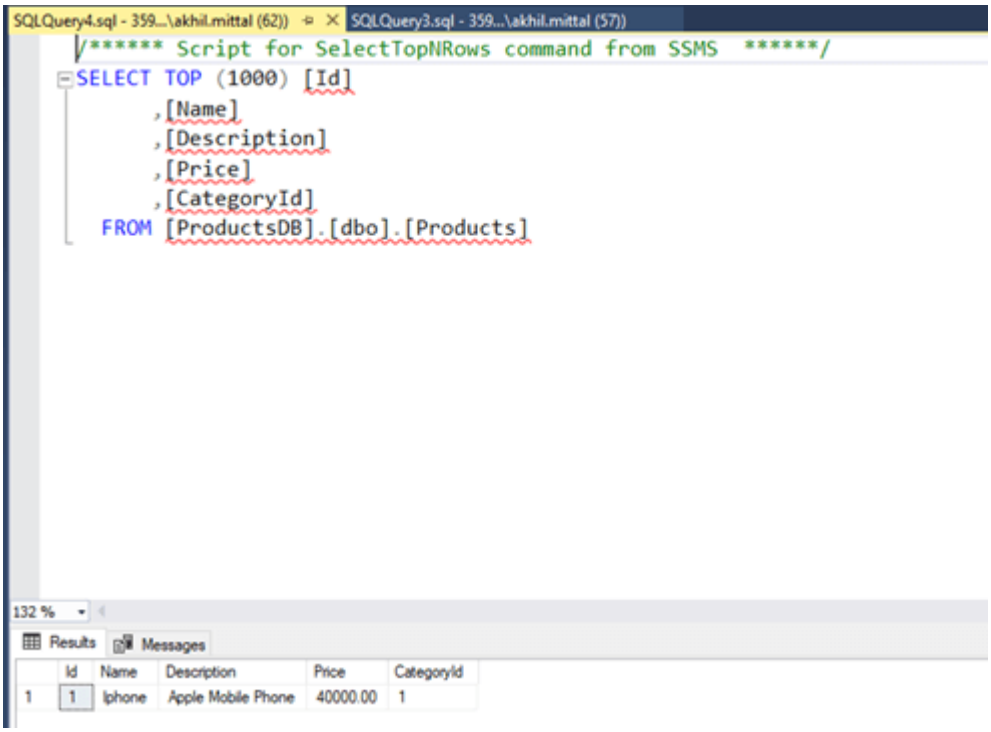


The “Post” method of the controller is responsible to create a resource in the database and send the response.

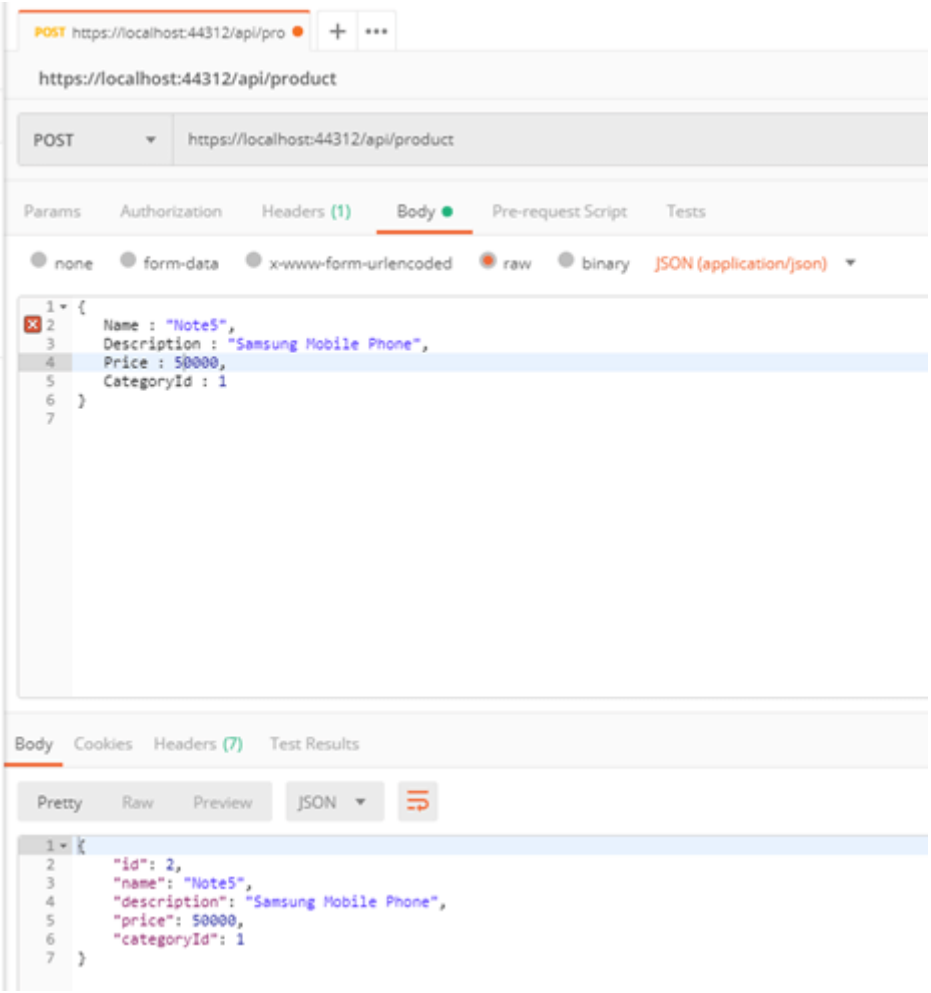
The line return CreatedAtAction(nameof(Get), new { id=product.Id }, product); returns the location of the created resource that could be checked in Location attribute in the response under Headers tab.



Perform a select query on the product table and an added row is shown for the newly created product.

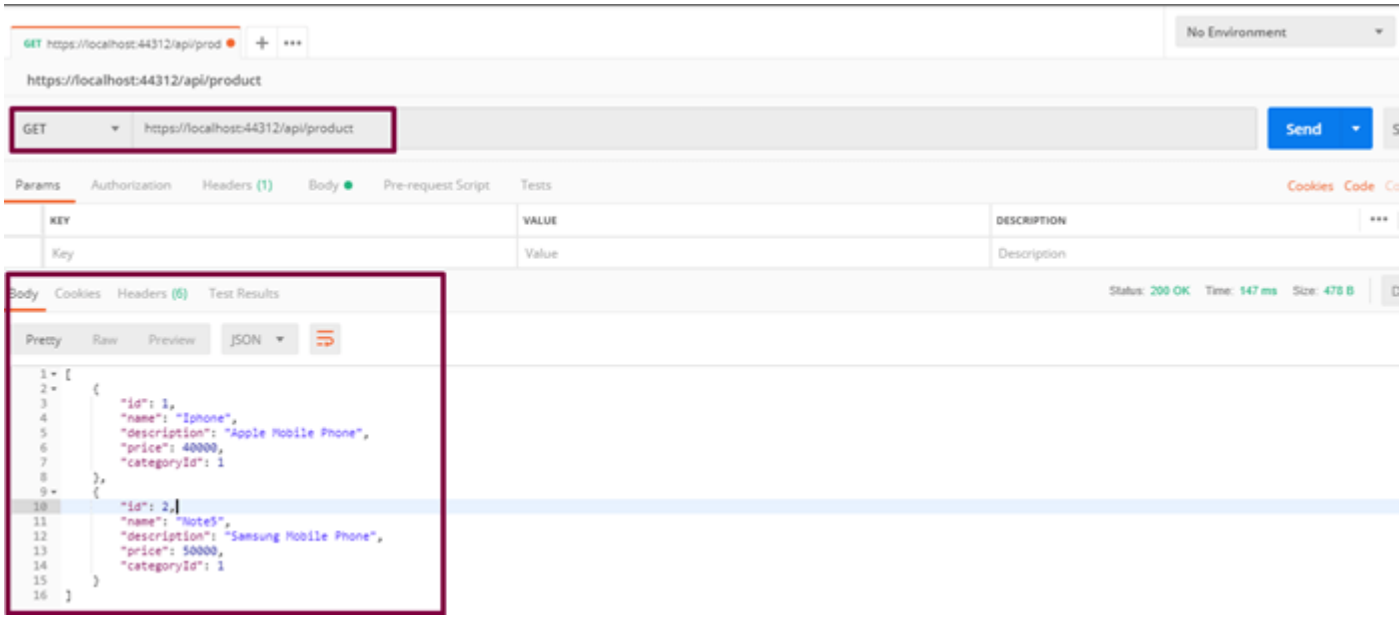


Create one more product in a similar way.



GET

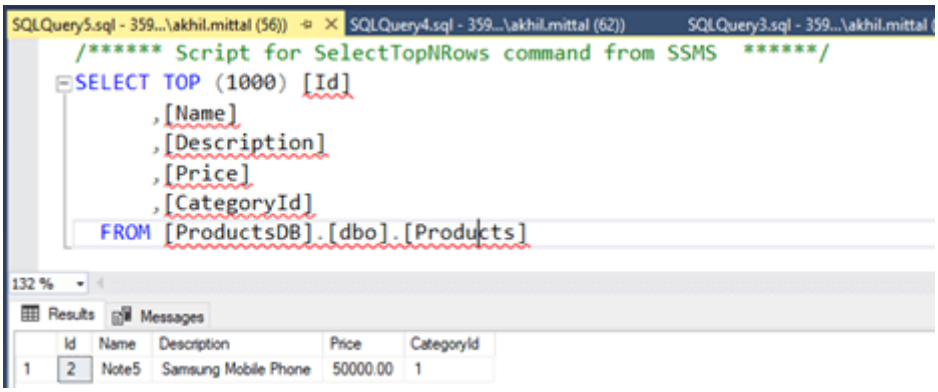
Perform a GET request now with the same address and two records are shown as a JSON result response.



DELETE

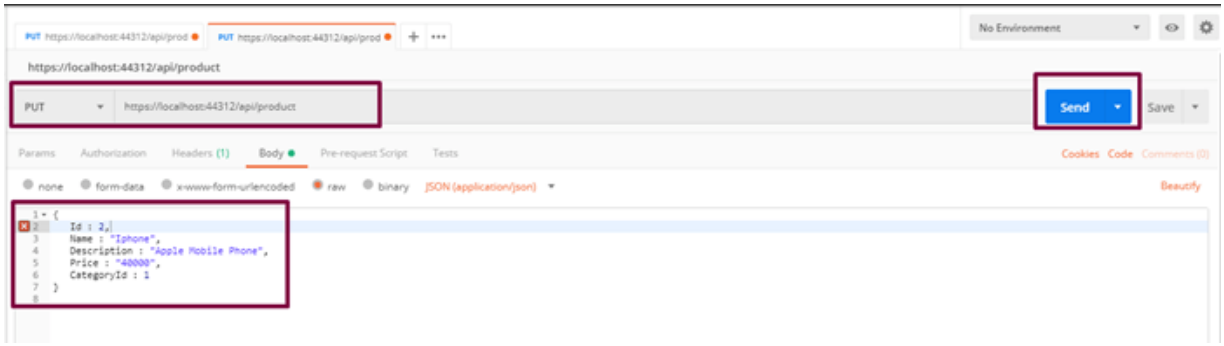
Perform the delete request by selecting DELETE as the verb and appending id as 1 (if the product with id 1 needs to be deleted) and press Send.

In the database, one record with Id 1 gets deleted.

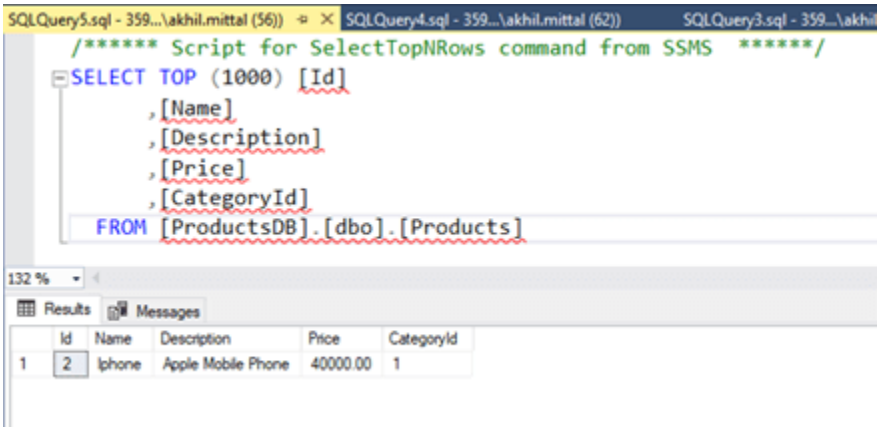


PUT

PUT verb is responsible for updating the resource. Select PUT verb, provide the API address and in the Body section, provide details of which product needs to be updated in JSON format. For example, update the product with Id 2 and update its name, description, and price from Samsung to iPhone specific. Press Send.



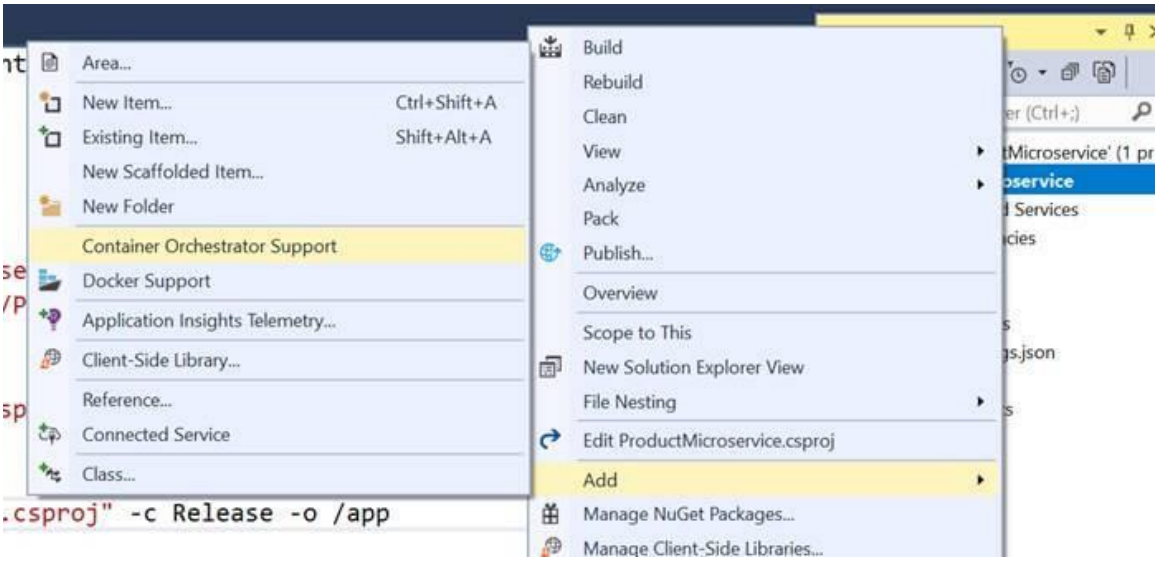
Check the database to see the updated product.



Via Docker Containers

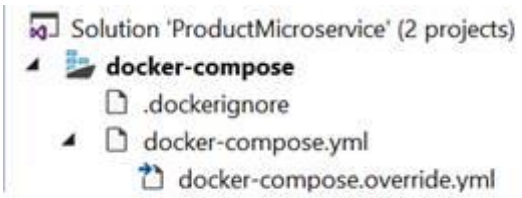
Running the service could be done via docker commands to be run in docker command prompt and using visual studio as well. Since we added the docker support, it is easy to run the service in docker container using visual studio.

1. Add container orchestrator support in the solution as shown below.



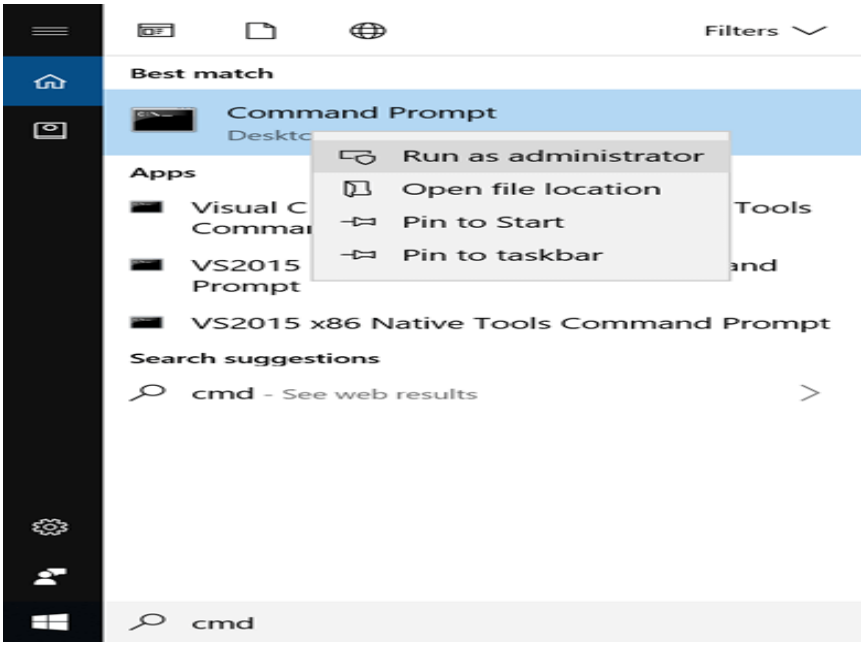
2. This will ask for the orchestrator. Select Docker Compose and press OK.

Once added to the solution, the solution will look like shown below having docker-compose with dockerignore and docker-compose.yml and its override file.

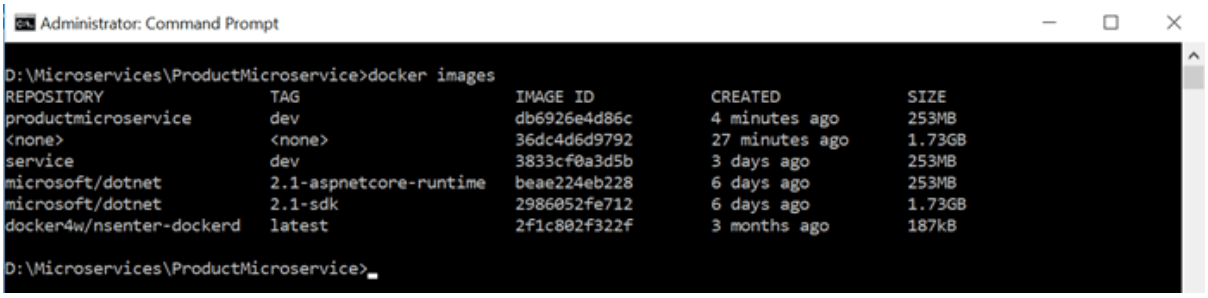


As soon as the solution is saved, it builds the project under the container and creates a docker image. All the commands execution can be seen in the output window when the solution is saved.

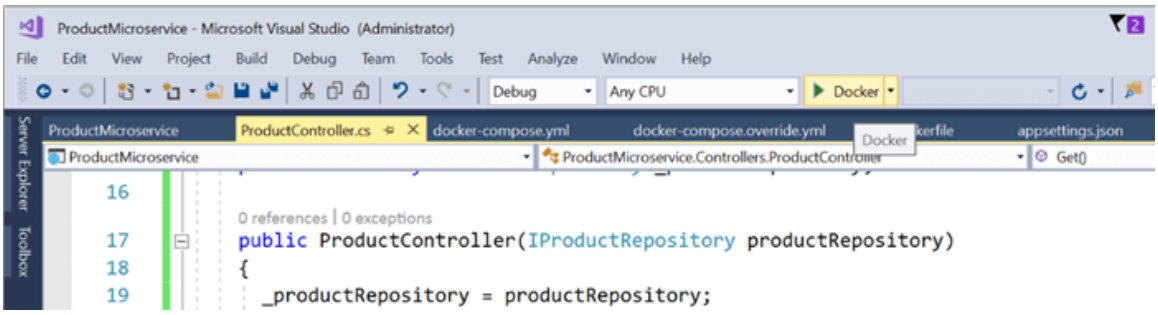
3. Open the command prompt in admin mode and navigate to the same folder where the project files are.



4. Run the command docker images to see all the created images. We see the ProductMicroserviceimage the latest one.

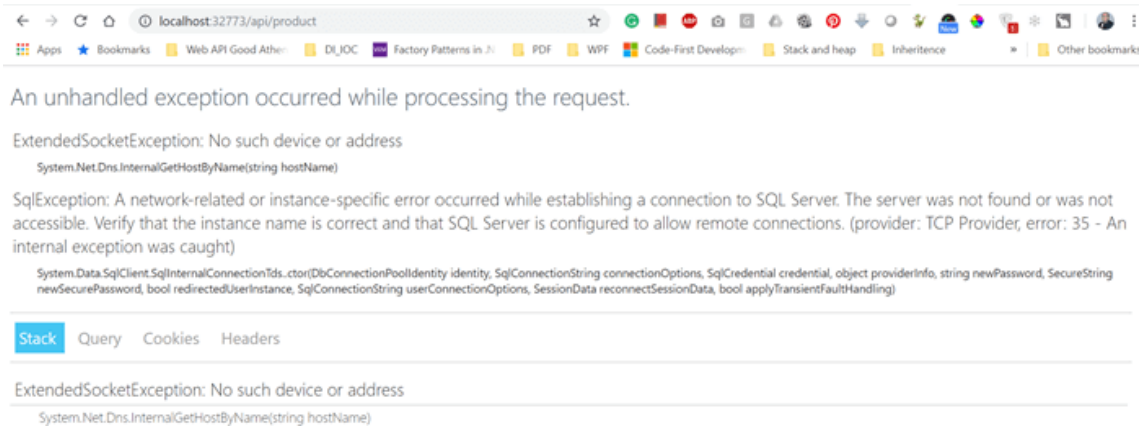


5. Now run the application with Docker as an option as shown below.



6. Now, run the command docker ps to see the running containers. It shows the container is running on 32773:80 port.

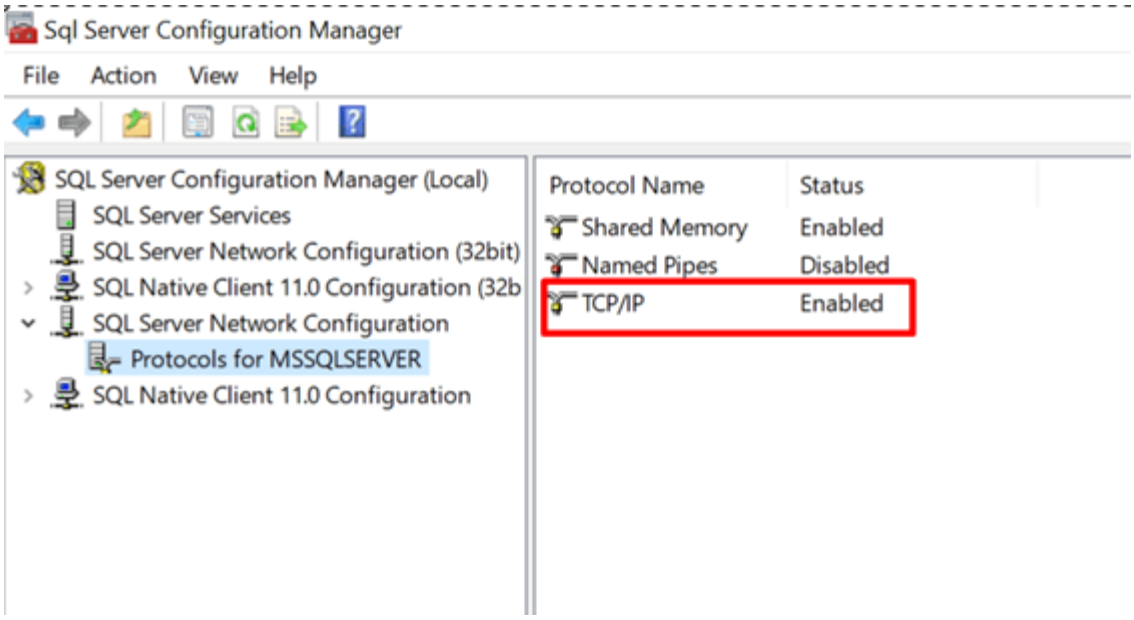
7. Since the container is in running state, it is good to test the service now running under the container. To test the service, replace ”values” with “product” in the address as shown below. Ideally, it should get the product details. But it gives exception as shown below.

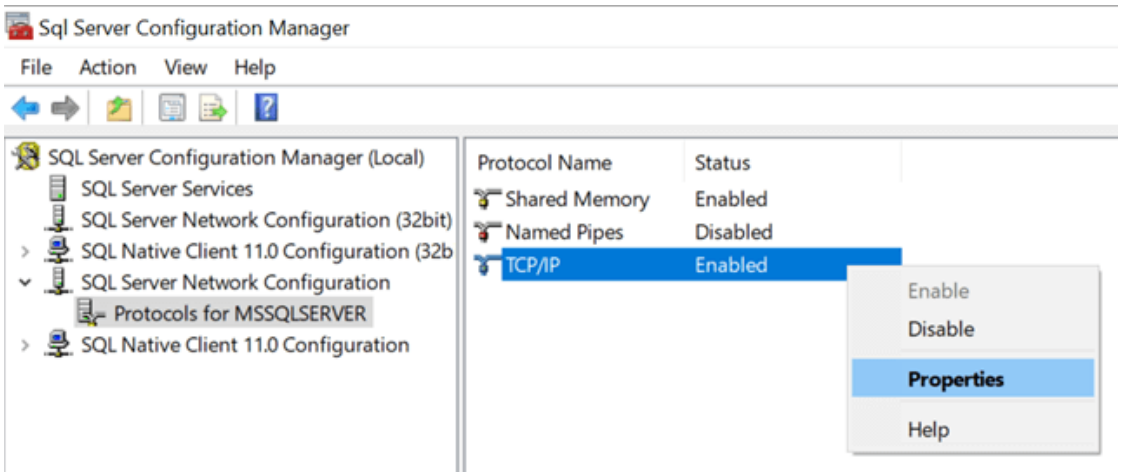


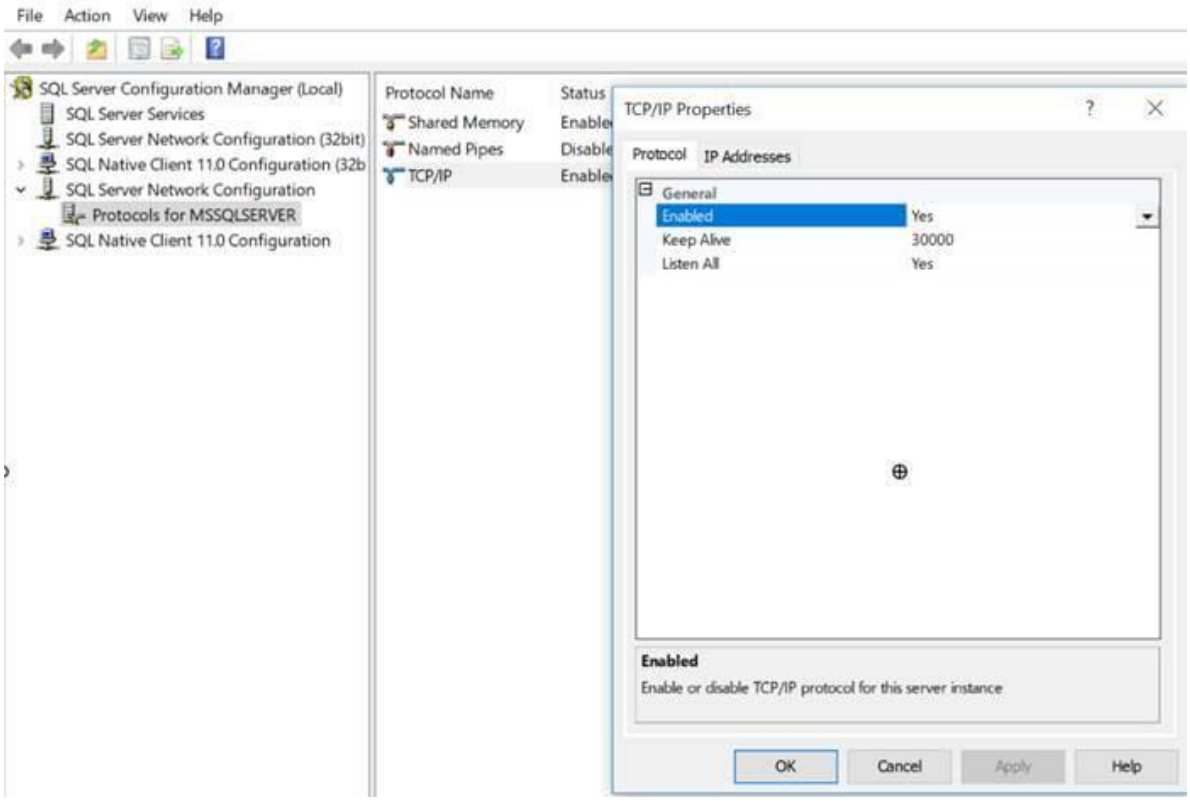
8. Running the same thing under IIS Express works fine i.e. on port 44312. Replace “values” with the product to get the product details,

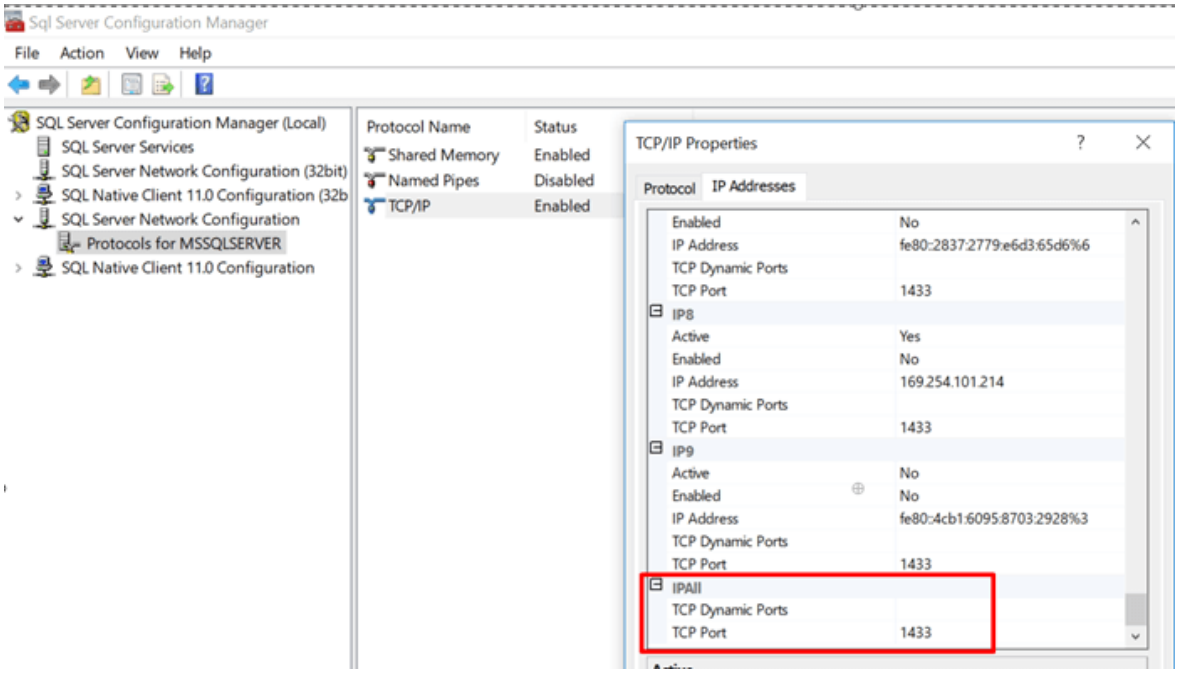
9. Since in IIS Express application runs fine and not in docker container, the error clearly shows that something is wrong with the SQL server that it does not understand our docker container or it is not running under docker container. In this scenario, the docker container is running as a separate machine inside the host computer. So, to connect to the SQL database in the host machine, remote connections to SQL needs to be enabled. We can fix this.

10. Open the SQL Server Configuration Manager. Now select Protocols for MSSQLSERVER and get the IPAll port number under TCP/IP section.

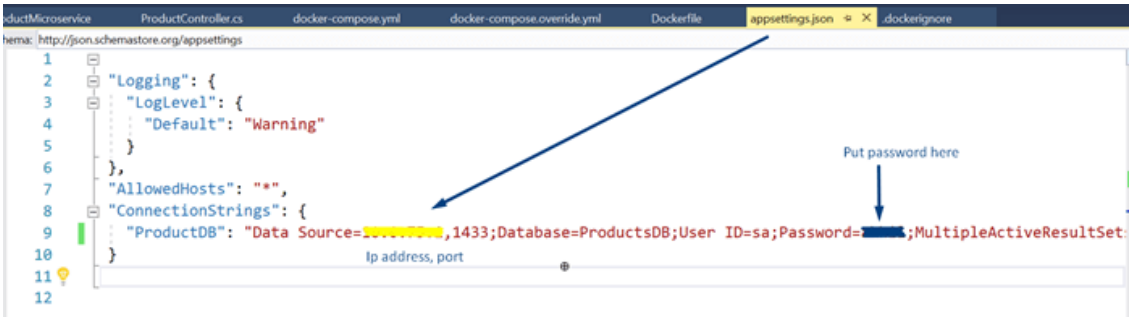

Sql Server Configuration Manager
File Action View Help
SQL Server Configuration Manager (Local)
SQL Server Services
SQL Server Network Configuration (32bit)
SQL Native Client 11.0 Configuration (32b
SQL Server Network Configuration
Protocols for MSSQLSERVER
SQL Native Client 11.0 Configuration
Protocol Name
Status
Shared Memory
Enabled
Named Pipes
Disabled
TCP/IP
Enabled


Sql Server Configuration Manager
File Action View Help
SQL Server Configuration Manager (Local)
SQL Server Services
SQL Server Network Configuration (32bit)
SQL Native Client 11.0 Configuration (32b
SQL Server Network Configuration
Protocols for MSSQLSERVER
SQL Native Client 11.0 Configuration
Protocol Name
Status
Shared Memory
Enabled
Named Pipes
Disabled
TCP/IP
Enabled
Enable
Disable
Properties
Help


File Action View Help
SQL Server Configuration Manager (Local)
SQL Server Services
SQL Server Network Configuration (32bit)
SQL Native Client 11.0 Configuration (32b
SQL Server Network Configuration
Protocols for MSSQLSERVER
SQL Native Client 11.0 Configuration
Protocol Name
Status
Shared Memory
Named Pipes
TCP/IP
TCP/IP Properties
Protocol
IP Addresses
General
Enabled
Yes
Keep Alive
30000
Listen All
Yes
Enabled
Enable or disable TCP/IP protocol for this server instance
OK
Cancel
Apply
Help


Sql Server Configuration Manager
File Action View Help
SQL Server Configuration Manager (Local)
SQL Server Services
SQL Server Network Configuration (32bit)
SQL Native Client 11.0 Configuration (32b
SQL Server Network Configuration
Protocols for MSSQLSERVER
SQL Native Client 11.0 Configuration
Protocol Name
Status
Shared Memory
Enabled
Named Pipes
Disabled
TCP/IP
Enabled
TCP/IP Properties
Protocol
IP Addresses
Enabled
No
IP Address
fe80:2837:2779:e6d3:65d6%6
TCP Dynamic Ports
TCP Port
1433
IP8
Active
Yes
Enabled
No
IP Address
169.254.101.214
TCP Dynamic Ports
TCP Port
1433
IP9
Active
No
Enabled
No
IP Address
fe80::4cb1:6095:8703:2928%3
TCP Dynamic Ports
TCP Port
1433
IPAll
TCP Dynamic Ports
TCP Port
1433

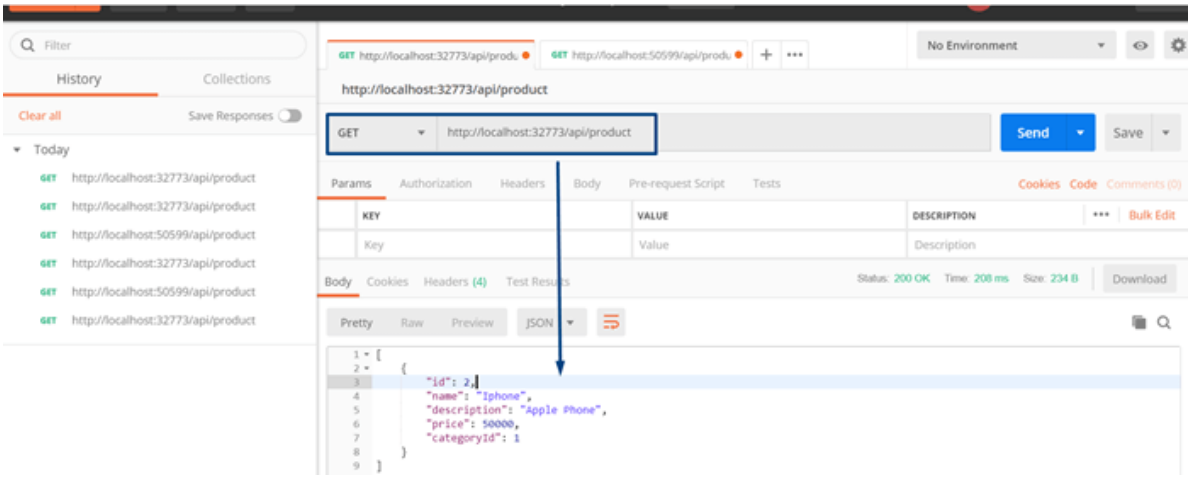
11. The connection string mentioned in the JSON file points to the data source as local which the docker container does not understand. It needs proper IP addresses with port and SQL authentication. So, provide the relevant details i.e. Data Source as Ip address, port number and SQL authentication details as shown below.



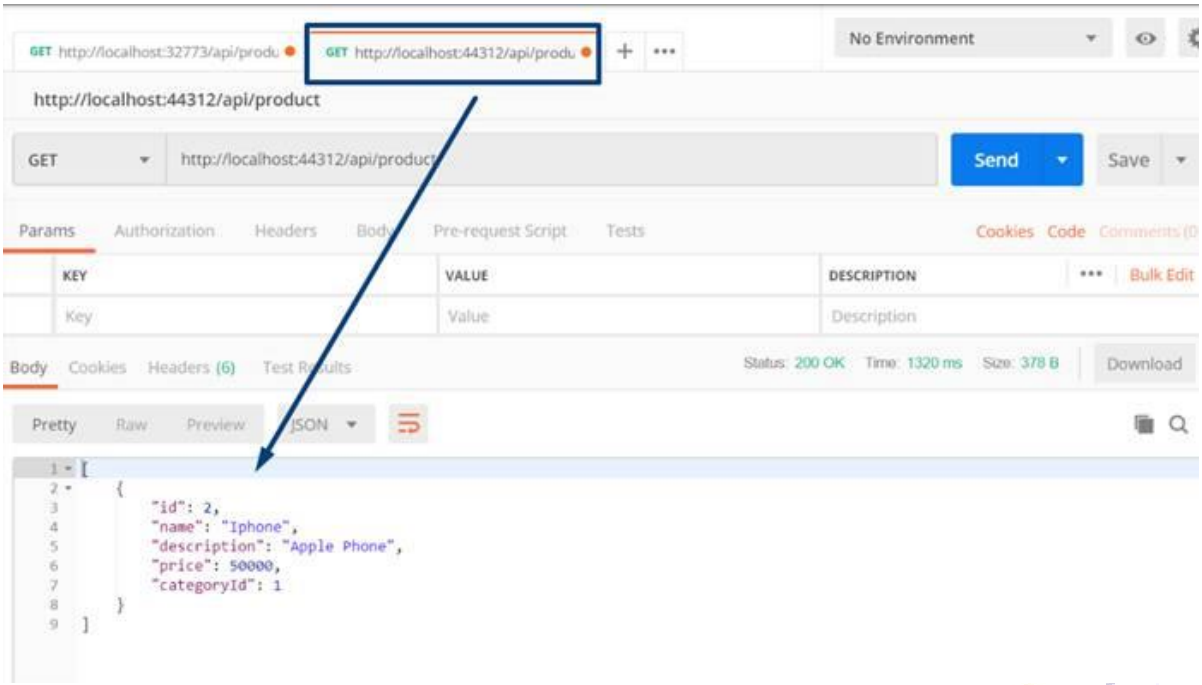
12. Now again run the application with Docker as an option like done earlier.

This time the response is received.

13. Test the same in the Postman.



14. Test again with IIS Express URL.



This proves that the microservice is running on two endpoints and on two operating systems independently locally deployed

Time Boxing

Activity Name	Activity Time	Total Time
Login Systems + Setting up Visual studio Environment	3 mints + 5 mints	8 mints
Walk through Theory & Tasks	60 mints	60 mints
Implement Tasks	80 mints	80 mints
Evaluation Time	30 mints	30 mints
	Total Duration	178 mints

Objectives

After completing this lab the student should be able to:

- Clearly understand the purpose and benefits that Cloud Computing has to offer.
- Understand the concept of Cloud Computing.
- Build a basic Cloud Computing base application.
- Use and learn Cloud Computing need.

Lab Tasks/Practical Work

1. Consume micro services for any website using asp.ne core.

Lab No. 13

ASP.NET Core Load Balancing

LAB # 13

ASP.NET Core Load Balancing

What the ASP.NET Core Load Balancer Will Do

The behaviour of a load balancer is quite simple so I avoid wasting time explaining what a balancer is. Anyway, I'll spend a few words describing how I decided to implement it.

Requirements

- Be plug and play: no complex installation
- Be standalone or integrated in web server (nginx, apache, iis)
- Changing configuration will provide: a proxy server, a balancing server, both of them
- Use as much as possible what ASP.NET Core gives out of the box
- Keeping in mind performances

Modules

The main idea is to define a set of "modules" that can be activated or not based on configuration. It has to be possible to add new modules and allow third parties to develop their one.

Filters

This module provides an easy way to filter request based on some rules. All requests that match the filter will be dropped. Each url is tested over a set of rules. If the url matches the rule, the request will be dropped. Only one match determines the rule activations so, basically, all rules are "OR" conditions by default. Each rule can test a set of request parameters (url, agent, headers). Inside the single rule, all conditions must be true to activate the rule. This means we are working with something like this (CONDITION A AND CONDITION B) OR (CONDITION C) and this will support most cases.

Caching

By using standard .NET Core caching module, we can provide cache support for url, defining policy, etc. Caching has many options that are basically a wrap of the original module, so you can refer here for more details.

Rewrite

This stage will allow static rewrite rule. This is often demanded to the applications but can be implemented here to simplify server part or to map virtual urls over many different applications

This is mostly a way to couple external url with internal one in case there isn't a way to change balanced application. Balancer itself will balance the output of this transformation.

Balancing

This is the core module that defines, for each url what will be the destination. This step generates only the real path, replacing selected host. The host can be selected using one of the following algorithms:

1. Number of requests coming
2. Number of requests pending
3. Quicker response
4. Affiliation (based on Cookie)

Proxy

After Balancing stage completes the computation of right url, proxy module will invoke the request replying to the client.

Time Boxing

Activity Name	Activity Time	Total Time
Login Systems + Setting up Visual studio Environment	3 mints + 5 mints	8 mints
Walk through Theory & Tasks	60 mints	60 mints
Implement Tasks	80 mints	80 mints
Evaluation Time	30 mints	30 mints
	Total Duration	178 mints

Objectives

After completing this lab the student should be able to:

- Clearly understand the purpose and benefits that Cloud Computing has to offer.
- Understand the concept of Cloud Computing.
- Build a basic Cloud Computing base application.
- Use and learn Cloud Computing need.

Lab Tasks/Practical Work

1. Consider any real world scenario and create a website using load balancing

Lab No. 14

Authentication in WEB API

LAB # 14

Authentication in Web API

Introduction

Authentication is used to protect our applications and websites from unauthorized access and also, it restricts the user from accessing the information from tools like postman and fiddler. In this article, we will discuss basic authentication, how to call the API method using postman, and consume the API using jQuery Ajax.

To access the web API method, we have to pass the user credentials in the request header. If we do not pass the user credentials in the request header, then the server returns 401 (unauthorized) status code indicating the server supports Basic Authentication.

Achieve Basic Authentication

Follow the below steps for Basic Authentication.

Step 1

Let us create a class BasicAuthenticationAttribute which inherits from the AuthorizationFilterAttribute **(**namespace System.Web.Http.Filters;**)** and overrides the method OnAuthorization from the base class (AuthorizationFilterAttribute).

The OnAuthorization method has a parameter action-context which provides access to the request and response object.

Code

```
1.  namespace BasicAuthentication
2.
3.    public class BasicAuthenticationAttribute : AuthorizationFilterAttribute
4.    {
5.       public override void OnAuthorization(HttpActionContext actionContext)
6.       {
7.          base.OnAuthorization(actionContext);
8.       }
9.    }
```

Now, we use the actionContext object to check if the request header is null or not. If null, then we return 401(unauthorized) status code; if not null, then we use the request header authorization

parameter for authorization and these parameters are formatted as the string “Username: Password” base64-encoded.

Code

```
public override void OnAuthorization(HttpActionContext actionContext)
{
    if (actionContext.Request.Headers.Authorization != null)
    {
        var authToken = actionContext.Request.Headers
            .Authorization.Parameter;

        // decoding authToken we get decode value in 'Username:Password' format
        var decodeauthToken = System.Text.Encoding.UTF8.GetString(
            Convert.FromBase64String(authToken));

        // spliting decodeauthToken using ':'
        var arrUserNameandPassword = decodeauthToken.Split(':');

        // at 0th postion of array we get username and at 1st we get password
        if (IsAuthorizedUser(arrUserNameandPassword[0], arrUserNameandPassword[1]))
        {
            // setting current principle
            Thread.CurrentPrincipal = new GenericPrincipal(
                new GenericIdentity(arrUserNameandPassword[0]), null);
        }
        else
        {
            actionContext.Response = actionContext.Request
                .CreateResponse(HttpStatusCode.Unauthorized);
        }
    }
    else
    {
        actionContext.Response = actionContext.Request
            .CreateResponse(HttpStatusCode.Unauthorized);
    }
}
```

Now, we need to decode the base64-encoded value and split by using ‘**:**’. After the split, we get the username at the 0^{th} position and the password at the 1^{st} position. Then, we pass the username and password to the below method to check whether a user is authorized or not.

Code

```
1.  public static bool IsAuthorizedUser(string Username, string Password)
2.  {
3.      // In this method we can handle our database logic here...
4.      return Username == "bhushan" && Password == "demo";
5.  }
```

If the above method returns true, then we create Generic Principle and set it to currentprinciple. The generic principle has two parameters - GenericIdentity and Roles.

If the methods return false, then we return 401(unauthorized) status code.

We can define BasicAuthenticationAttribute globally, at Controller and at View. To define the basic authentication, we have to create a controller.

If we want to declare globally, we will declare it in WebApiConfig.cs.

```
1.  config.Filters.Add(new BasicAuthenticationAttribute());
```

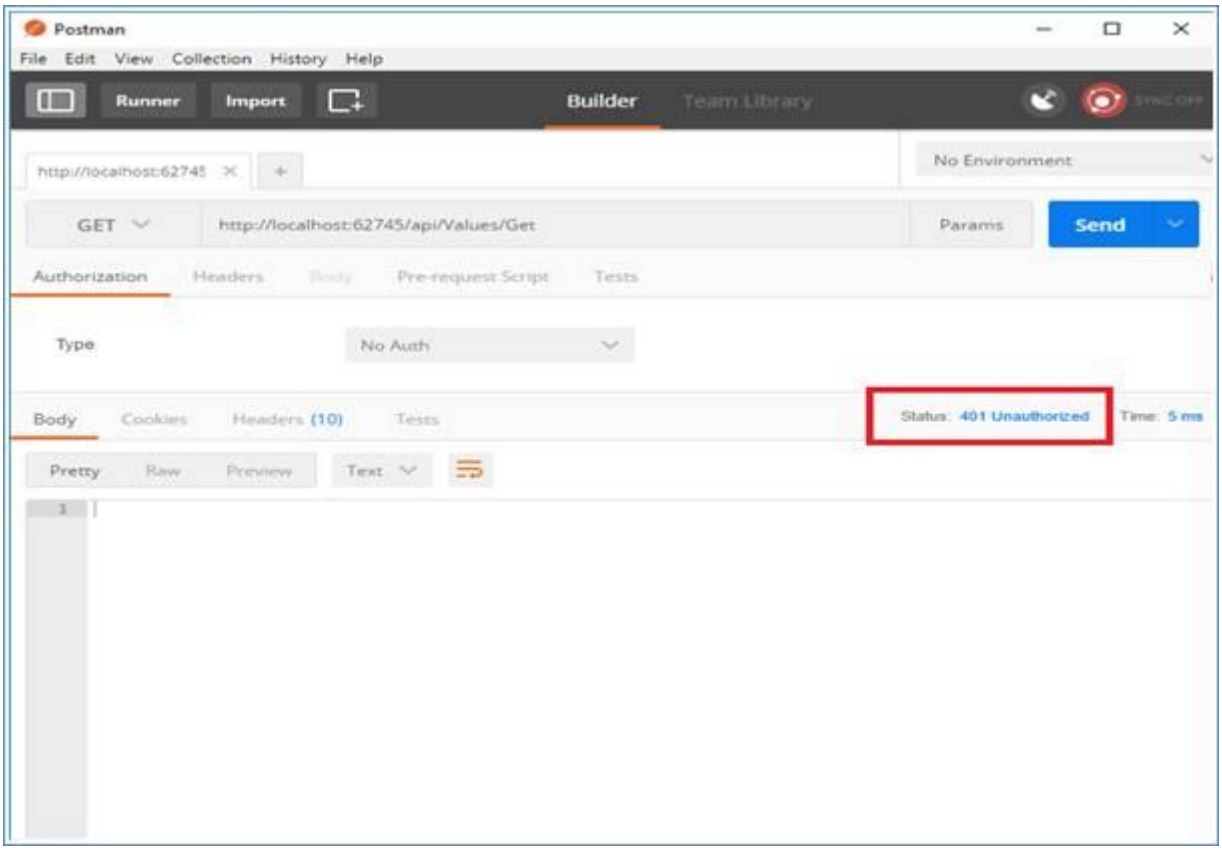
Step 2

In this step, let us create a controller and decorate the Get method with BasicAuthentication.

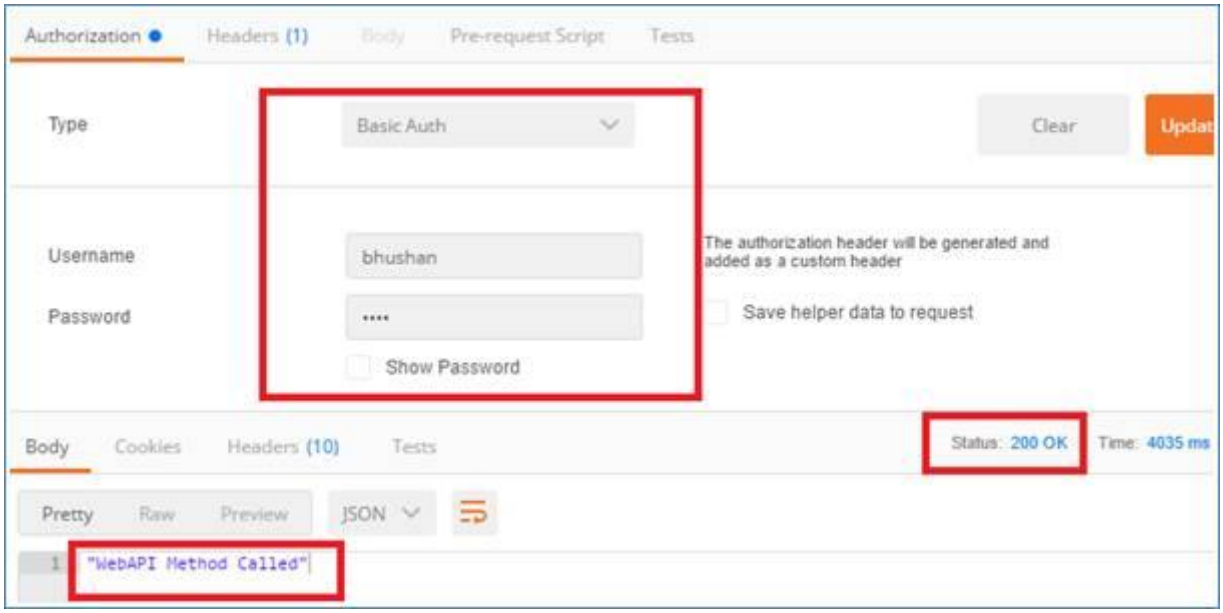
Code

```
1.  namespace BasicAuthentication.Controllers
2.  {
3.
4.      public class ValuesController : ApiController
5.      {
6.          [BasicAuthentication]
7.          public string Get()
8.          {
9.              return "WebAPI Method Called";
10.         }
11.     }
12. }
```

When we hit the URL in postman without adding Basic Authentication in request header, this will return 401 Status code.



When we add authorization and pass the credentials, it will allow us to access the Get method and return the status 200.



To access the above Web API method using jQuery AJAX, use the following code.

Script

```
<script type="text/javascript">
        $.ajax({
            type: 'GET',
            url: "api/values/Get",
            datatype: 'json',
            headers:
                {
                    Authorization: 'Basic ' + btoa(username + ':' + password)
                },
            success: function (data) {
            },
            error: function (data) {
            }
        });
</script>
```

Time Boxing

Activity Name	Activity Time	Total Time
Login Systems + Setting up Visual studio Environment	3 mints + 5 mints	8 mints
Walk through Theory & Tasks	60 mints	60 mints
Implement Tasks	80 mints	80 mints
Evaluation Time	30 mints	30 mints
	Total Duration	178 mints

Objectives

After completing this lab the student should be able to:

- Clearly understand the purpose and benefits that Cloud Computing has to offer.
- Understand the concept of Cloud Computing.
- Build a basic Cloud Computing base application.
- Use and learn Cloud Computing need.

Lab Tasks/Practical Work

1. Implement secure web API for a bank transaction using post man.