ASP.NET MVC Entity Framework

SummitWorks™
GLOBAL SOLUTION ARCHITECTS

Authorized & published by Summitworks Technologies Inc

Agenda

- Working with Views
  - Understanding Views
  - Adding a View Page
  - Create Master Page Layout
  - RenderBody, RenderPage, and RenderSection
  - Working with Razor View Engine
  - Passing Data from View to Controller
  - Partial Views
    - How to create a Partial View
    - How to consume a Partial View
- Working with Models
  - Understanding Model
  - Adding a Model
  - Insert, Update, Delete without Entity Framework
  - Insert, Update, Delete using Entity Framework
  - Scaffolding Example

Working with Views

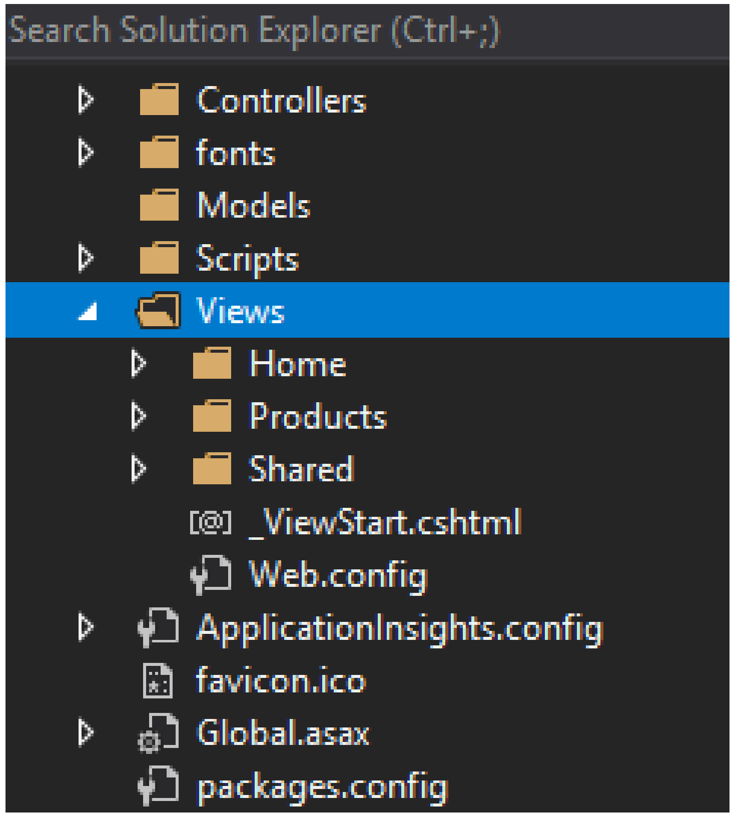
Understanding Views

View is a user interface. View displays data from the model to the user and also enables them to modify the data.

ASP.NET MVC views are stored in **Views** folder. Different action methods of a single controller class can render different views, so the Views folder contains a separate folder for each controller with the same name as controller, in order to accommodate multiple views.

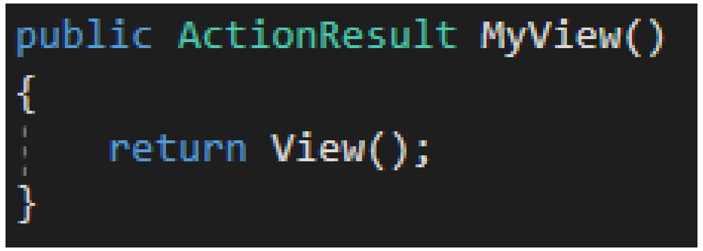
ASP.NET MVC supports following types of view files:

View file extension	Description
.cshtml	C# Razor view. Supports C# with html tags.
.vbhtml	Visual Basic Razor view. Supports Visual Basic with html tags.
.aspx	ASP.Net web form
.ascx	ASP.NET web control



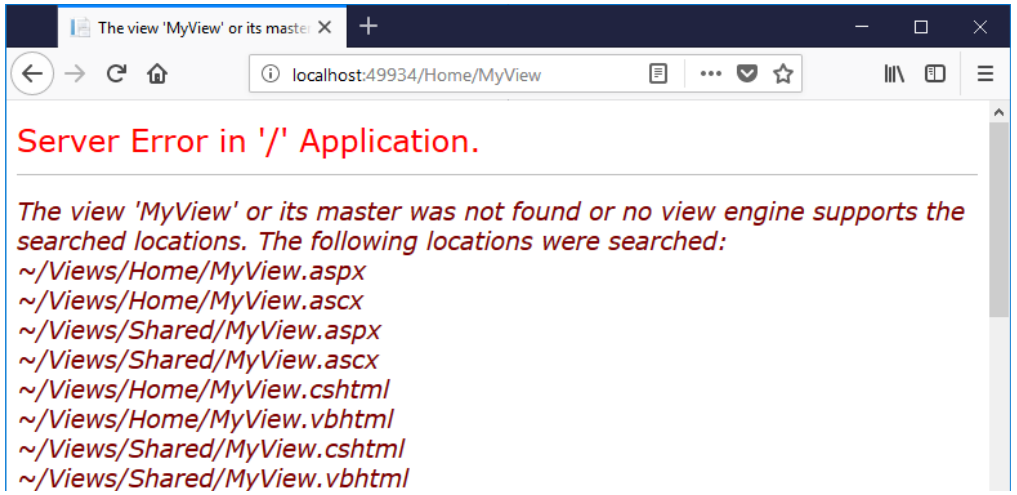
Adding a View Page

Before adding a view let's add an action into HomeController, which will return a default view.



```
public ActionResult MyView()
{
    return View();
}
```

Run this application and apend /Home/MyView to the URL in the browser and press enter. You will receive the following output.



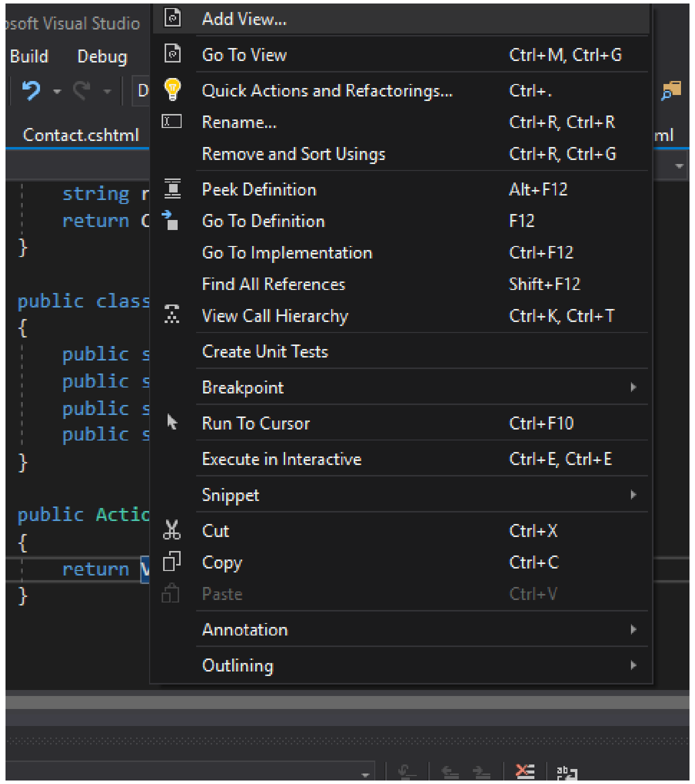
Server Error in '/' Application.

The view 'MyView' or its master was not found or no view engine supports the searched locations. The following locations were searched:
~/Views/Home/MyView.aspx
~/Views/Home/MyView.ascx
~/Views/Shared/MyView.aspx
~/Views/Shared/MyView.ascx
~/Views/Home/MyView.cshtml
~/Views/Home/MyView.vbhtml
~/Views/Shared/MyView.cshtml
~/Views/Shared/MyView.vbhtml

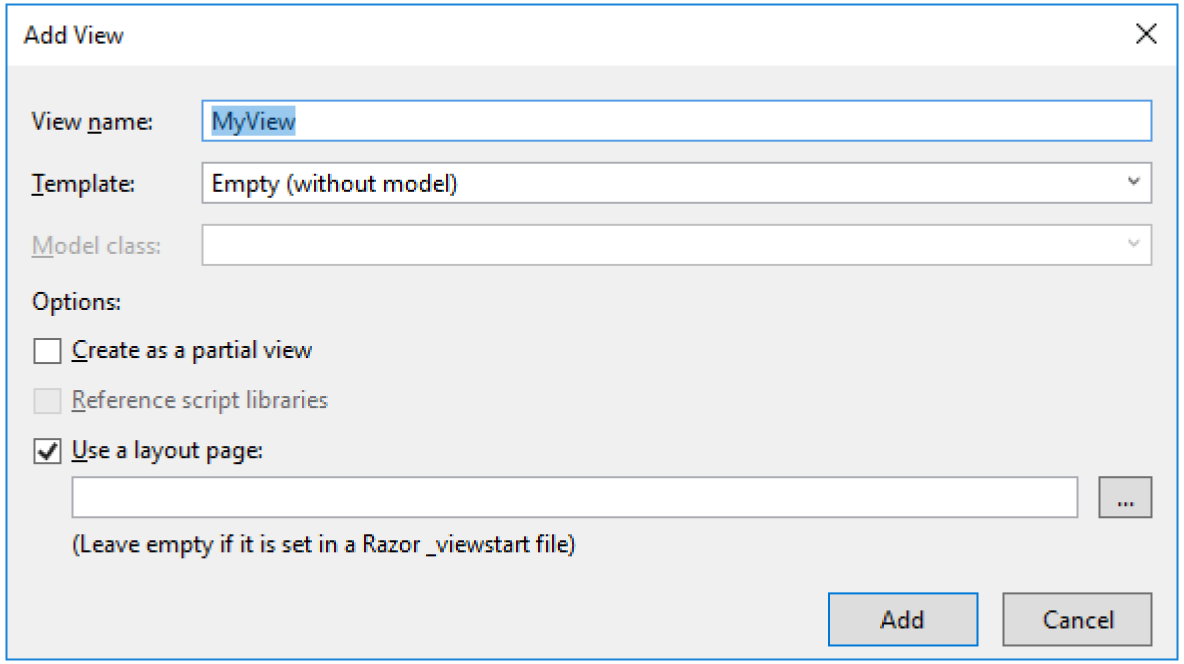
Adding a View Page

You can see in previous slide that we have an error and this error is actually quite descriptive, which tells us it can't find the MyView view.

To add a view, right-click inside the MyView action and select Add view.



Add View...
Go To View Ctrl+M, Ctrl+G
Quick Actions and Refactorings... Ctrl+.
Rename... Ctrl+R, Ctrl+R
Remove and Sort Usings Ctrl+R, Ctrl+G
Peek Definition Alt+F12
Go To Definition F12
Go To Implementation Ctrl+F12
Find All References Shift+F12
View Call Hierarchy Ctrl+K, Ctrl+T
Create Unit Tests
Breakpoint
Run To Cursor Ctrl+F10
Execute in Interactive Ctrl+E, Ctrl+E
Snippet
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Annotation
Outlining
Build
Debug
Contact.cshtml


Add View

View name: MyView

Template: Empty (without model)

Model class:

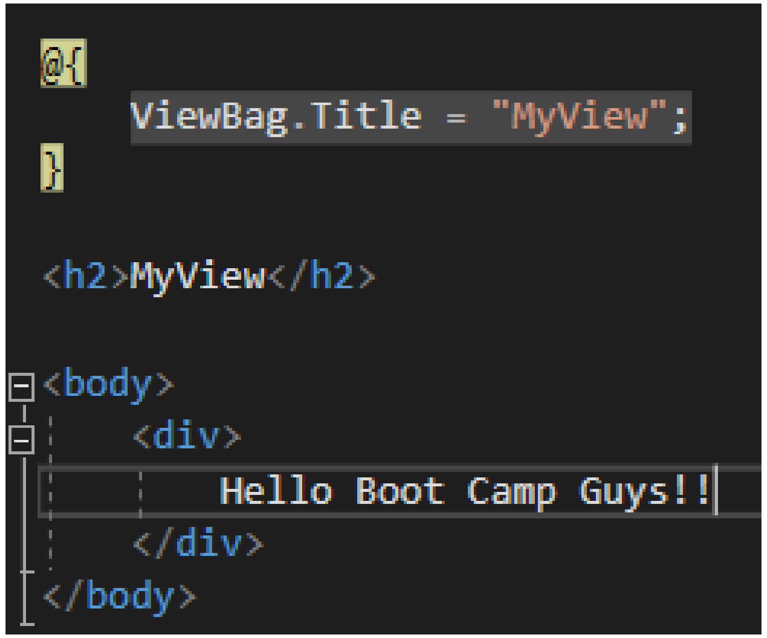
Options:

- [ ] Create as a partial view
- [ ] Reference script libraries
- [x] Use a layout page:

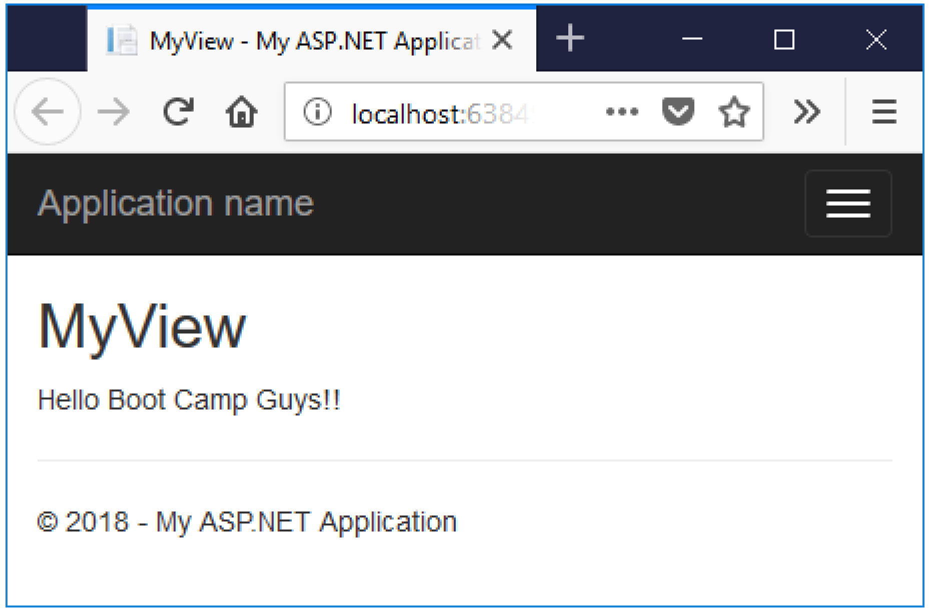
(Leave empty if it is set in a Razor _viewstart file)

Adding a View Page

Run this application and apend /Home/MyView to the URL in the browser. Press enter and you will receive the following output.



```
@{
    ViewBag.Title = "MyView";
}

<h2>MyView</h2>

<body>
    <div>
        Hello Boot Camp Guys!!
    </div>
</body>
```



MyView

Hello Boot Camp Guys!!

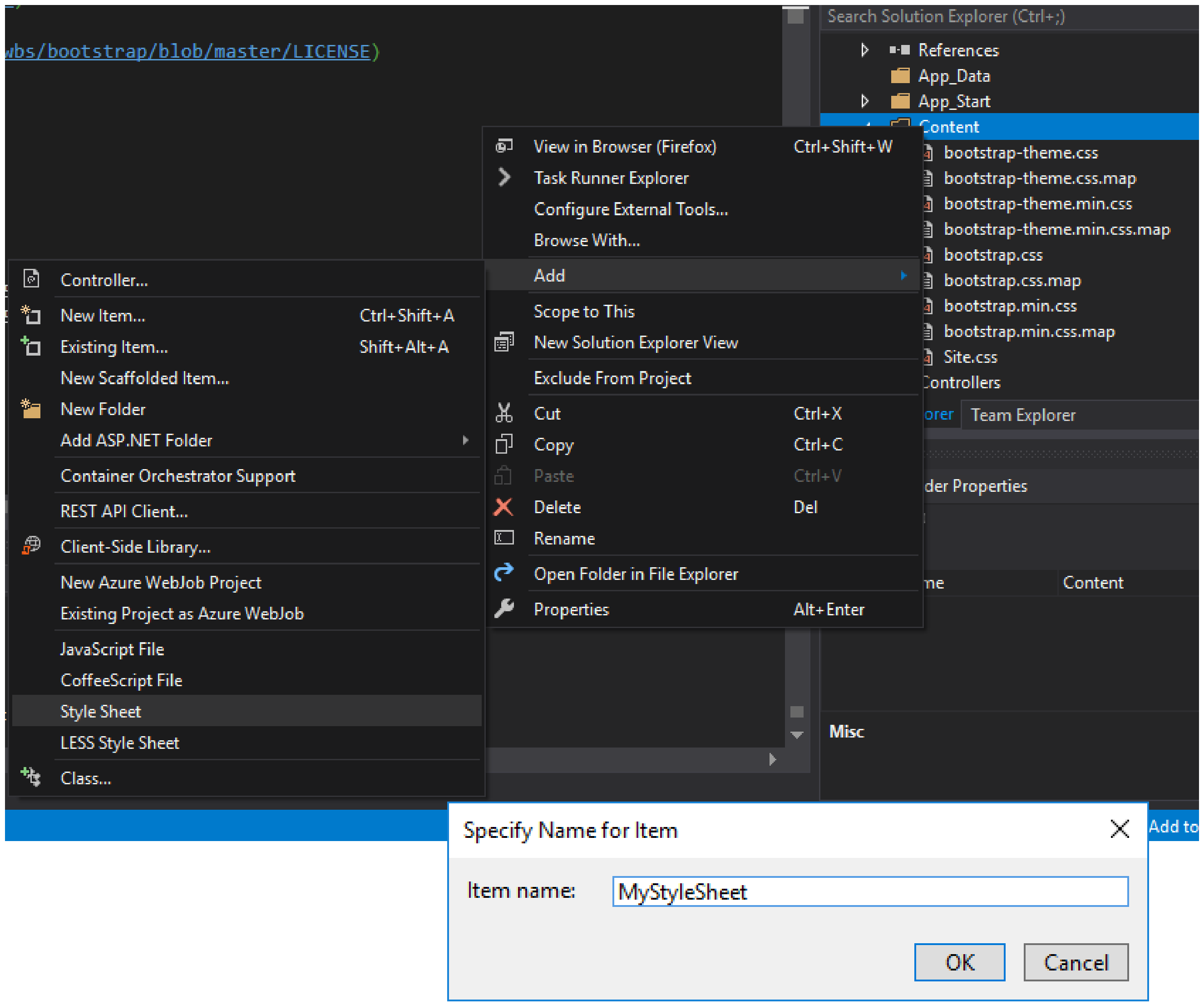
© 2018 - My ASP.NET Application

Create Master Page Layout

Layouts are used in MVC to provide a consistent look and feel on all the pages of our application. It is the same as defining the Master Pages but MVC provides some more functionalities.

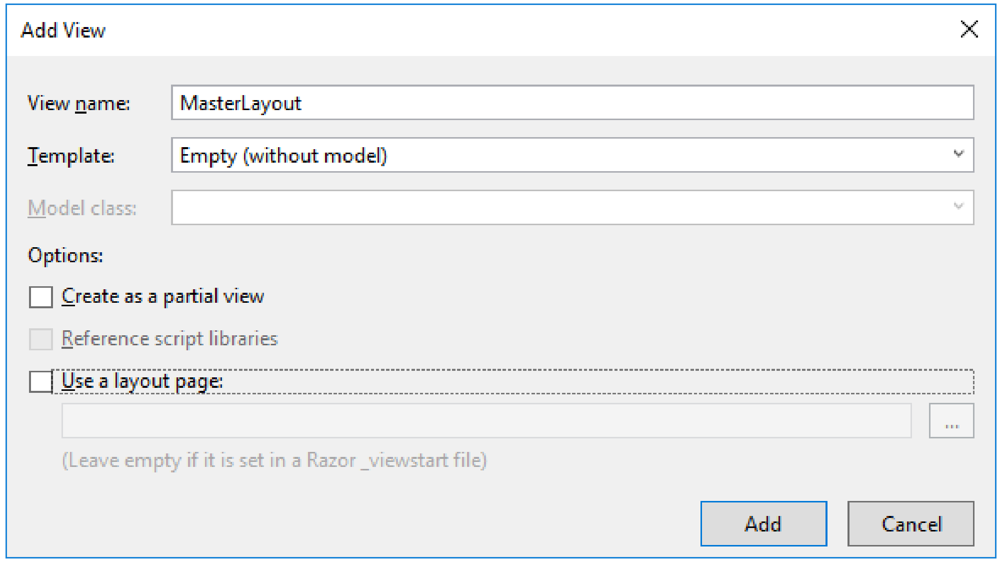
Create MVC Master Page Layout

Create a Style Sheet file named MyStyleSheet.css under the CONTENT folder. This CSS file will contain all the CSS classes necessary for a consistent web application page design.



Create Master Page Layout

Create a MasterLayout.cshtml file under the Shared folder. The file MasterLayout.cshtml represents the layout of each page in the application. Right-click on the Shared folder in the Solution Explorer, then go to Add item and click View.


Add View

View name: MasterLayout

Template: Empty (without model)

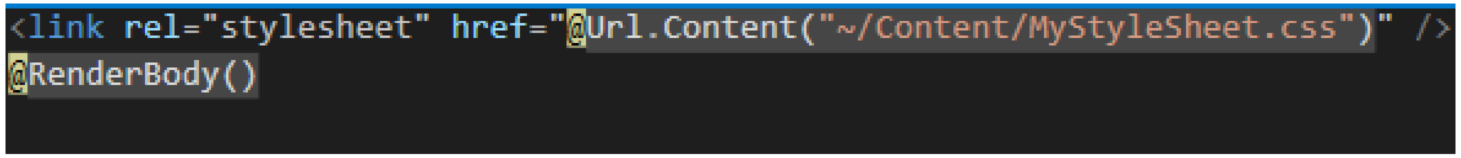
Model class:

Options:

- [ ] Create as a partial view
- [ ] Reference script libraries
- [ ] Use a layout page:

(Leave empty if it is set in a Razor _viewstart file)

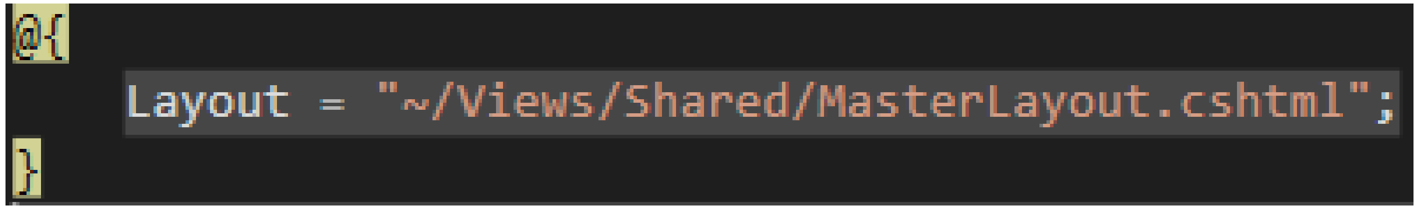
Copy the following layout code:



```
<link rel="stylesheet" href="@Url.Content("~/Content/MyStyleSheet.css")" />
@RenderBody()
```

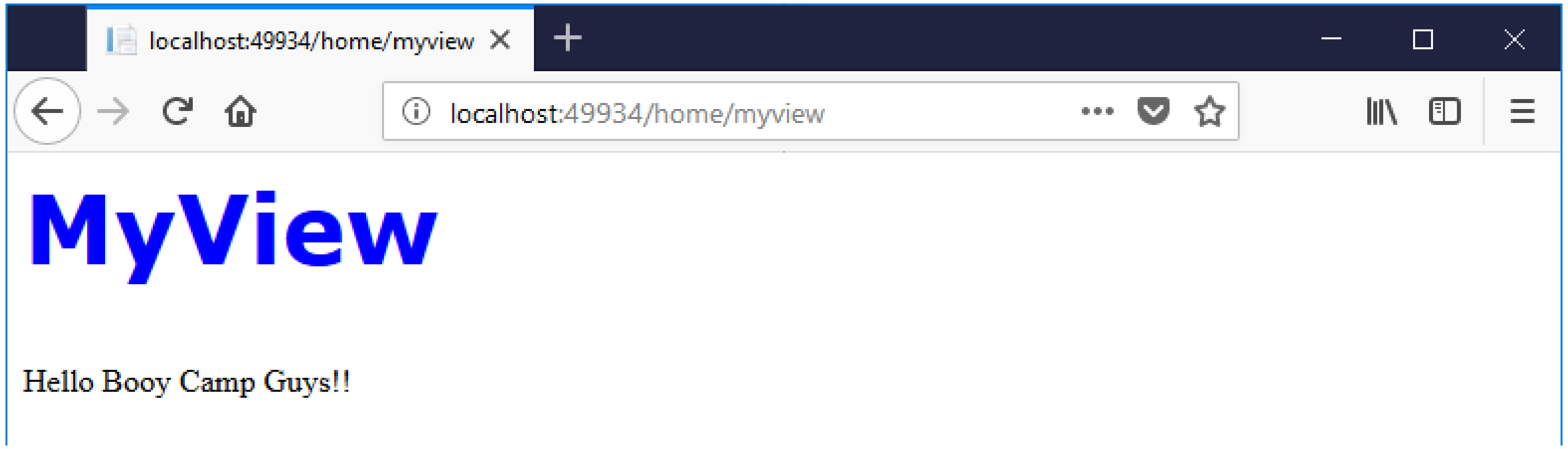
Create Master Page Layout

Finally, open the _ViewStart.cshtml file inside Views folder and add the following code



```
@{
    Layout = "~/Views/Shared/MasterLayout.cshtml";
}
```

Run the application now to see the modified home page.



localhost:49934/home/myview

MyView

Hello Booy Camp Guys!!

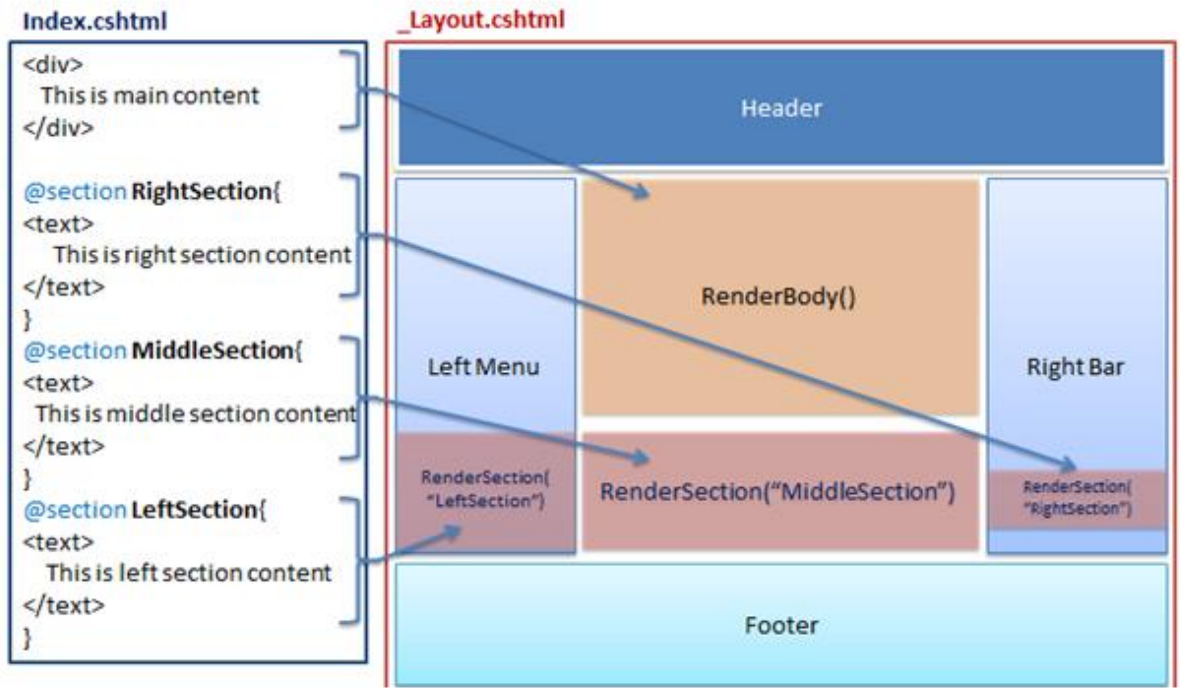
Rendering Methods

ASP.NET MVC layout view renders child views using the following methods.

RenderBody
Renders the portion of the child view that is not within a named section. Layout view must include RenderBody() method.

RenderSection
Renders a content of named section and specifies whether the section is required. RenderSection() is optional in Layout view.

Render Page
RenderPage also exists in Layout page and multiple RenderPage() can be there in Layout page.


Index.cshtml

```
<div>
  This is main content
</div>

@section RightSection{
<text>
   This is right section content
</text>
}
@section MiddleSection{
<text>
  This is middle section content
</text>
}
@section LeftSection{
<text>
  This is left section content
</text>
}
```

_Layout.cshtml
Header
RenderBody()
Left Menu
Right Bar
RenderSection("LeftSection")
RenderSection("MiddleSection")
RenderSection("RightSection")
Footer

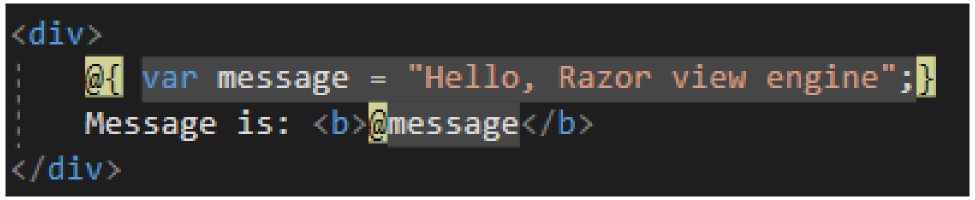
Working with Razor View Engine

Razor Syntax

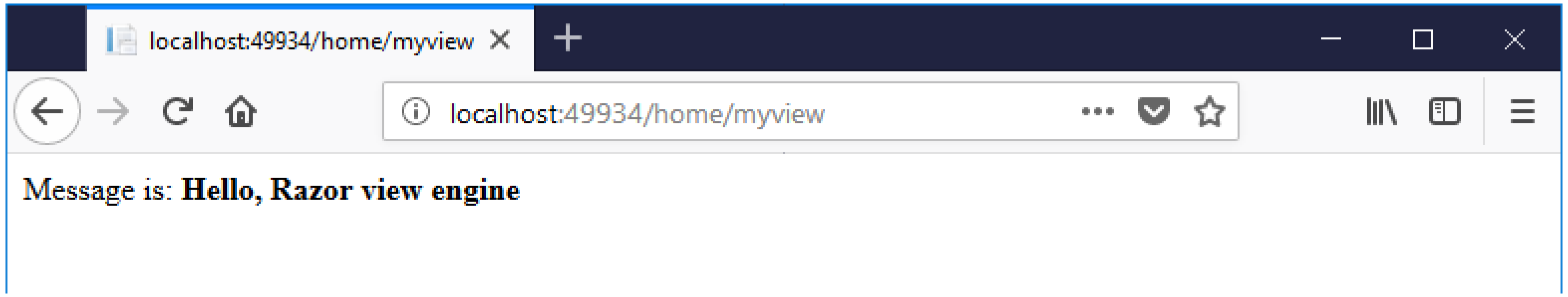
To understand the Razor View Engine we should learn its syntax so let's start with the syntax.

Single statement block and inline expression

Each code block will be start and end by opening and closing curly brackets {..} respectively. A statement is defined in a code block, in other words between opening and closing curly brackets and ending with a semicolon (";"). But when it's used as an inline expression then it does not need to use a semicolon.



```
<div>
    @{ var message = "Hello, Razor view engine";}
    Message is: <b>@message</b>
</div>
```

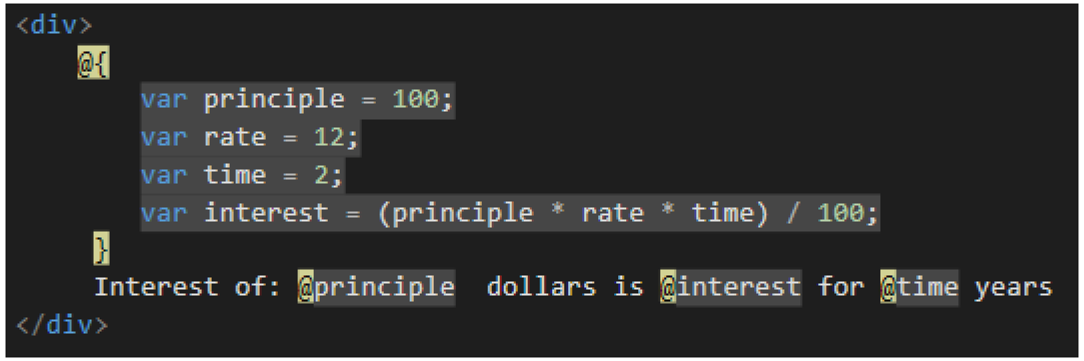


Message is: **Hello, Razor view engine**

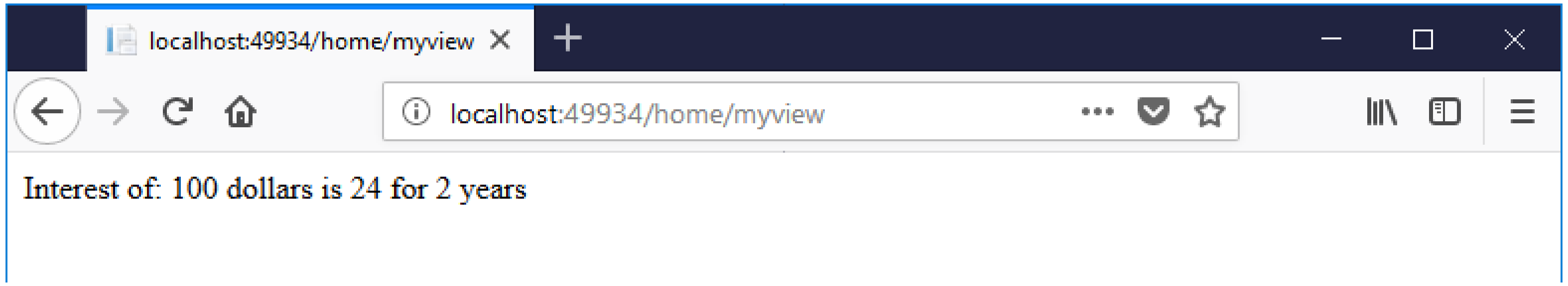
Working with Razor View Engine

Multi statement block

We can also define a multiline statement block as a single-line statement block. In a multiline statement block we can define multiple code statements and can process data. A multiline block will exist between opening and closing curly braces but the opening brace will have the "@" character in the same line if we define the "@" and opening curly braces in different lines then it will generate an error.



```
<div>
    @{
        var principle = 100;
        var rate = 12;
        var time = 2;
        var interest = (principle * rate * time) / 100;
     }
     Interest of: @principle  dollars is @interest for @time years
</div>
```



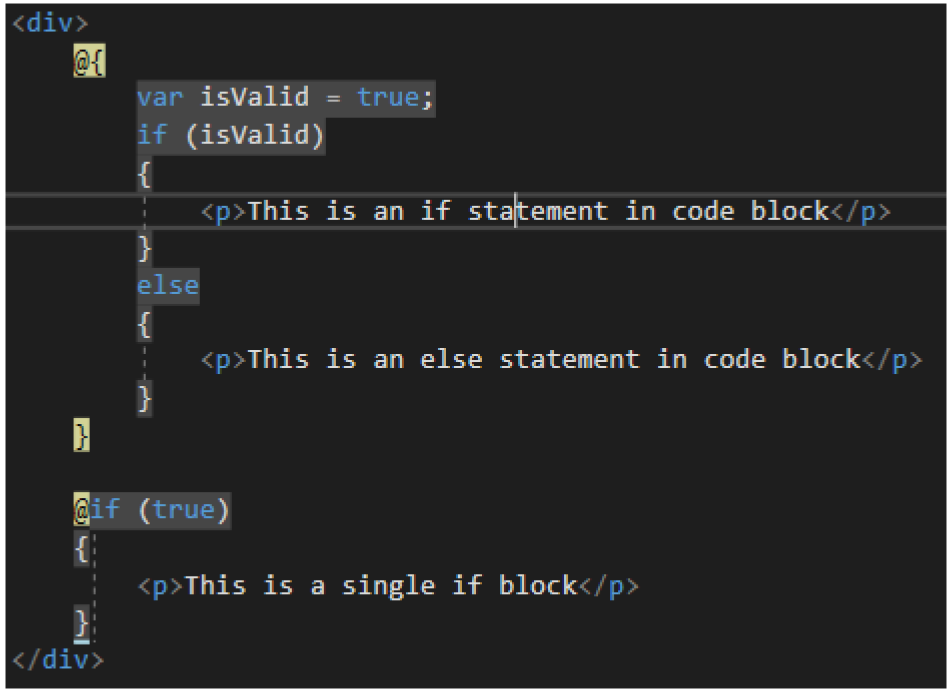
localhost:49934/home/myview

```
Interest of: 100 dollars is 24 for 2 years
```

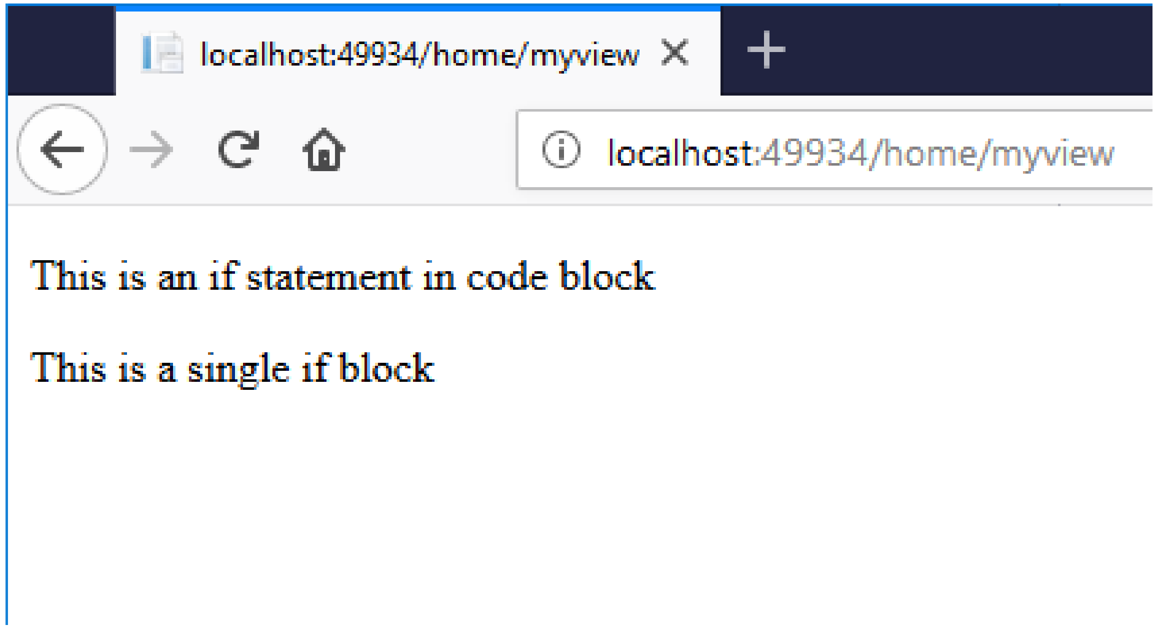
Working with Razor View Engine

Conditional statements

We can create a dynamic web page in Razor View Engine as condition based. We can define a condition inside a code block or outside the code block. The If statement works the same as it does in other programming languages.



```
<div>
    @{
        var isValid = true;
        if (isValid)
        {
            <p>This is an if statement in code block</p>
        }
        else
        {
            <p>This is an else statement in code block</p>
        }
    }

    @if (true)
    {
        <p>This is a single if block</p>
    }
</div>
```

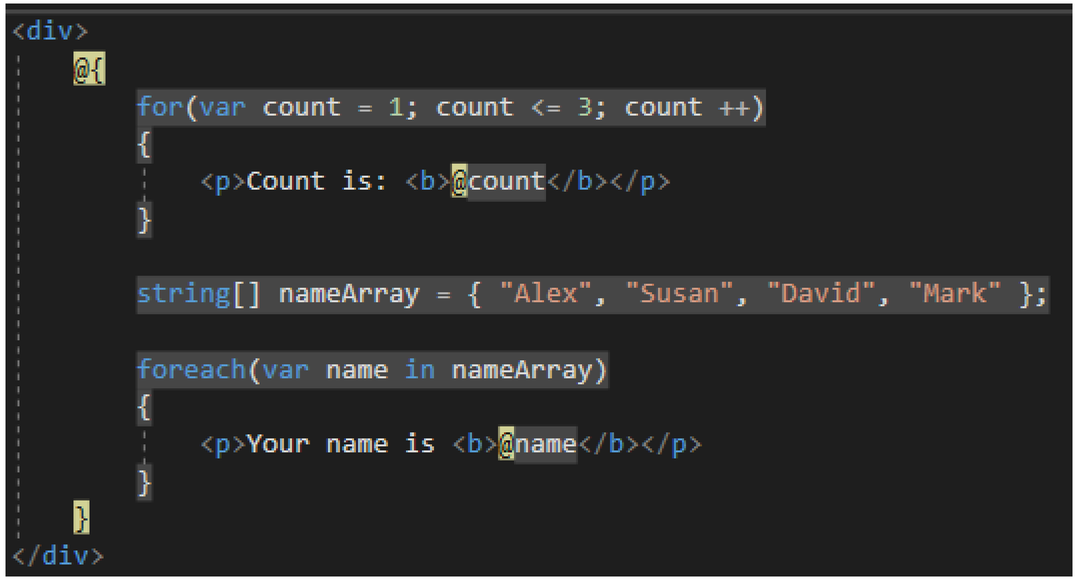


```
This is an if statement in code block

This is a single if block
```

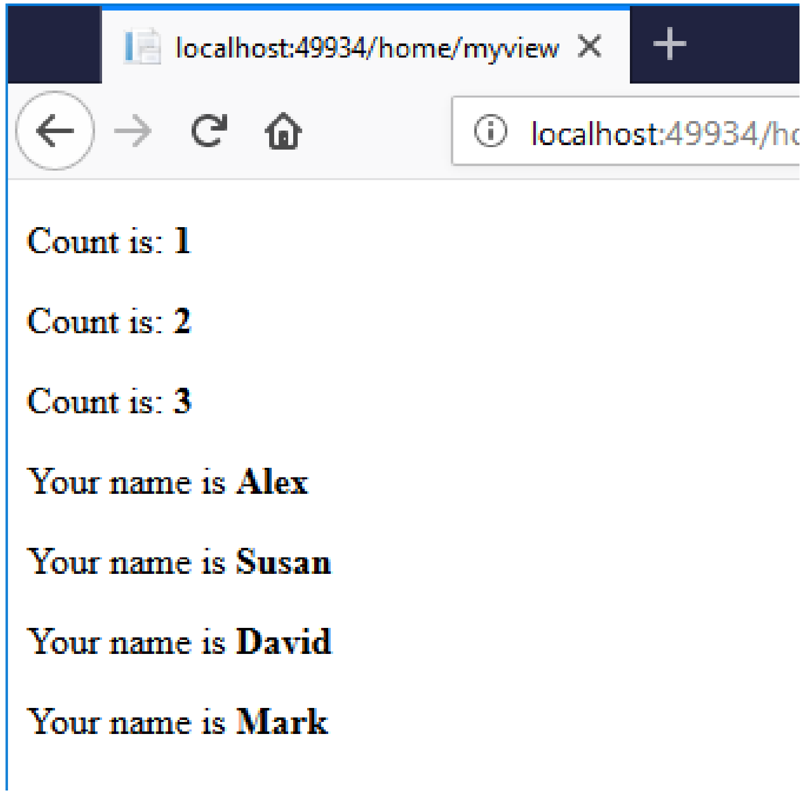
Working with Razor View Engine

Looping

All loops work the same as in other programming languages, we can define looping inside a code or outside a code block. We can define a for, do while or while loop in a code block and use the same syntax for initialization, increment/decrement and to check a condition.



```
<div>
    @{
        for(var count = 1; count <= 3; count ++)
        {
            <p>Count is: <b>@count</b></p>
        }

        string[] nameArray = { "Alex", "Susan", "David", "Mark" };

        foreach(var name in nameArray)
        {
            <p>Your name is <b>@name</b></p>
        }
    }
</div>
```



localhost:49934/home/myview

localhost:49934/hc

Count is: **1**

Count is: **2**

Count is: **3**

Your name is **Alex**

Your name is **Susan**

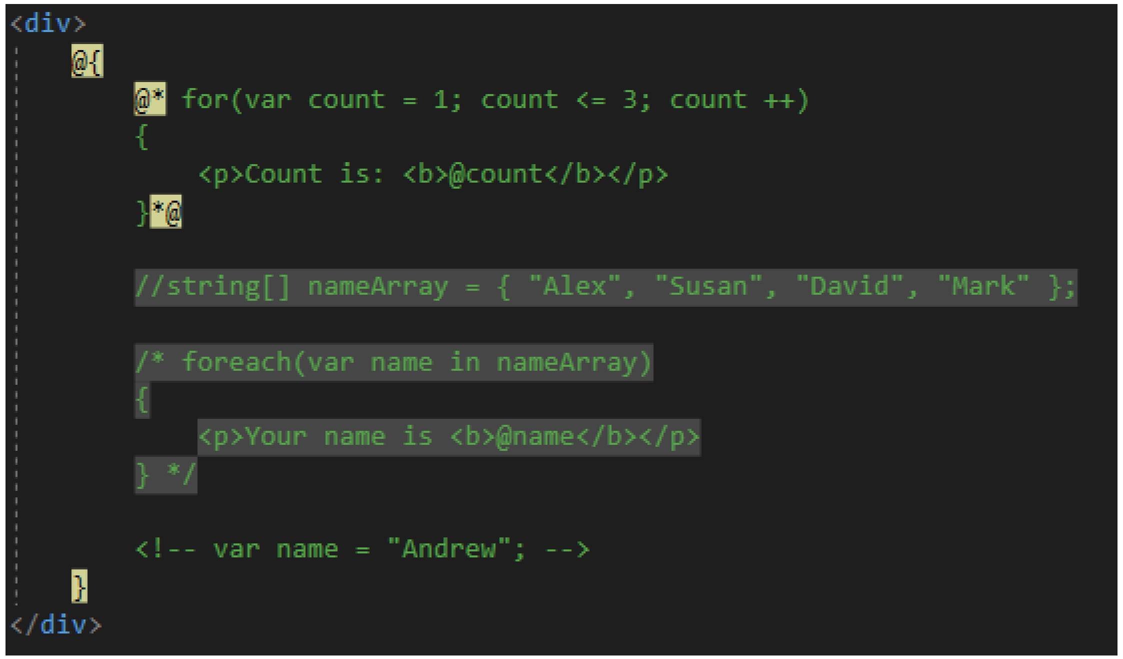
Your name is **David**

Your name is **Mark**

Working with Razor View Engine

Comments

Razor View Engine has two types of comments, one is single-line and another is multiline. Razor uses the syntax "@* ..*@" for the comment block but in a C# code block we can also use "/* */" or "//". HTML comments are the same, "<!-- -->".



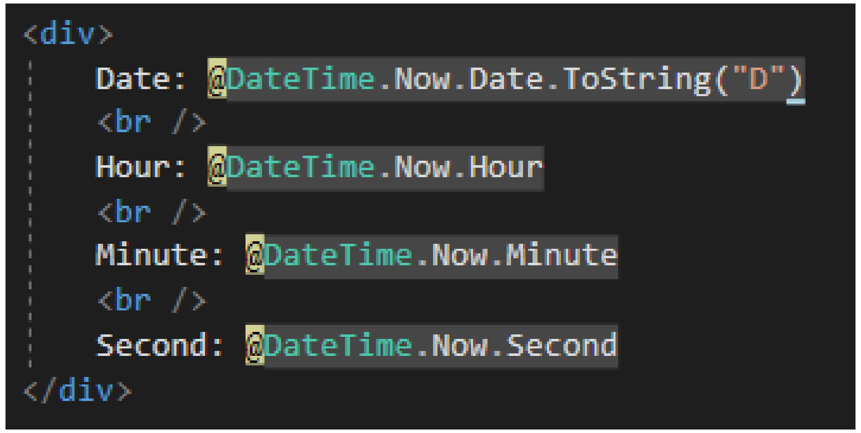
```
<div>
    @{
        @* for(var count = 1; count <= 3; count ++)
        {
            <p>Count is: <b>@count</b></p>
        }*@

        //string[] nameArray = { "Alex", "Susan", "David", "Mark" };

        /* foreach(var name in nameArray)
        {
            <p>Your name is <b>@name</b></p>
        } */

        <!-- var name = "Andrew"; -->
    }
</div>
```

Working with Razor View Engine

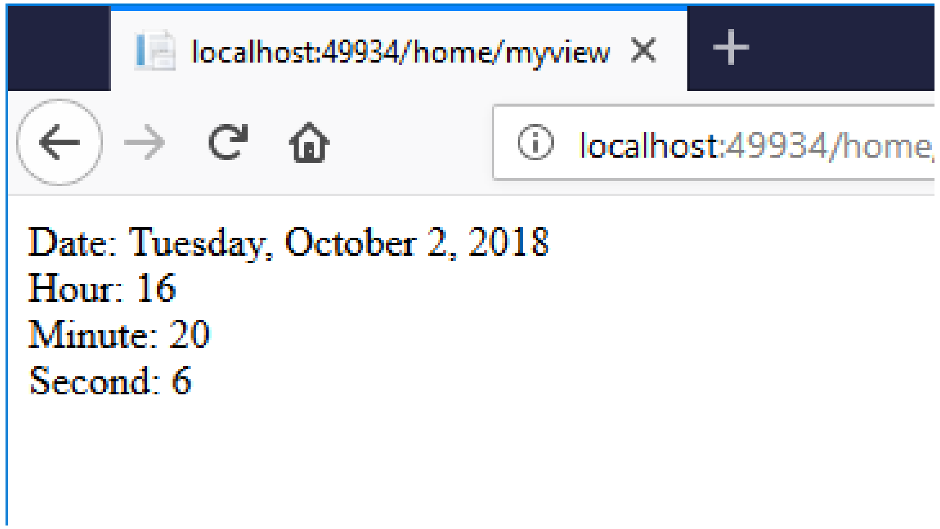
Use of Object

We can also use an object in both a code block and HTML using razor syntax.

Here we used a Date object of the C# language and access the properties of its.



```
<div>
    Date: @DateTime.Now.Date.ToString("D")
    <br />
    Hour: @DateTime.Now.Hour
    <br />
    Minute: @DateTime.Now.Minute
    <br />
    Second: @DateTime.Now.Second
</div>
```

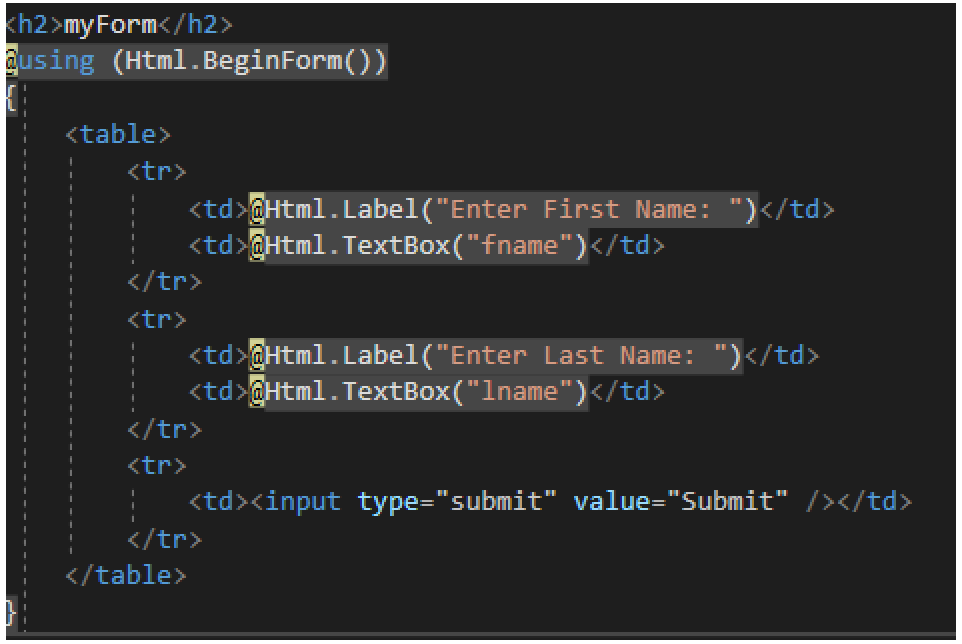


```
Date: Tuesday, October 2, 2018
Hour: 16
Minute: 20
Second: 6
```

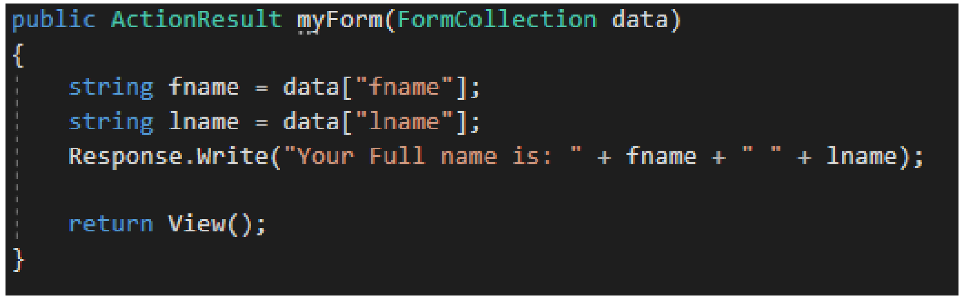
Passing Data from View to Controller

Using Form Collection Object

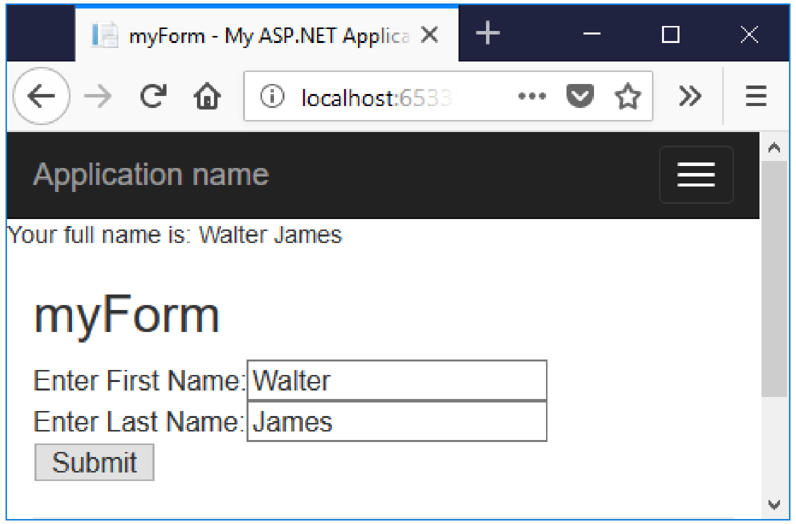
The FormCollection object also has requested data in the name/value collection as the Request object. To get data from the FormCollection object we need to pass it is as a parameter and it has all the input field data submitted on the form.



```
<h2>myForm</h2>
@using (Html.BeginForm())
{
    <table>
        <tr>
            <td>@Html.Label("Enter First Name: ")</td>
            <td>@Html.TextBox("fname")</td>
        </tr>
        <tr>
            <td>@Html.Label("Enter Last Name: ")</td>
            <td>@Html.TextBox("lname")</td>
        </tr>
        <tr>
            <td><input type="submit" value="Submit" /></td>
        </tr>
    </table>
}
```



```
public ActionResult myForm(FormCollection data)
{
    string fname = data["fname"];
    string lname = data["lname"];
    Response.Write("Your Full name is: " + fname + " " + lname);

    return View();
}
```



Your full name is: Walter James

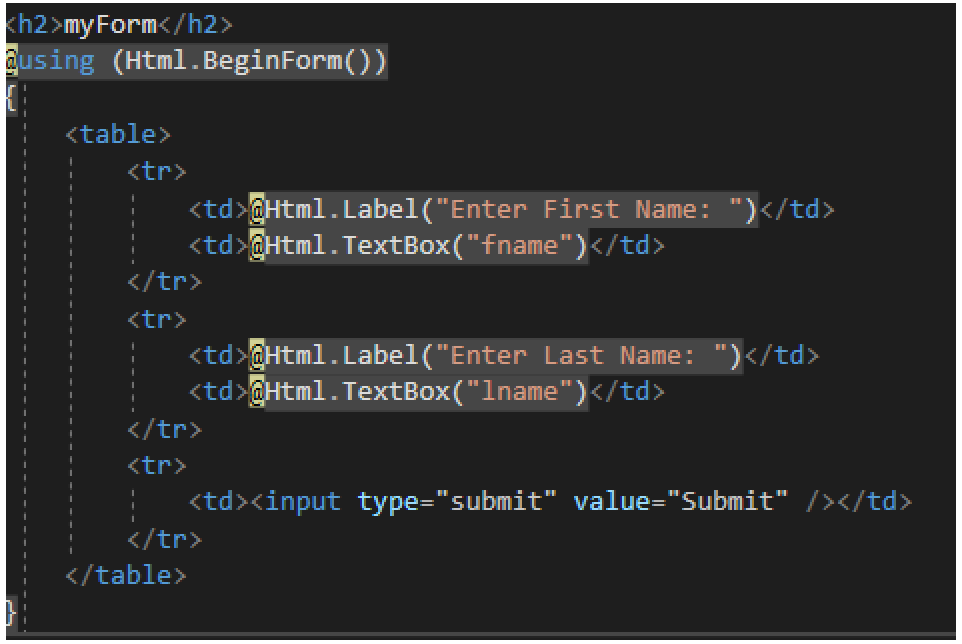
myForm

Enter First Name: Walter
Enter Last Name: James
Submit

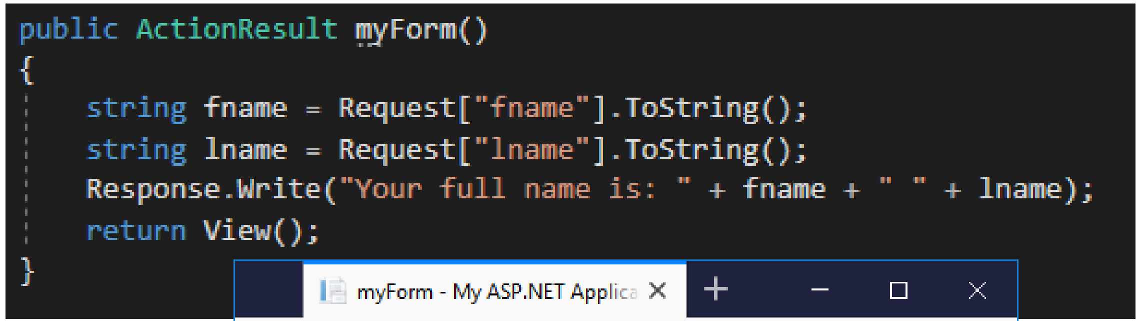
Passing Data from View to Controller

Using Traditional Approach

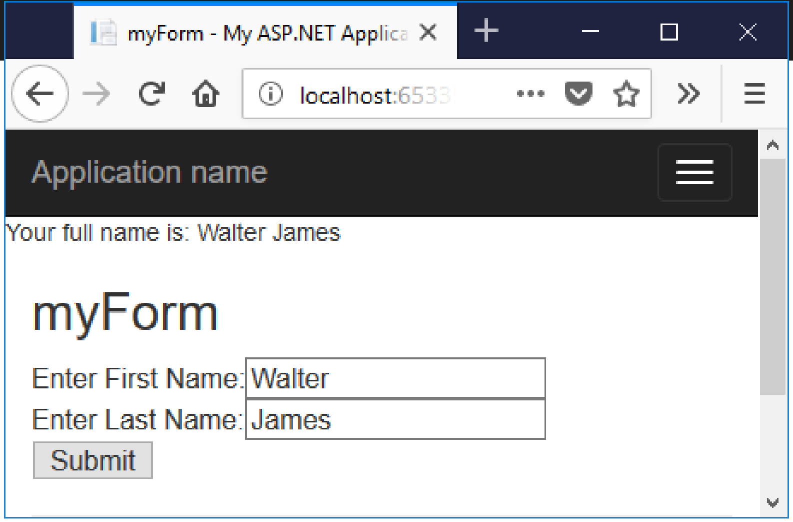
In the traditional approach we use the request object of the HttpRequestBase class. The request object has view input field values in name/value pairs. When we create a submit button then the request type POST is created and calls the POST method.



```
<h2>myForm</h2>
@using (Html.BeginForm())
{
    <table>
        <tr>
            <td>@Html.Label("Enter First Name: ")</td>
            <td>@Html.TextBox("fname")</td>
        </tr>
        <tr>
            <td>@Html.Label("Enter Last Name: ")</td>
            <td>@Html.TextBox("lname")</td>
        </tr>
        <tr>
            <td><input type="submit" value="Submit" /></td>
        </tr>
    </table>
}
```



```
public ActionResult myForm()
{
    string fname = Request["fname"].ToString();
    string lname = Request["lname"].ToString();
    Response.Write("Your full name is: " + fname + " " + lname);
    return View();
}
```



Your full name is: Walter James

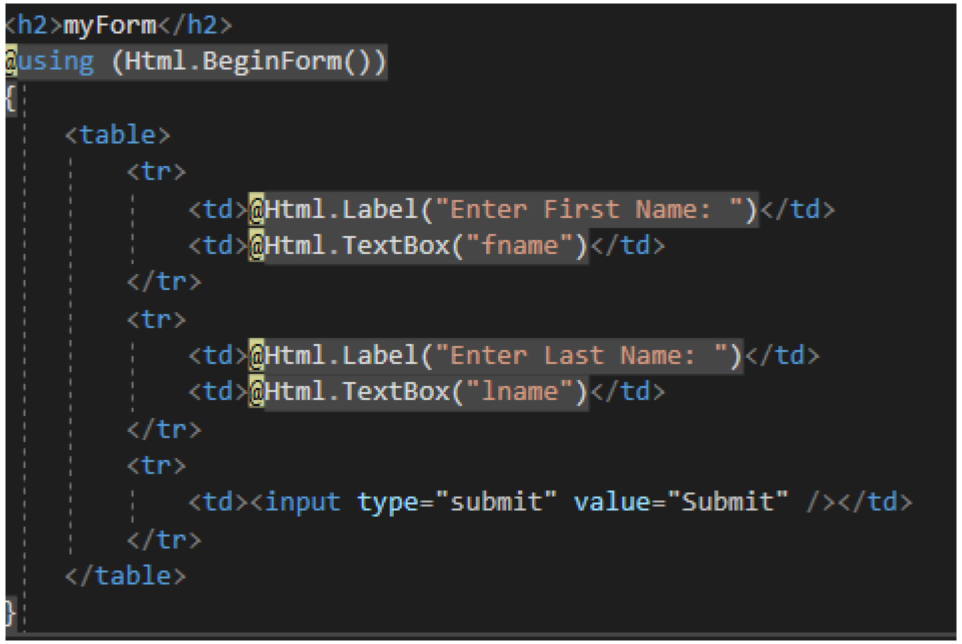
myForm

Enter First Name: Walter
Enter Last Name: James

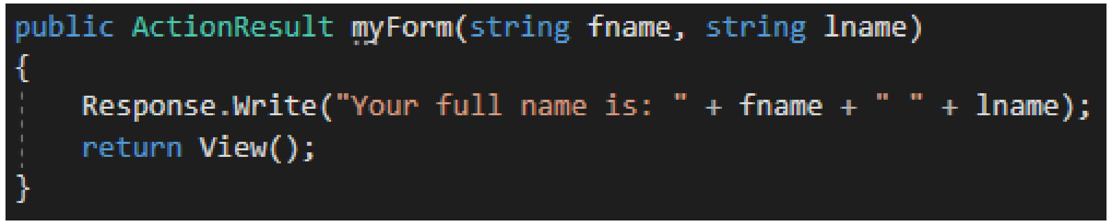
Passing Data from View to Controller

Using the Parameters

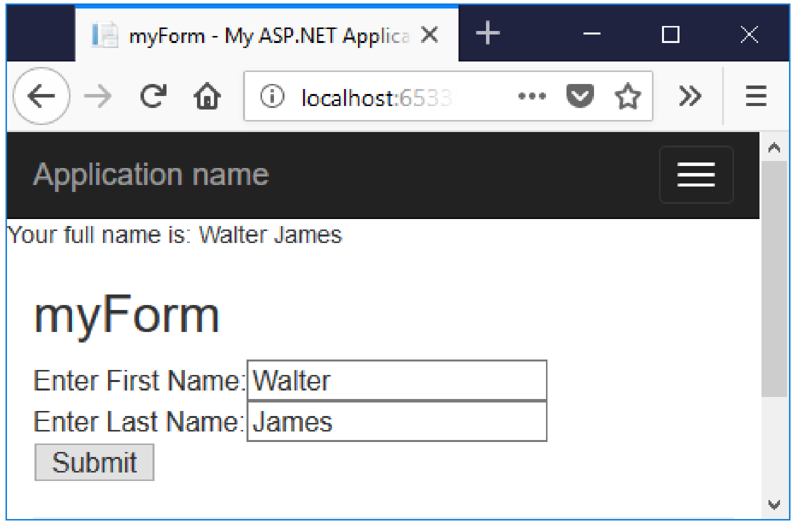
We can pass all input field names as a parameter to the post action method. The input field name and parameter name should be the same. These parameters have input field values that were entered by the user. So we can access view input field values from these parameters. The input field takes a string value from the user so the parameter should be a string type.



```
<h2>myForm</h2>
@using (Html.BeginForm())
{
    <table>
        <tr>
            <td>@Html.Label("Enter First Name: ")</td>
            <td>@Html.TextBox("fname")</td>
        </tr>
        <tr>
            <td>@Html.Label("Enter Last Name: ")</td>
            <td>@Html.TextBox("lname")</td>
        </tr>
        <tr>
            <td><input type="submit" value="Submit" /></td>
        </tr>
    </table>
}
```



```
public ActionResult myForm(string fname, string lname)
{
    Response.Write("Your full name is: " + fname + " " + lname);
    return View();
}
```



Your full name is: Walter James

myForm

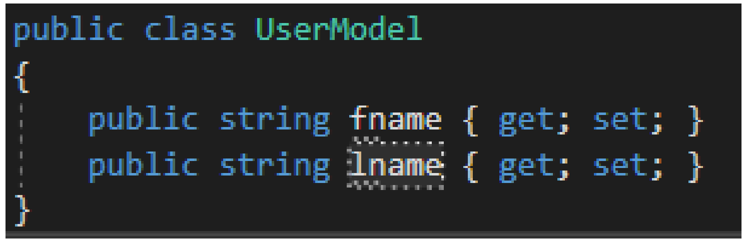
Enter First Name: Walter
Enter Last Name: James
Submit

Passing Data from View to Controller

Strongly typed view

We bind a model to the view; that is called strongly type model binding.

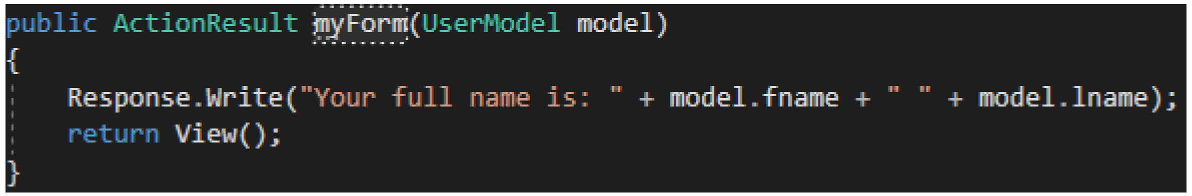
Step 1
Create a Model.



```
public class UserModel
{
    public string fname { get; set; }
    public string lname { get; set; }
}
```

Step 2

Create an action method that render a view on the UI.

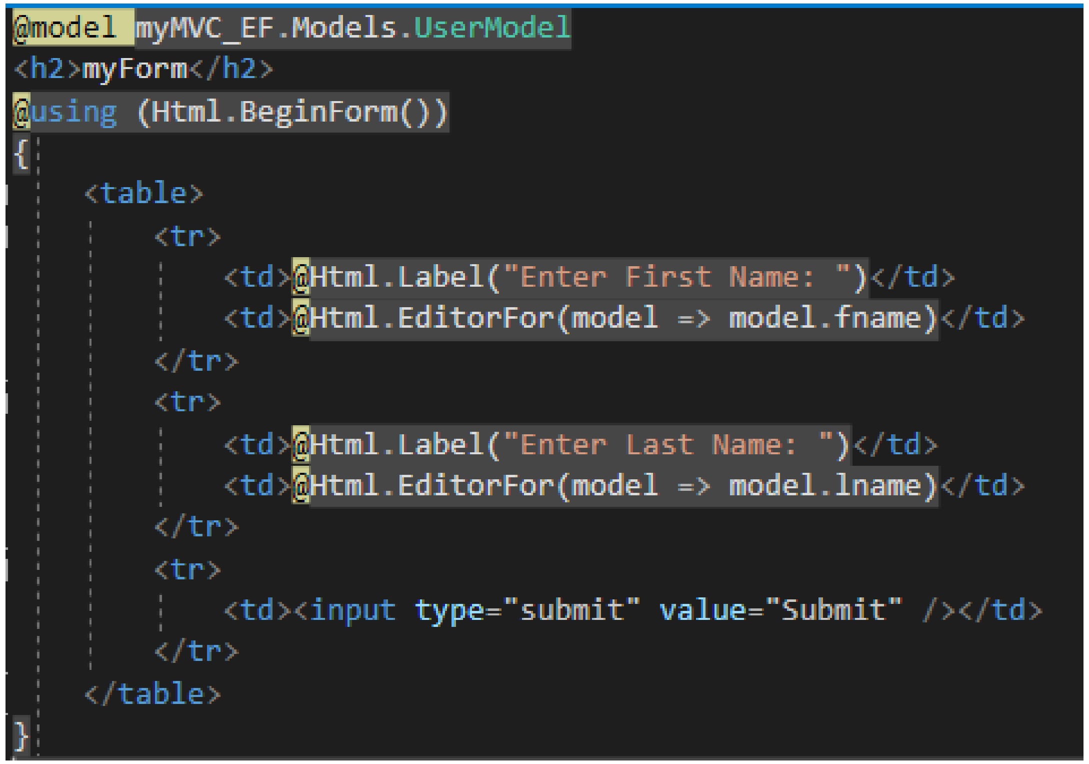


```
public ActionResult myForm(UserModel model)
{
    Response.Write("Your full name is: " + model.fname + " " + model.lname);
    return View();
}
```

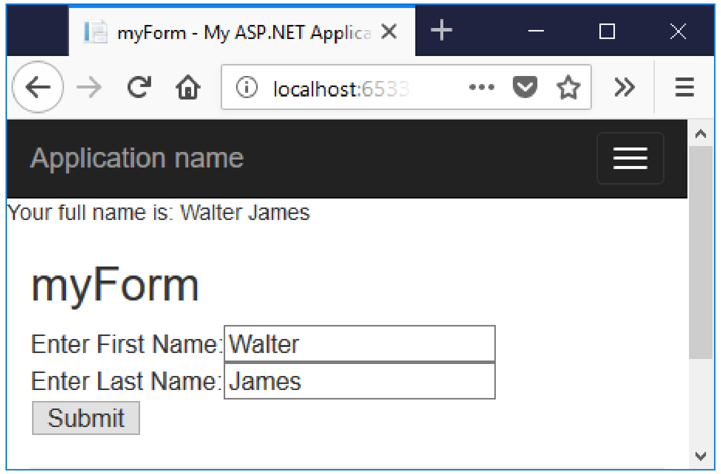
Passing Data from View to Controller

Step 3

Create a strongly typed view.



```
@model myMVC_EF.Models.UserModel
<h2>myForm</h2>
@using (Html.BeginForm())
{
    <table>
        <tr>
            <td>@Html.Label("Enter First Name: ")</td>
            <td>@Html.EditorFor(model => model.fname)</td>
        </tr>
        <tr>
            <td>@Html.Label("Enter Last Name: ")</td>
            <td>@Html.EditorFor(model => model.lname)</td>
        </tr>
        <tr>
            <td><input type="submit" value="Submit" /></td>
        </tr>
    </table>
}
```



myForm - My ASP.NET Applica

localhost:6533

Application name

Your full name is: Walter James

myForm

Enter First Name: Walter

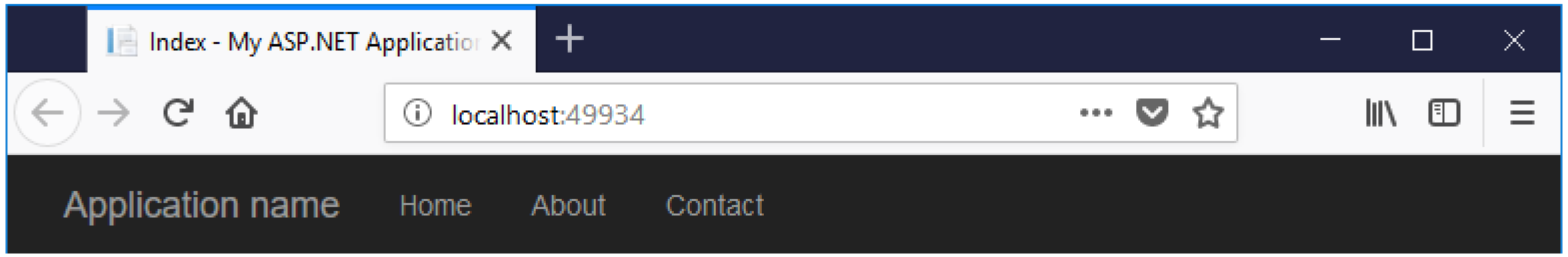
Enter Last Name: James

Submit

Partial Views

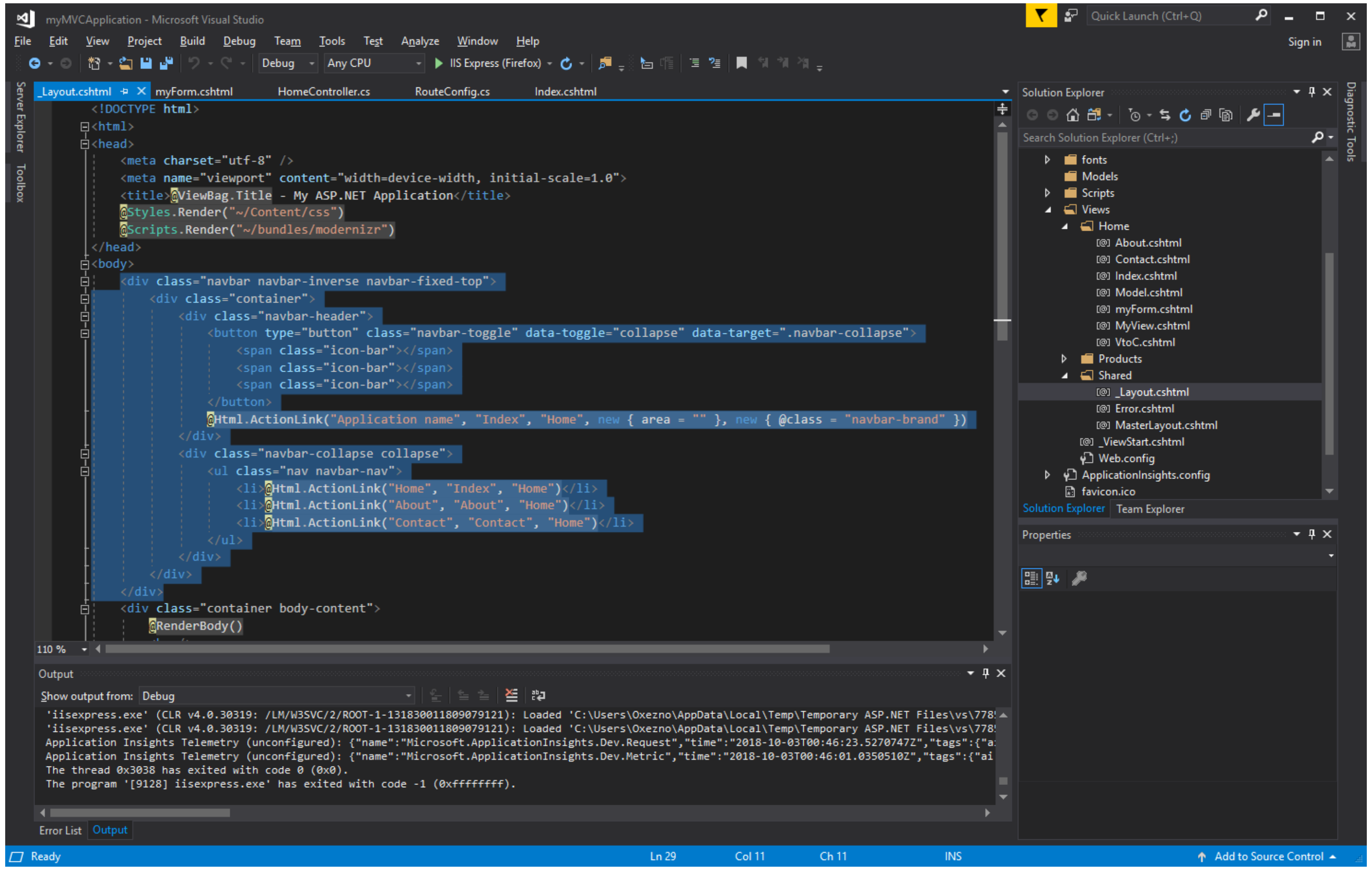
Partial view is a reusable view, which can be used as a child view in multiple other views. It eliminates duplicate coding by reusing same partial view in multiple places. You can use the partial view in the layout view, as well as other content views.

To start with, let's create a simple partial view for the following navigation bar. We will create a partial view for it, so that we can use the same navigation bar in multiple layout views without rewriting the same code everywhere.



Partial Views

The following figure shows the html code for the above navigation bar. We will cut and paste this code in a seperate partial view.



```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - My ASP.NET Application</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
</head>
<body>
    <div class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                @Html.ActionLink("Application name", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li>@Html.ActionLink("Home", "Index", "Home")</li>
                    <li>@Html.ActionLink("About", "About", "Home")</li>
                    <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
                </ul>
            </div>
        </div>
    </div>
    <div class="container body-content">
        @RenderBody()
```

Show output from: Debug

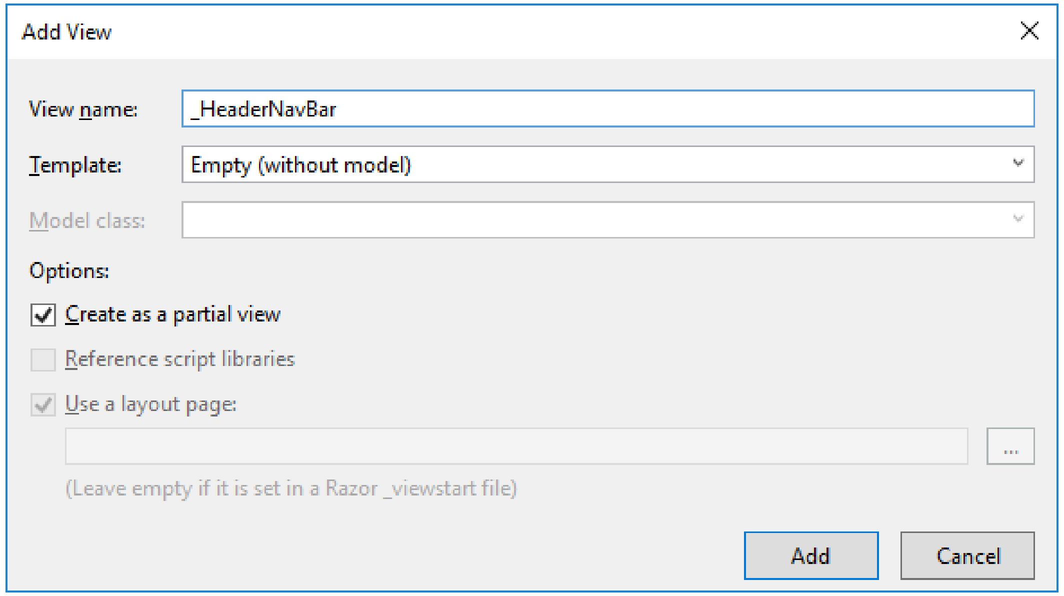
```
'iisexpress.exe' (CLR v4.0.30319: /LM/W3SVC/2/ROOT-1-131830011809079121): Loaded 'C:\Users\Oxezno\AppData\Local\Temp\Temporary ASP.NET Files\vs\778
'iisexpress.exe' (CLR v4.0.30319: /LM/W3SVC/2/ROOT-1-131830011809079121): Loaded 'C:\Users\Oxezno\AppData\Local\Temp\Temporary ASP.NET Files\vs\778
Application Insights Telemetry (unconfigured): {"name":"Microsoft.ApplicationInsights.Dev.Request","time":"2018-10-03T00:46:23.5270747Z","tags":{"a
Application Insights Telemetry (unconfigured): {"name":"Microsoft.ApplicationInsights.Dev.Metric","time":"2018-10-03T00:46:01.0350510Z","tags":{"ai
The thread 0x3038 has exited with code 0 (0x0).
The program '[9128] iisexpress.exe' has exited with code -1 (0xffffffff).
```

Partial Views

How to create a Partial View

To create a partial view, right click on Shared folder -> select **Add** -> click on **View..**

In the Add View dialogue, enter View name and select "Create as a partial view" checkbox and click Add.



Add View

View name: _HeaderNavBar

Template: Empty (without model)

Model class:

Options:

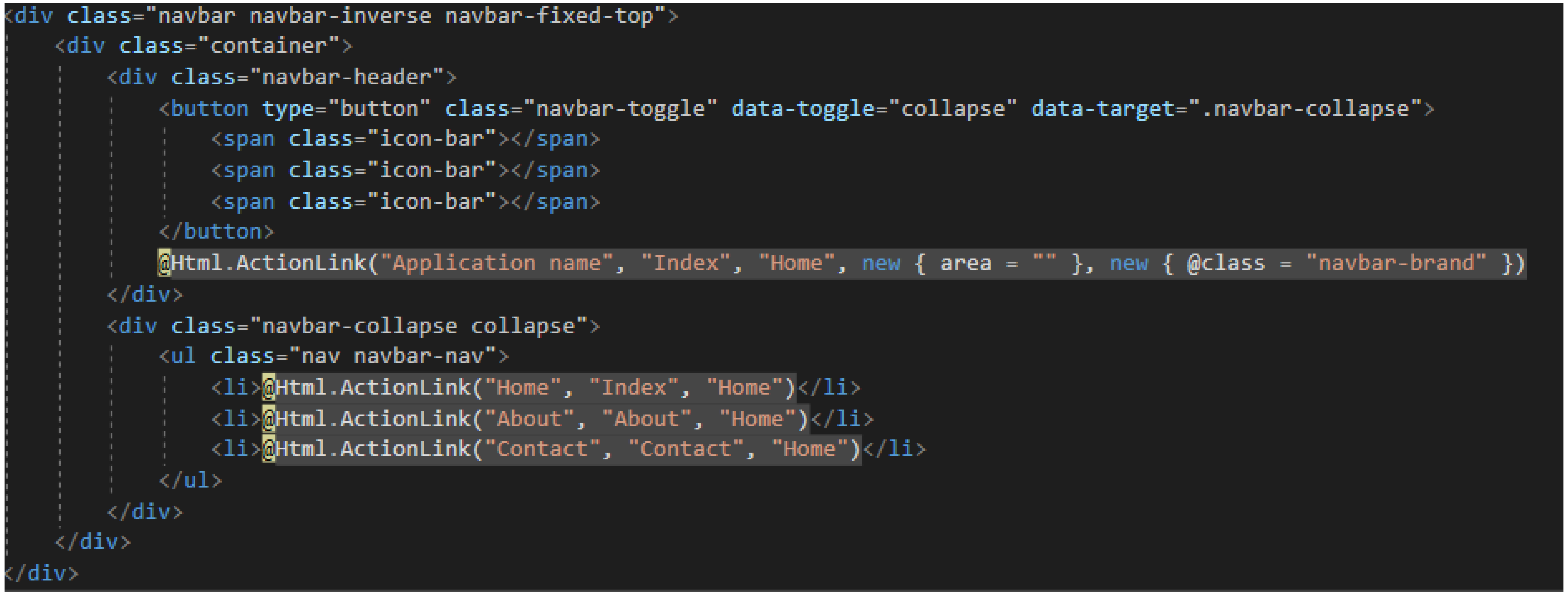
- [✓] Create as a partial view
- [ ] Reference script libraries
- [✓] Use a layout page:

(Leave empty if it is set in a Razor _viewstart file)

We are not going to use any model for this partial view, so keep the Template dropdown as Empty (without model) and click **Add**. This will create an empty partial view in Shared folder.

Partial Views

Now, you can cut the above code for navigation bar and paste it in _HeaderNavBar.cshtml as shown below:

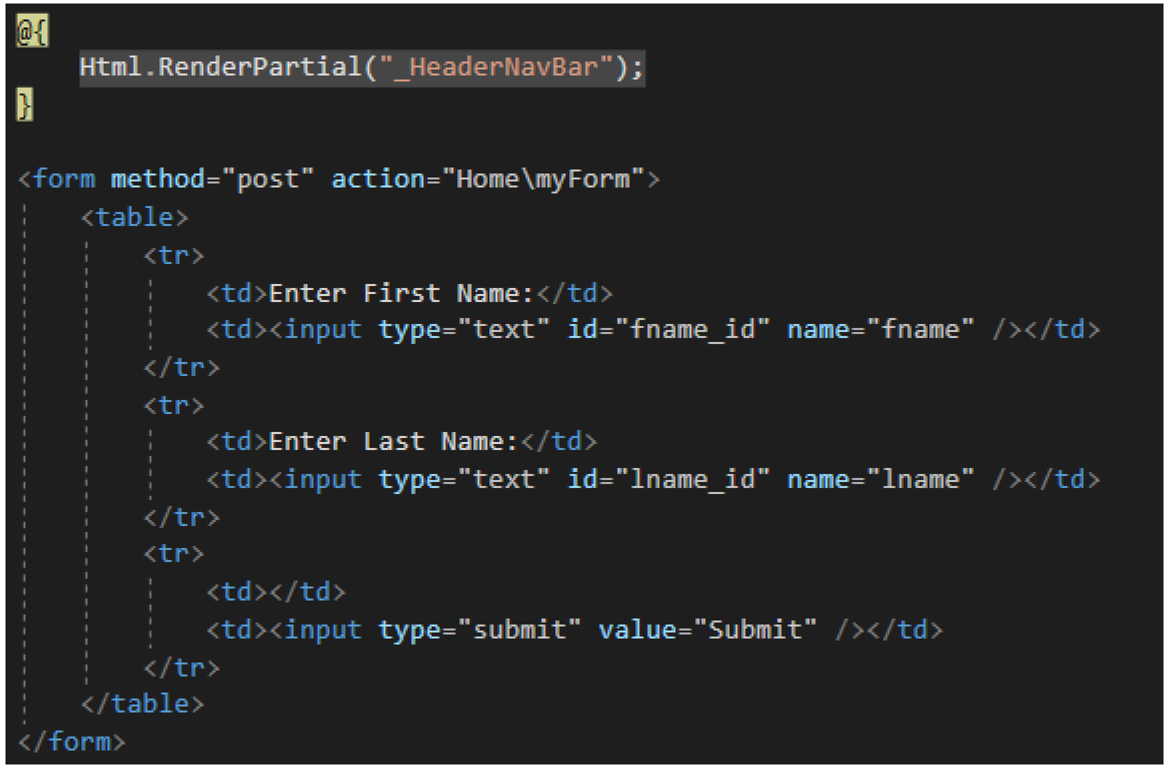


```
<div class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            @Html.ActionLink("Application name", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })
        </div>
        <div class="navbar-collapse collapse">
            <ul class="nav navbar-nav">
                <li>@Html.ActionLink("Home", "Index", "Home")</li>
                <li>@Html.ActionLink("About", "About", "Home")</li>
                <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
            </ul>
        </div>
    </div>
</div>
```

Partial Views

How to consume a Partial View

The following layout view renders partial view using the RenderPartial() method.

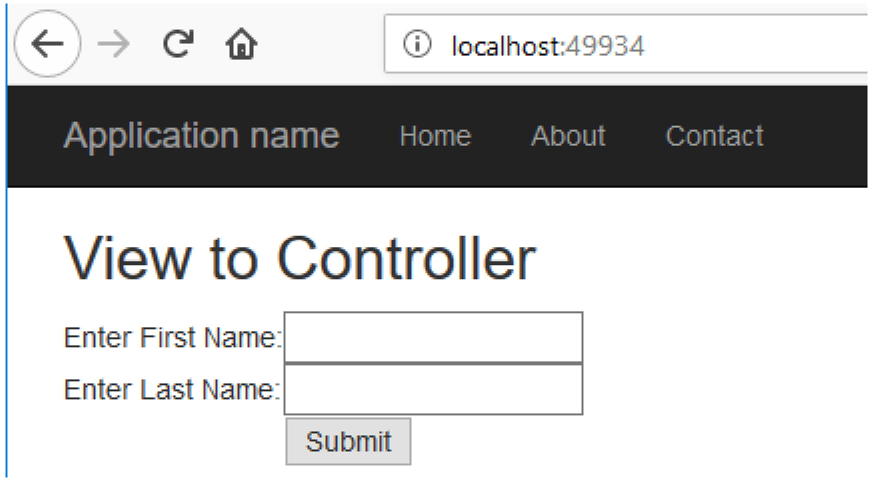


```
@{
    Html.RenderPartial("_HeaderNavBar");
}

<form method="post" action="Home\myForm">
    <table>
        <tr>
            <td>Enter First Name:</td>
            <td><input type="text" id="fname_id" name="fname" /></td>
        </tr>
        <tr>
            <td>Enter Last Name:</td>
            <td><input type="text" id="lname_id" name="lname" /></td>
        </tr>
        <tr>
            <td></td>
            <td><input type="submit" value="Submit" /></td>
        </tr>
    </table>
</form>
```

Partial Views

View with Partial View

View without Partial View



localhost:49934

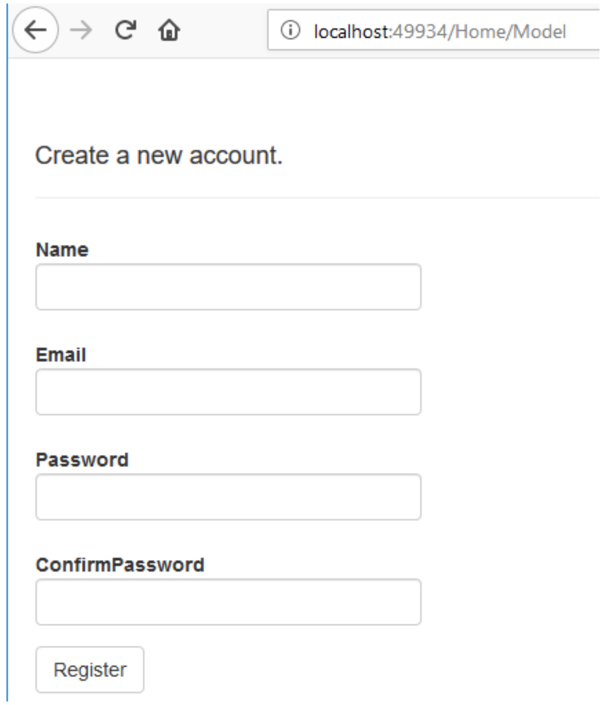
Application name Home About Contact

View to Controller

Enter First Name:

Enter Last Name:

Submit



localhost:49934/Home/Model

Create a new account.

Name

Email

Password

ConfirmPassword

Register

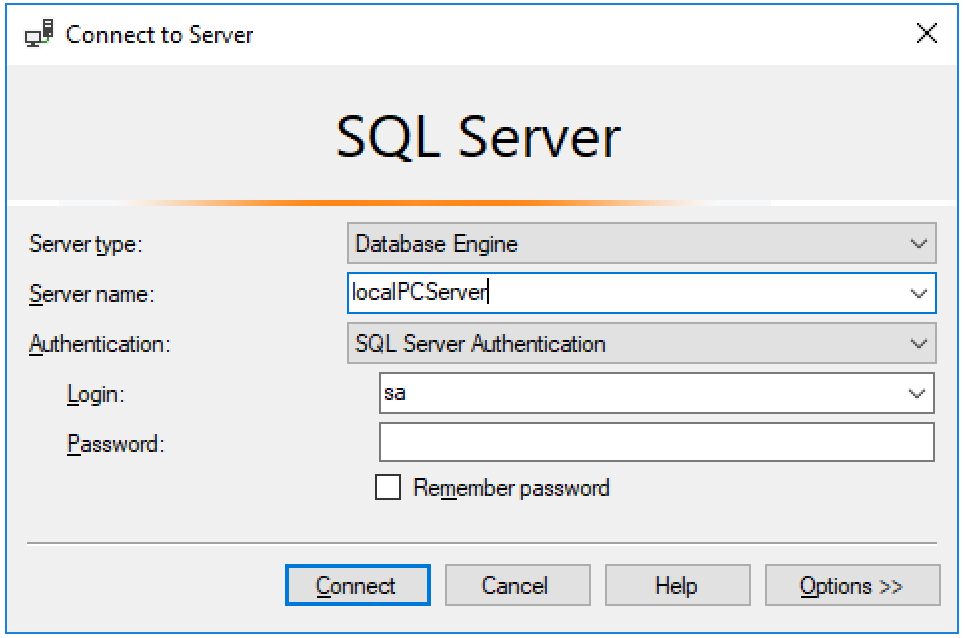
Working with Models

Create a MVC Model

The component ‘Model’ is responsible for managing the data of the application. It responds to the request from the view and it also responds to instructions from the controller to update itself.

Create Database Entities

Connect to MSSQL Server and create a new database.


Connect to Server

SQL Server

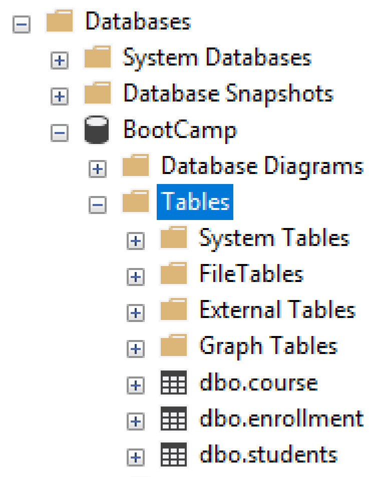
Server type: Database Engine
Server name: localPCServer
Authentication: SQL Server Authentication
Login: sa
Password:
Remember password

Create a MVC Model

Now run the following queries to create new tables.

```
CREATE TABLE [dbo].[students] (
[StudentID] int NOT NULL ,
[FirstName] varchar(45) NULL ,
[LastName] varchar(45) NULL ,
[EnrollmentDate] datetime NULL
)
ALTER TABLE [dbo].[students] ADD PRIMARY KEY ([StudentID])


CREATE TABLE [dbo].[course] (
[CourseID] int NOT NULL ,
[Title] varchar(50) NULL ,
[Credits] int NULL
)
ALTER TABLE [dbo].[course] ADD PRIMARY KEY ([CourseID])
```

```
CREATE TABLE [dbo].[enrollment] (
[EnrollmentID] int NOT NULL ,
[Grade] decimal(18) NULL ,
[CourseID] int NULL ,
[StudentID] int NULL
)
ALTER TABLE [dbo].[enrollment] ADD PRIMARY KEY ([EnrollmentID])
ALTER TABLE [dbo].[enrollment] ADD FOREIGN KEY ([CourseID])
REFERENCES [dbo].[course] ([CourseID]) ON DELETE NO ACTION ON
UPDATE NO ACTION
ALTER TABLE [dbo].[enrollment] ADD FOREIGN KEY ([StudentID])
REFERENCES [dbo].[students] ([StudentID]) ON DELETE NO ACTION ON
UPDATE NO ACTION
```



```
Databases
  System Databases
  Database Snapshots
  BootCamp
    Database Diagrams
    Tables
      System Tables
      FileTables
      External Tables
      Graph Tables
      dbo.course
      dbo.enrollment
      dbo.students
```

Insert, Update, Delete without Entity Framework

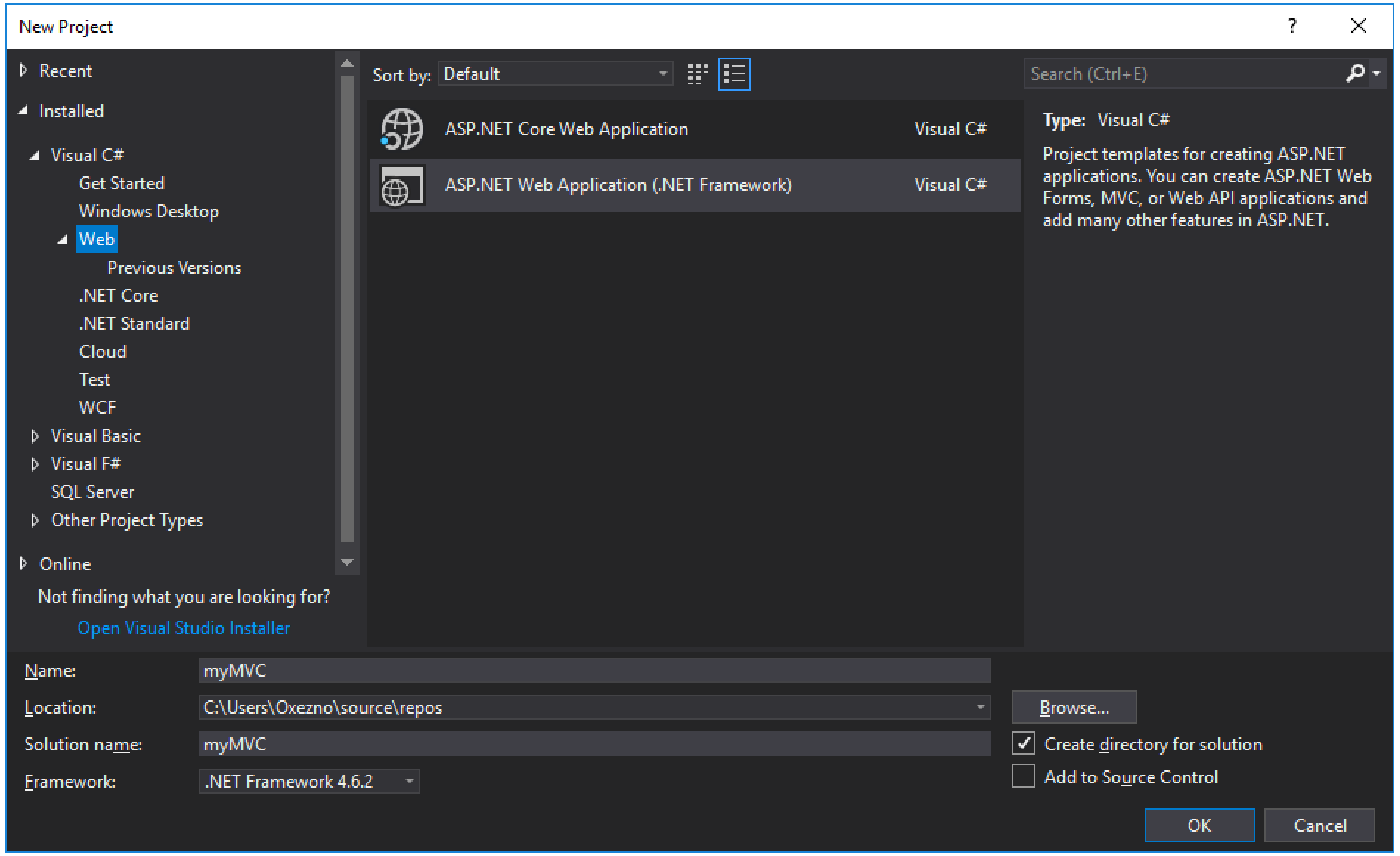
Insert, Update, Delete without EF

Create a new MVC Application

Start your Visual Studio and select:

File → New → Project. Select Web → ASP.NET Web Application (.NET Framework)

Name this project as **myMVC_NoEF**



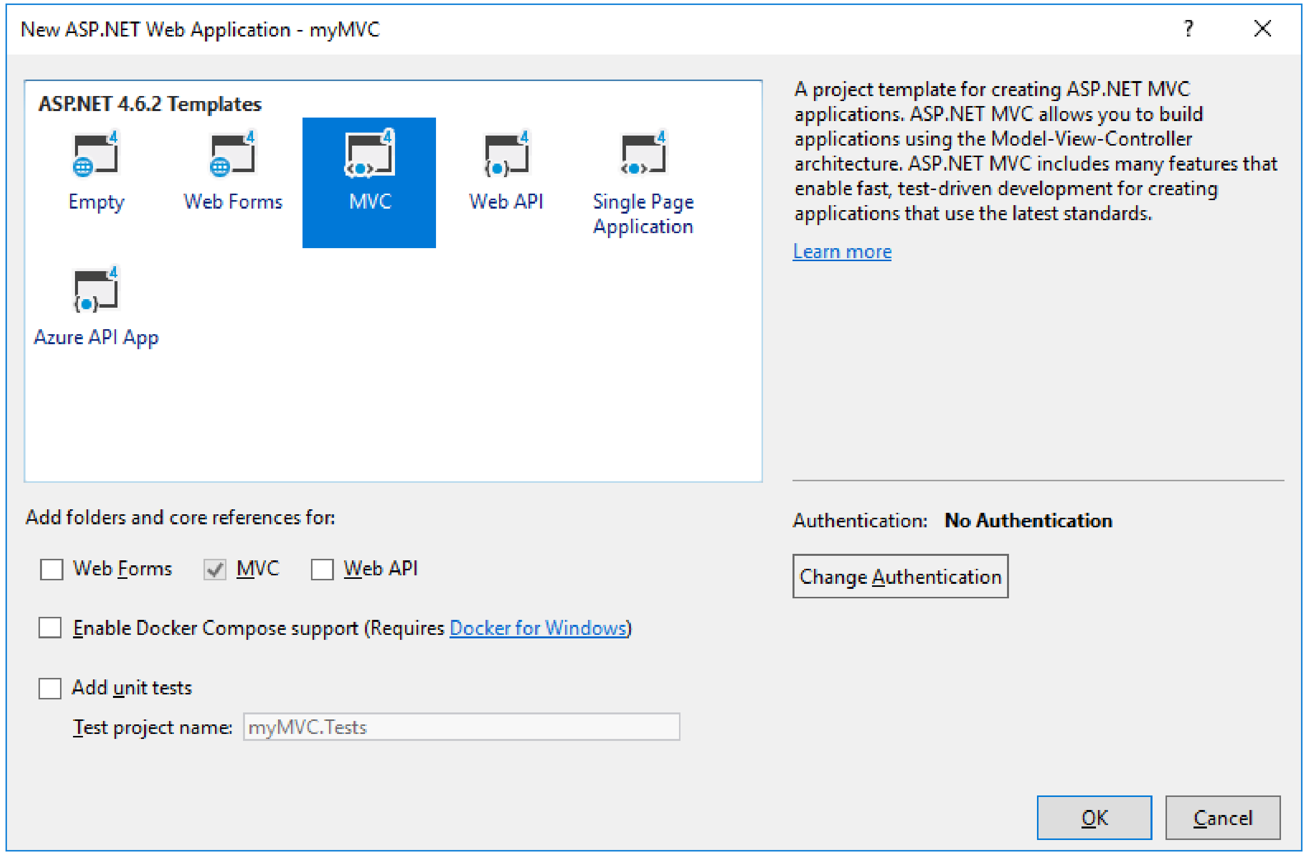
Type: Visual C#

Project templates for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Name: myMVC
Location: C:\Users\Oxezno\source\repos
Solution name: myMVC
Framework: .NET Framework 4.6.2
[✓] Create directory for solution
[ ] Add to Source Control

Insert, Update, Delete without EF

Select MVC Template. Click OK.



New ASP.NET Web Application - myMVC

ASP.NET 4.6.2 Templates

A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

Learn more

Add folders and core references for:

- [ ] Web Forms
- [✓] MVC
- [ ] Web API

- [ ] Enable Docker Compose support (Requires Docker for Windows)

- [ ] Add unit tests

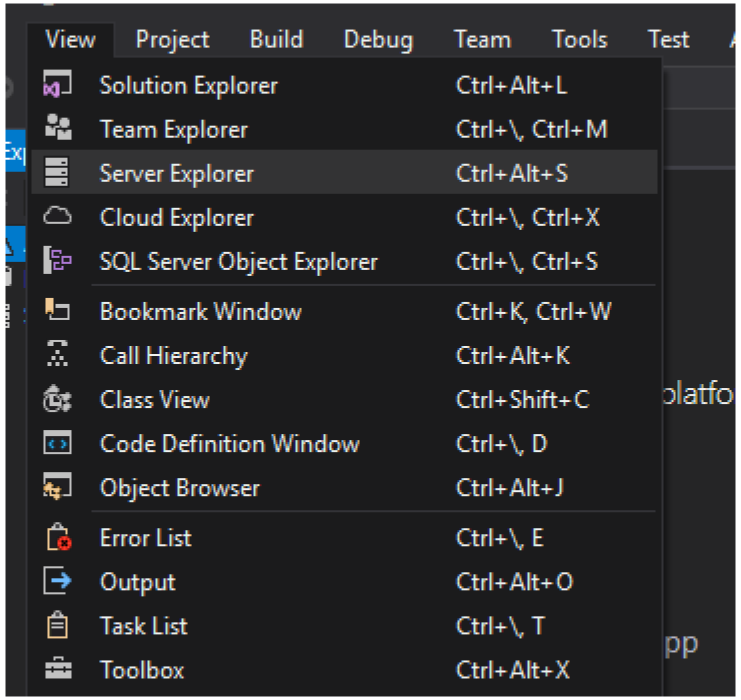
Test project name: myMVC.Tests

Authentication: **No Authentication**

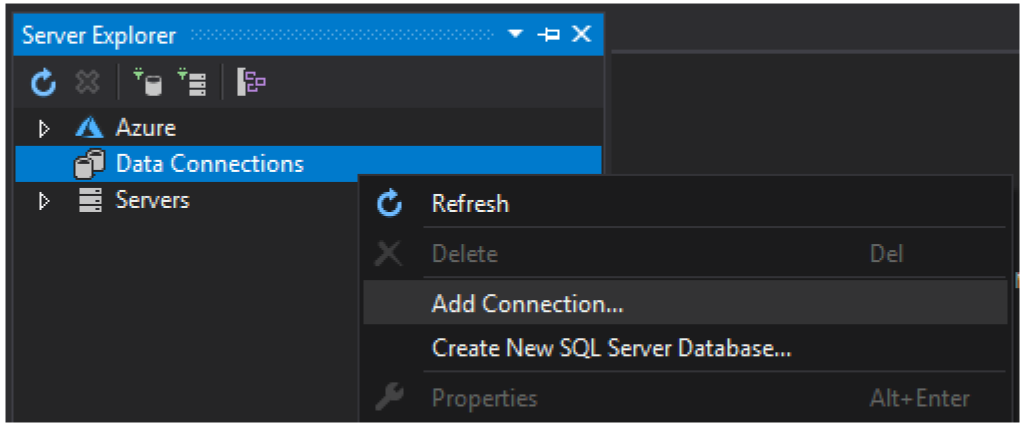
Insert, Update, Delete without EF

Open Server Explorer in Visual Studio.
Go to *View → Server Explorer.*

Right Click on
Data Connections → Add Connection

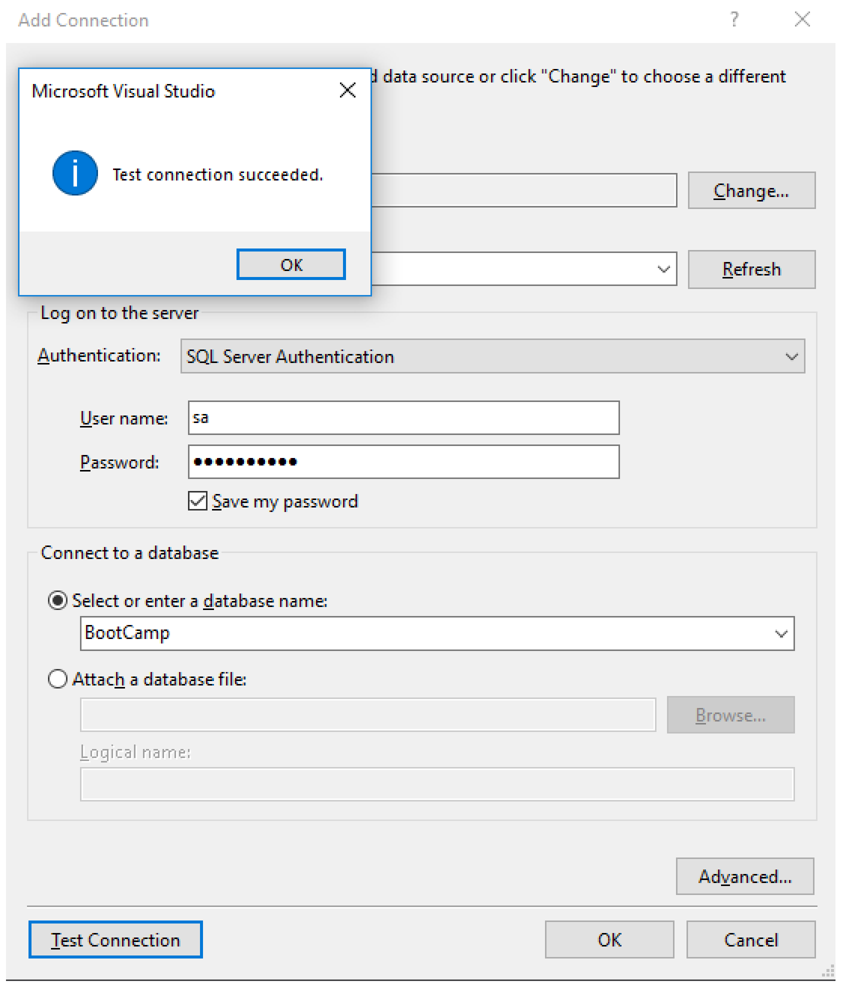


View Project Build Debug Team Tools Test
Solution Explorer Ctrl+Alt+L
Team Explorer Ctrl+\, Ctrl+M
Server Explorer Ctrl+Alt+S
Cloud Explorer Ctrl+\, Ctrl+X
SQL Server Object Explorer Ctrl+\, Ctrl+S
Bookmark Window Ctrl+K, Ctrl+W
Call Hierarchy Ctrl+Alt+K
Class View Ctrl+Shift+C
Code Definition Window Ctrl+\, D
Object Browser Ctrl+Alt+J
Error List Ctrl+\, E
Output Ctrl+Alt+O
Task List Ctrl+\, T
Toolbox Ctrl+Alt+X



Insert, Update, Delete without EF

Fill Server name, User name, Password and select your Database. Test the connection and click OK.


Add Connection

Microsoft Visual Studio

Test connection succeeded.

OK

d data source or click "Change" to choose a different

Change...

Refresh

Log on to the server

Authentication: SQL Server Authentication

User name: sa

Password: ●●●●●●●●●●

[✓] Save my password

Connect to a database

(●) Select or enter a database name:

BootCamp

( ) Attach a database file:

Browse...

Logical name:

Advanced...

Test Connection

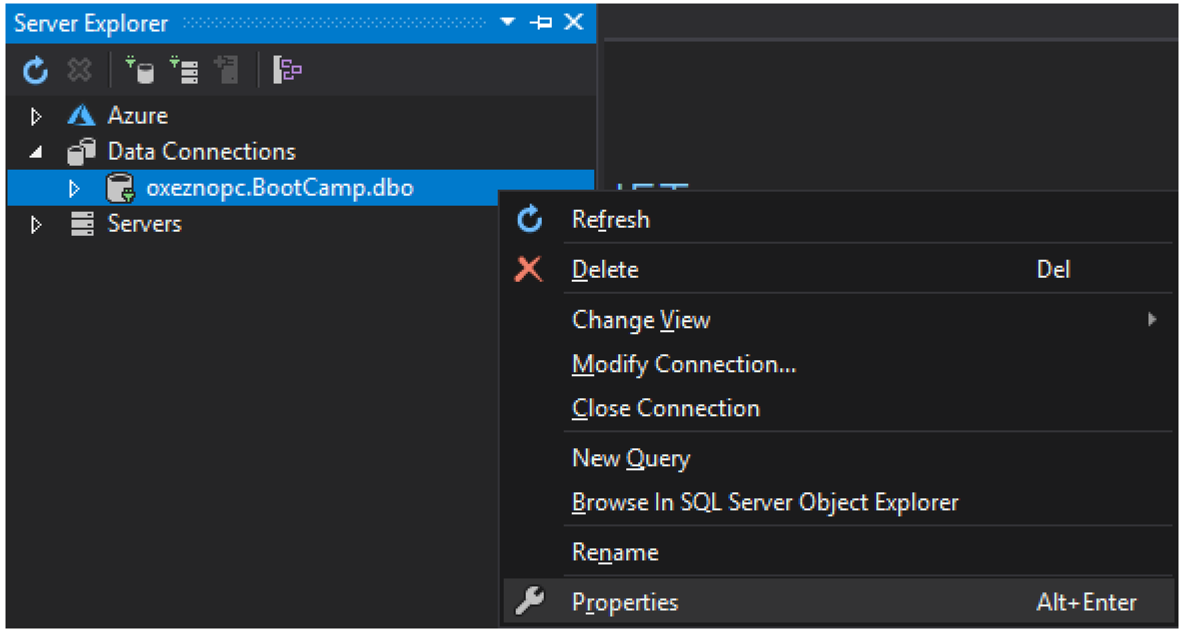
OK

Cancel

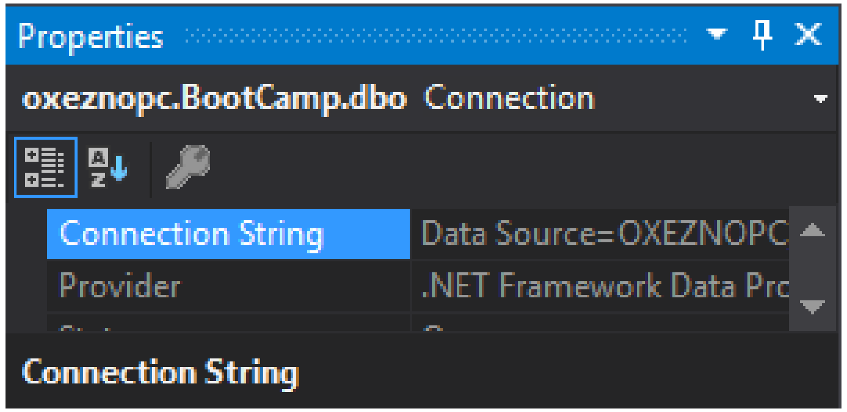
Insert, Update, Delete without EF

Add connection string in “Web.config” file.

Right Click on your Database Connection in Server Explorer and Select Properties.



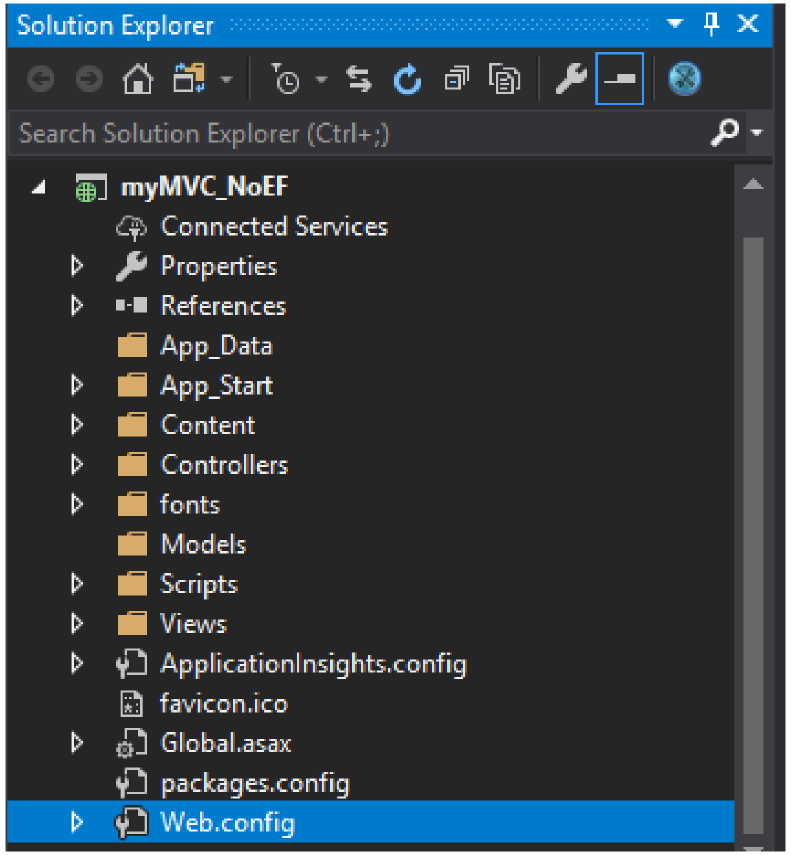
You will find Connection String in Properties window. Copy connection string and add it in **web.config** file.

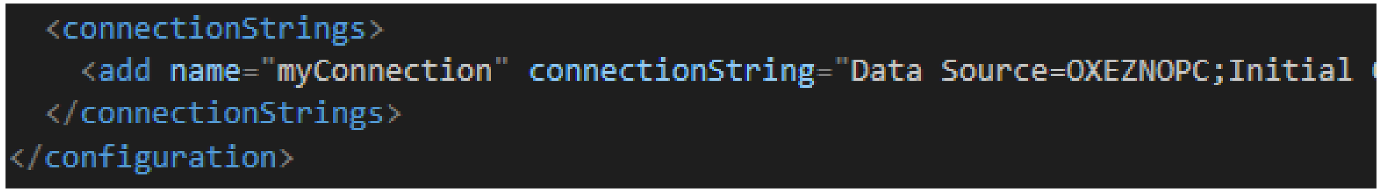


Connection String	Data Source=OXEZNOPC
Provider	.NET Framework Data Pro

Insert, Update, Delete without EF

Open Root Web.config file and add connection string before </Configuration> as follow.





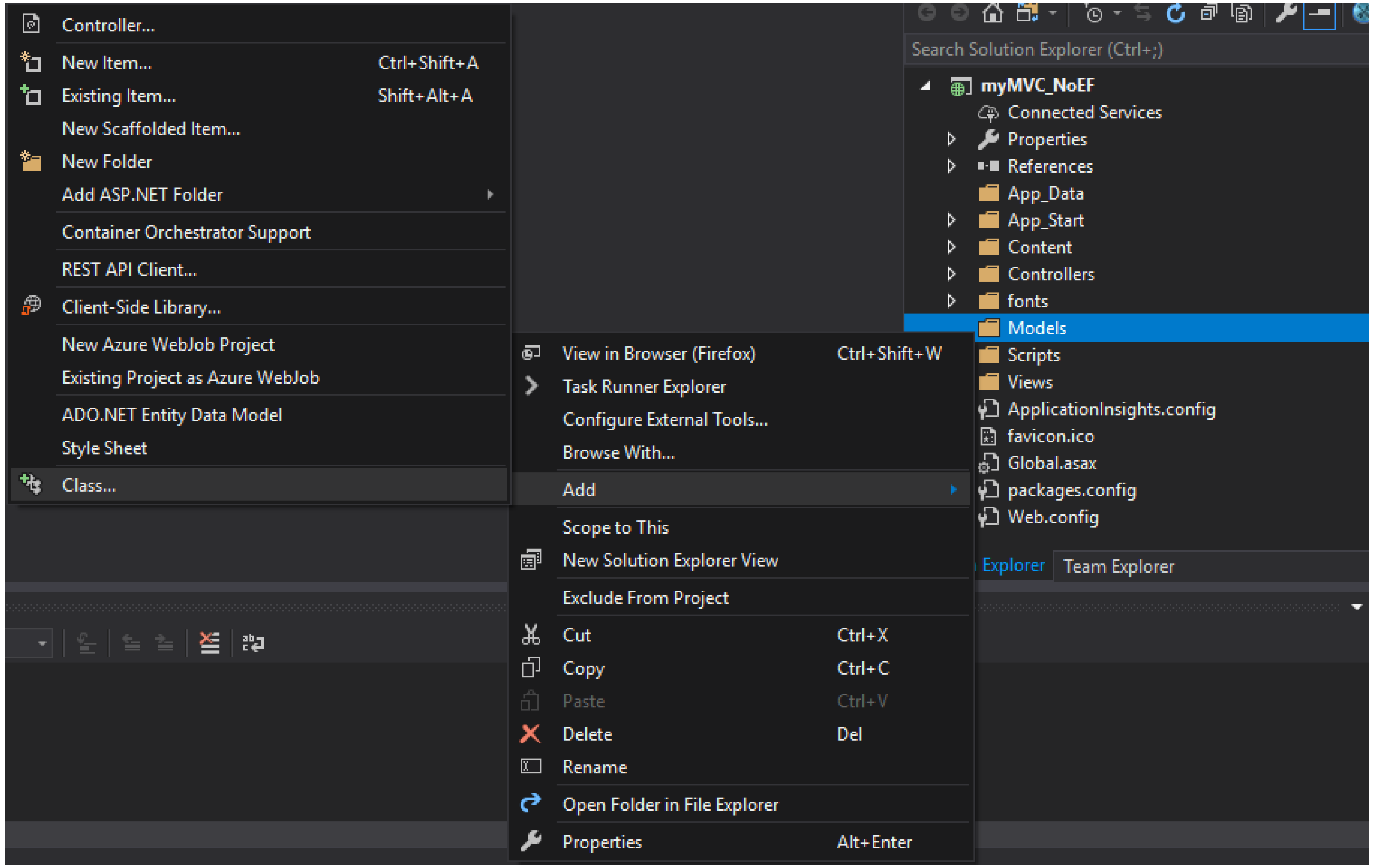
```
  <connectionStrings>
    <add name="myConnection" connectionString="Data Source=OXEZNOPC;Initial
  </connectionStrings>
</configuration>
```

Code:

```
<connectionStrings>
    <add name="myConnection" connectionString=""/>
</connectionStrings>
```

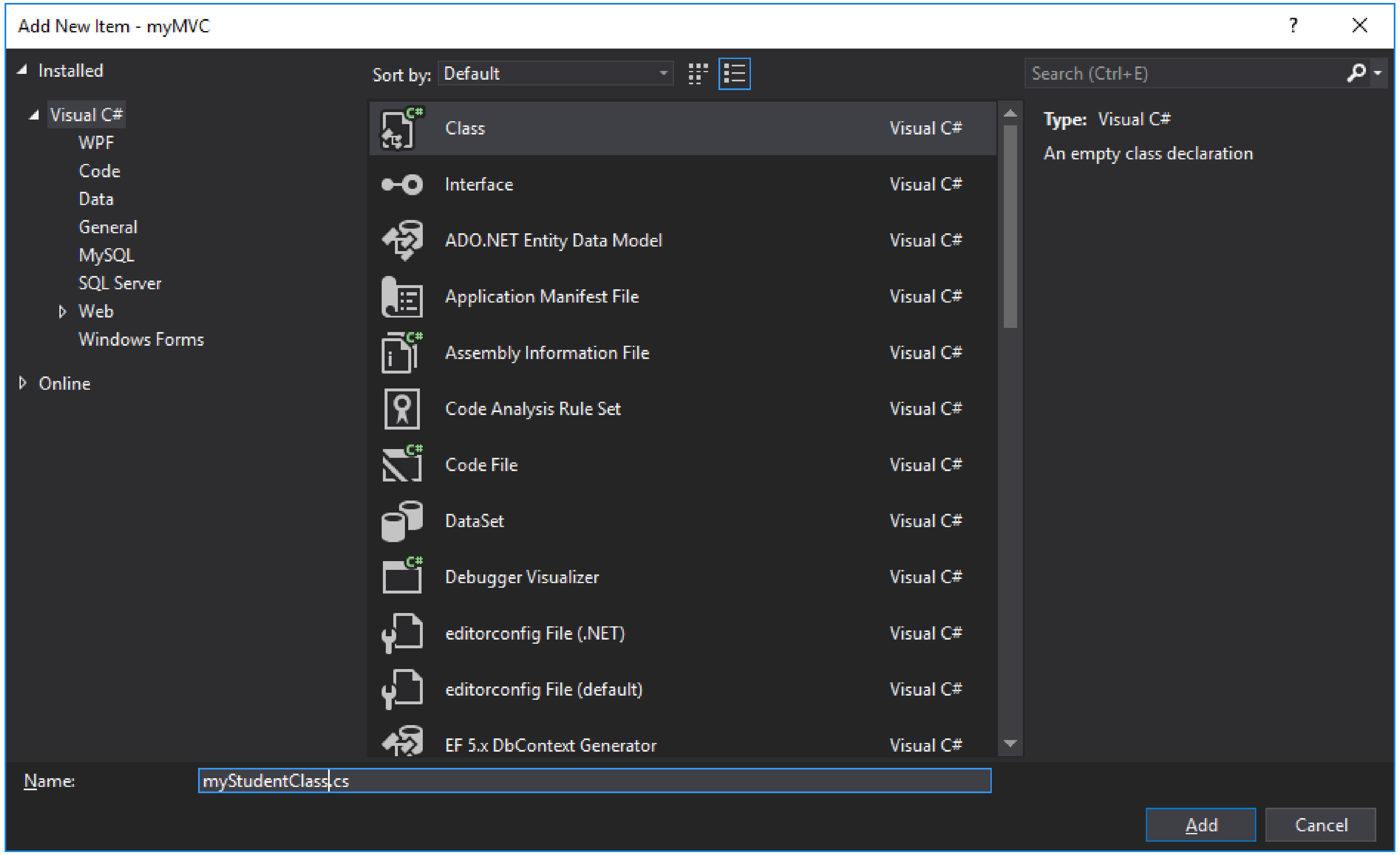
Insert, Update, Delete without EF

Create a new class in **Model**.
Right click on ***Model → Add → Class***



Insert, Update, Delete without EF

Name this class as myStudentClass and click “Add”



Add New Item - myMVC

Sort by: Default

Search (Ctrl+E)

- Installed
  - Visual C#
    - WPF
    - Code
    - Data
    - General
    - MySQL
    - SQL Server
    - Web
    - Windows Forms
- Online

Class	Visual C#
Interface	Visual C#
ADO.NET Entity Data Model	Visual C#
Application Manifest File	Visual C#
Assembly Information File	Visual C#
Code Analysis Rule Set	Visual C#
Code File	Visual C#
DataSet	Visual C#
Debugger Visualizer	Visual C#
editorconfig File (.NET)	Visual C#
editorconfig File (default)	Visual C#
EF 5.x DbContext Generator	Visual C#

Type: Visual C#

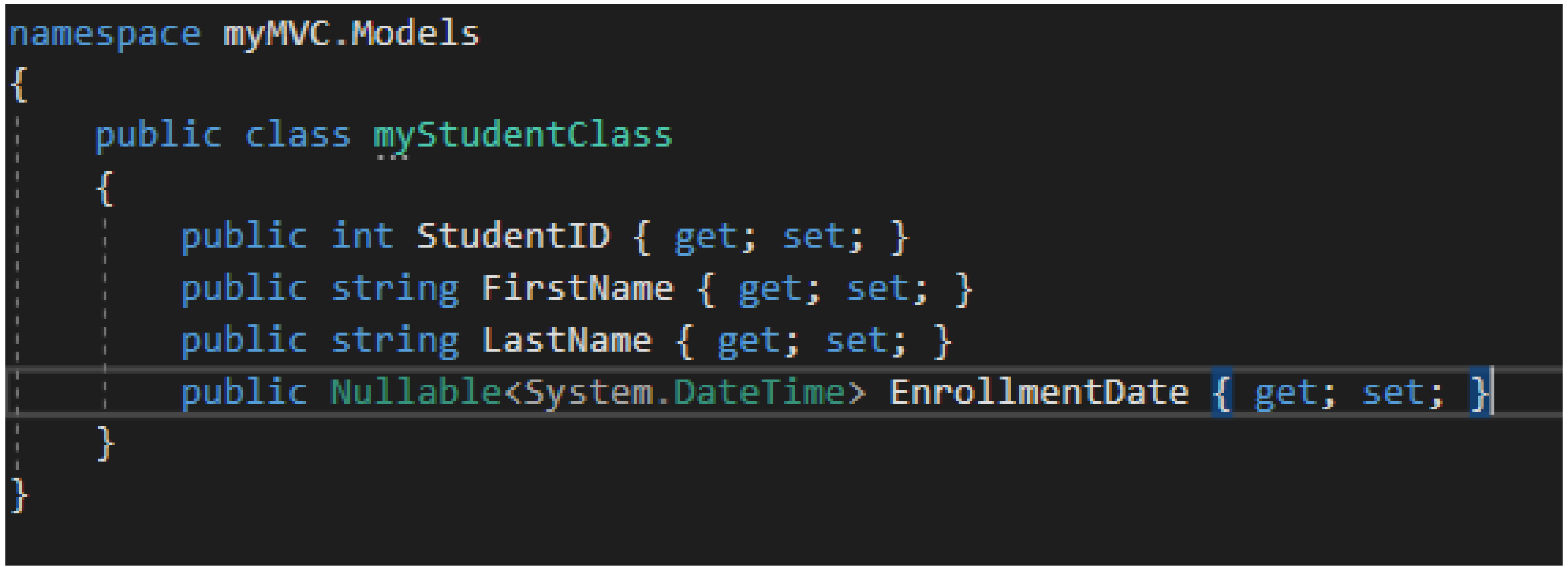
An empty class declaration

Name: myStudentClass.cs

Add Cancel

Insert, Update, Delete without EF

Add the following mapping code in it.



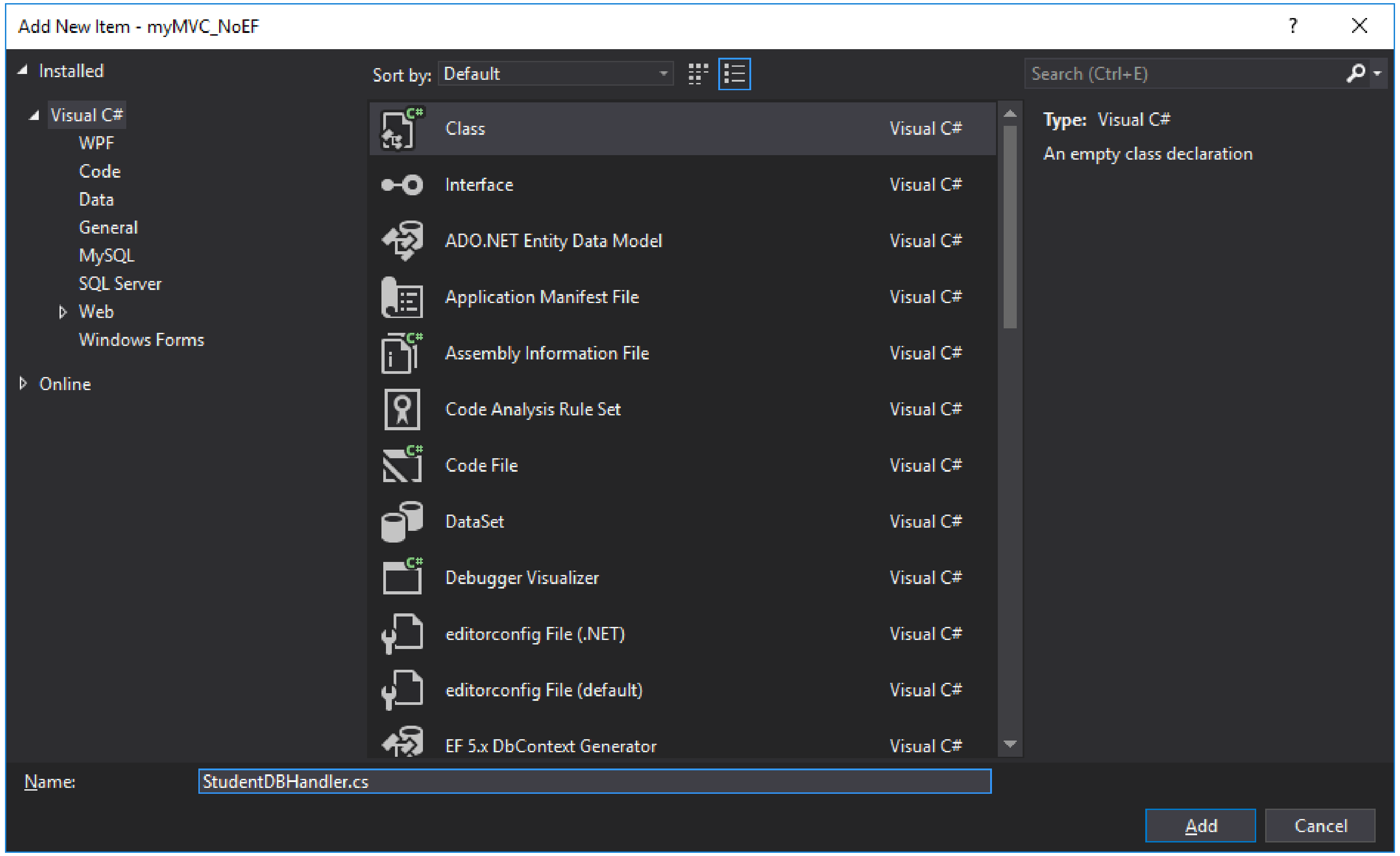
```
namespace myMVC.Models
{
    public class myStudentClass
    {
        public int StudentID { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public Nullable<System.DateTime> EnrollmentDate { get; set; }
    }
}
```

Insert, Update, Delete without EF

Create a "StudentDBHandler.cs" class to handle Database Query.

Create a class that will handle all the database related query as insert, update and delete. Model will use this class to make a connection with database and process record.

Right click on *Model → Add → Class*



Add New Item - myMVC_NoEF

Sort by: Default

Type: Visual C#

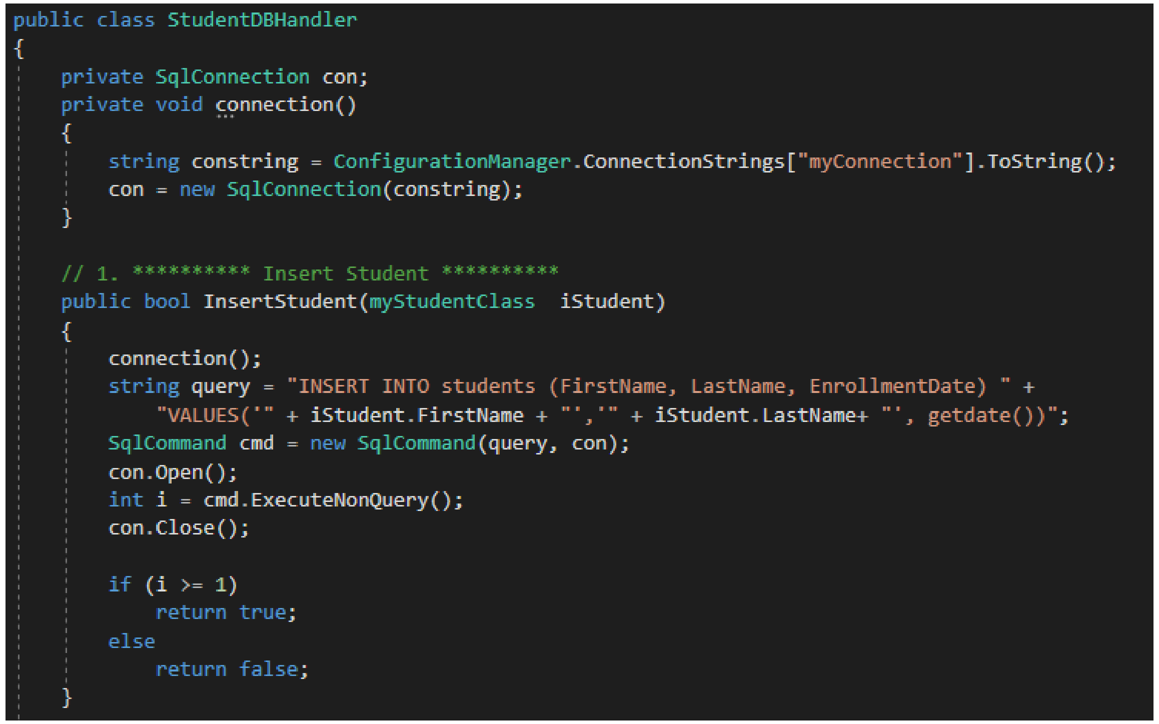
An empty class declaration

Class	Visual C#
Interface	Visual C#
ADO.NET Entity Data Model	Visual C#
Application Manifest File	Visual C#
Assembly Information File	Visual C#
Code Analysis Rule Set	Visual C#
Code File	Visual C#
DataSet	Visual C#
Debugger Visualizer	Visual C#
editorconfig File (.NET)	Visual C#
editorconfig File (default)	Visual C#
EF 5.x DbContext Generator	Visual C#

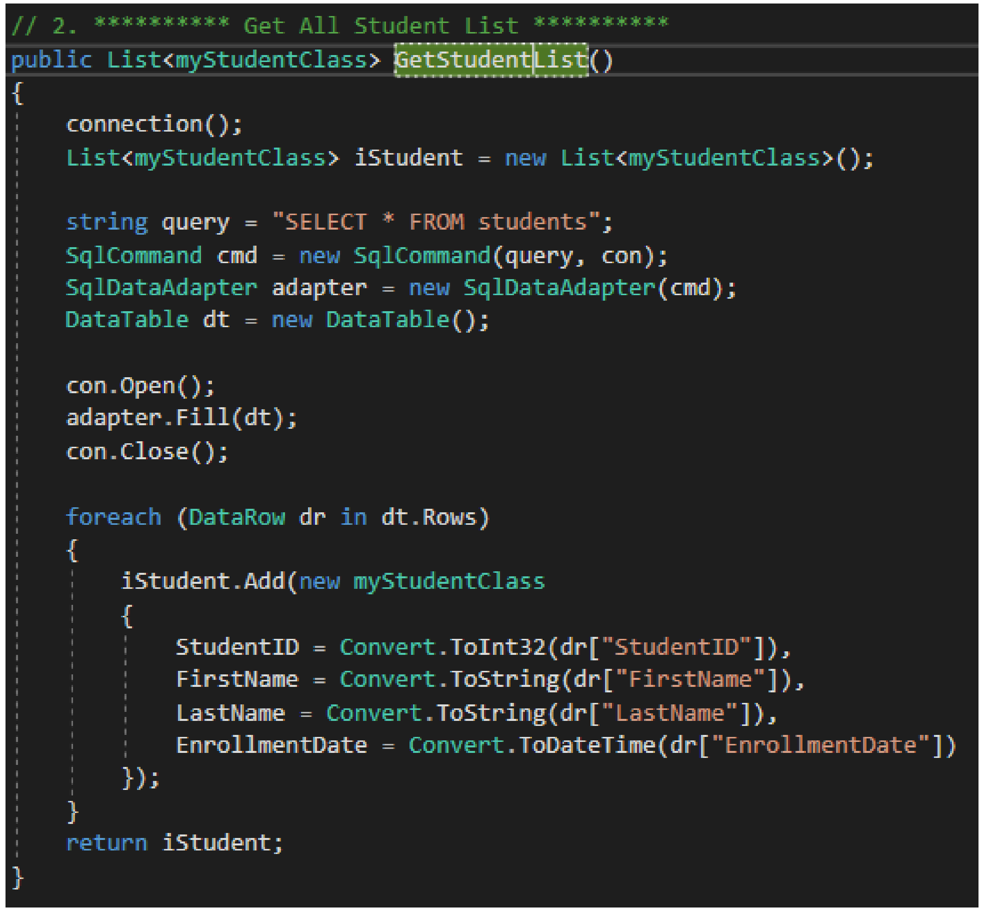
Name: StudentDBHandler.cs

Insert, Update, Delete without EF

Add the following code in StudentDBHandler.cs class.



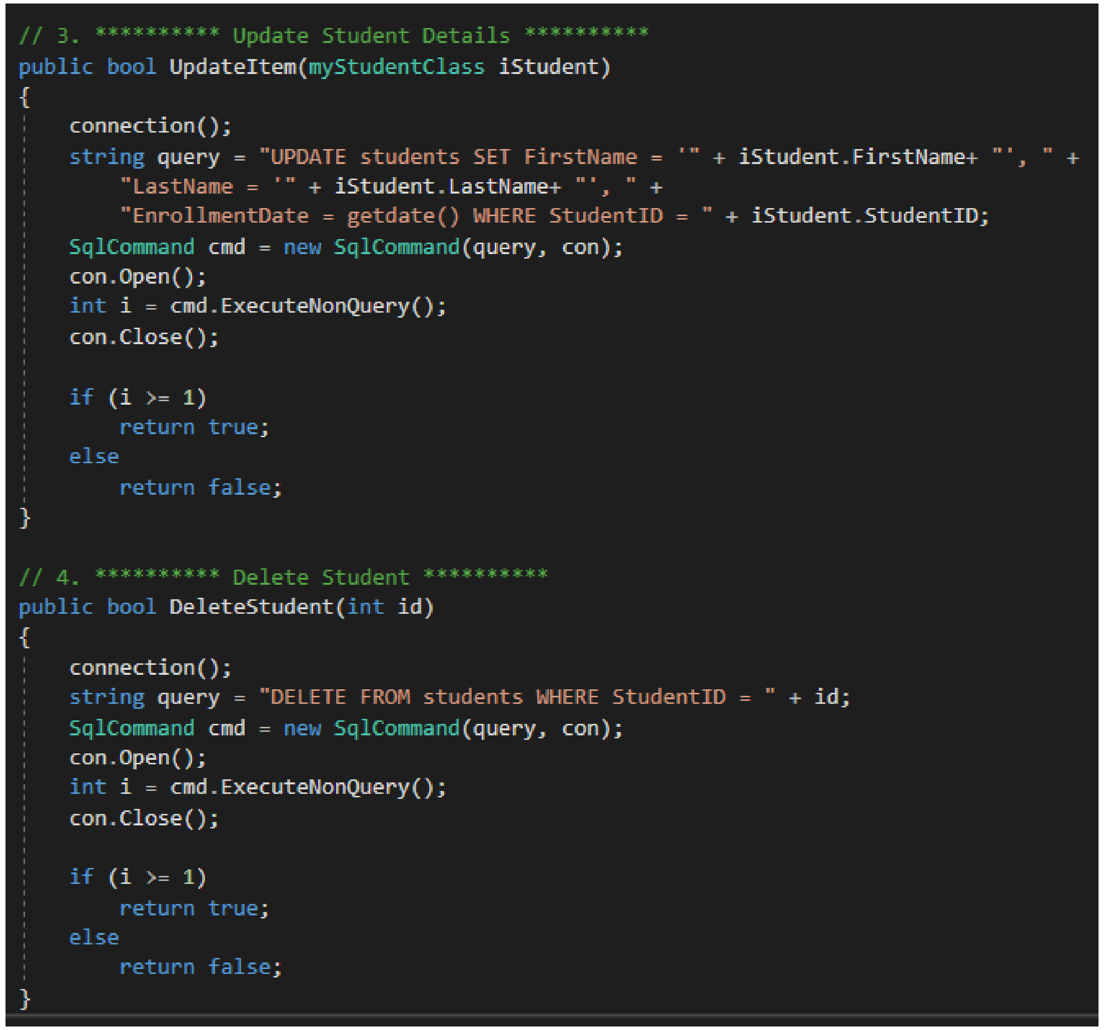
```
public class StudentDBHandler
{
    private SqlConnection con;
    private void connection()
    {
        string constring = ConfigurationManager.ConnectionStrings["myConnection"].ToString();
        con = new SqlConnection(constring);
    }

    // 1. ********** Insert Student **********
    public bool InsertStudent(myStudentClass  iStudent)
    {
        connection();
        string query = "INSERT INTO students (FirstName, LastName, EnrollmentDate) " +
            "VALUES('" + iStudent.FirstName + "','" + iStudent.LastName+ "', getdate())";
        SqlCommand cmd = new SqlCommand(query, con);
        con.Open();
        int i = cmd.ExecuteNonQuery();
        con.Close();

        if (i >= 1)
            return true;
        else
            return false;
    }
```



```
// 2. ********** Get All Student List **********
public List<myStudentClass> GetStudentList()
{
    connection();
    List<myStudentClass> iStudent = new List<myStudentClass>();

    string query = "SELECT * FROM students";
    SqlCommand cmd = new SqlCommand(query, con);
    SqlDataAdapter adapter = new SqlDataAdapter(cmd);
    DataTable dt = new DataTable();

    con.Open();
    adapter.Fill(dt);
    con.Close();

    foreach (DataRow dr in dt.Rows)
    {
        iStudent.Add(new myStudentClass
        {
            StudentID = Convert.ToInt32(dr["StudentID"]),
            FirstName = Convert.ToString(dr["FirstName"]),
            LastName = Convert.ToString(dr["LastName"]),
            EnrollmentDate = Convert.ToDateTime(dr["EnrollmentDate"])
        });
    }
    return iStudent;
}
```

Insert, Update, Delete without EF

Add the following code in StudentDBHandler.cs class.



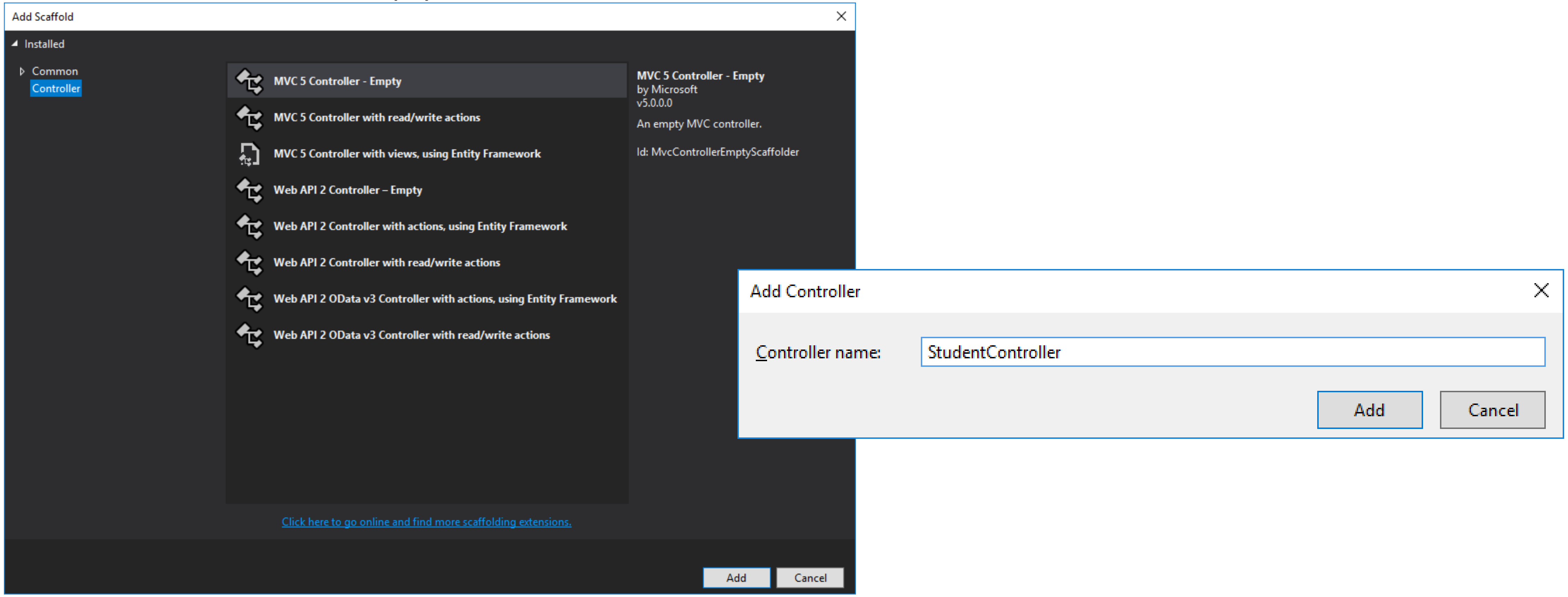
```
// 3. ********** Update Student Details **********
public bool UpdateItem(myStudentClass iStudent)
{
    connection();
    string query = "UPDATE students SET FirstName = '" + iStudent.FirstName+ "', " +
        "LastName = '" + iStudent.LastName+ "', " +
        "EnrollmentDate = getdate() WHERE StudentID = " + iStudent.StudentID;
    SqlCommand cmd = new SqlCommand(query, con);
    con.Open();
    int i = cmd.ExecuteNonQuery();
    con.Close();

    if (i >= 1)
        return true;
    else
        return false;
}

// 4. ********** Delete Student **********
public bool DeleteStudent(int id)
{
    connection();
    string query = "DELETE FROM students WHERE StudentID = " + id;
    SqlCommand cmd = new SqlCommand(query, con);
    con.Open();
    int i = cmd.ExecuteNonQuery();
    con.Close();

    if (i >= 1)
        return true;
    else
        return false;
}
```

Insert, Update, Delete without EF

Create a Controller "StudentController.cs"

Right Click on Controllers *Add → Controller*

Create MVC 5 Controller - Empty and click Add button.



Add Scaffold

Installed

- Common
- Controller

MVC 5 Controller - Empty
MVC 5 Controller with read/write actions
MVC 5 Controller with views, using Entity Framework
Web API 2 Controller – Empty
Web API 2 Controller with actions, using Entity Framework
Web API 2 Controller with read/write actions
Web API 2 OData v3 Controller with actions, using Entity Framework
Web API 2 OData v3 Controller with read/write actions

MVC 5 Controller - Empty
by Microsoft
v5.0.0.0

An empty MVC controller.

Id: MvcControllerEmptyScaffolder

Click here to go online and find more scaffolding extensions.

Add Cancel

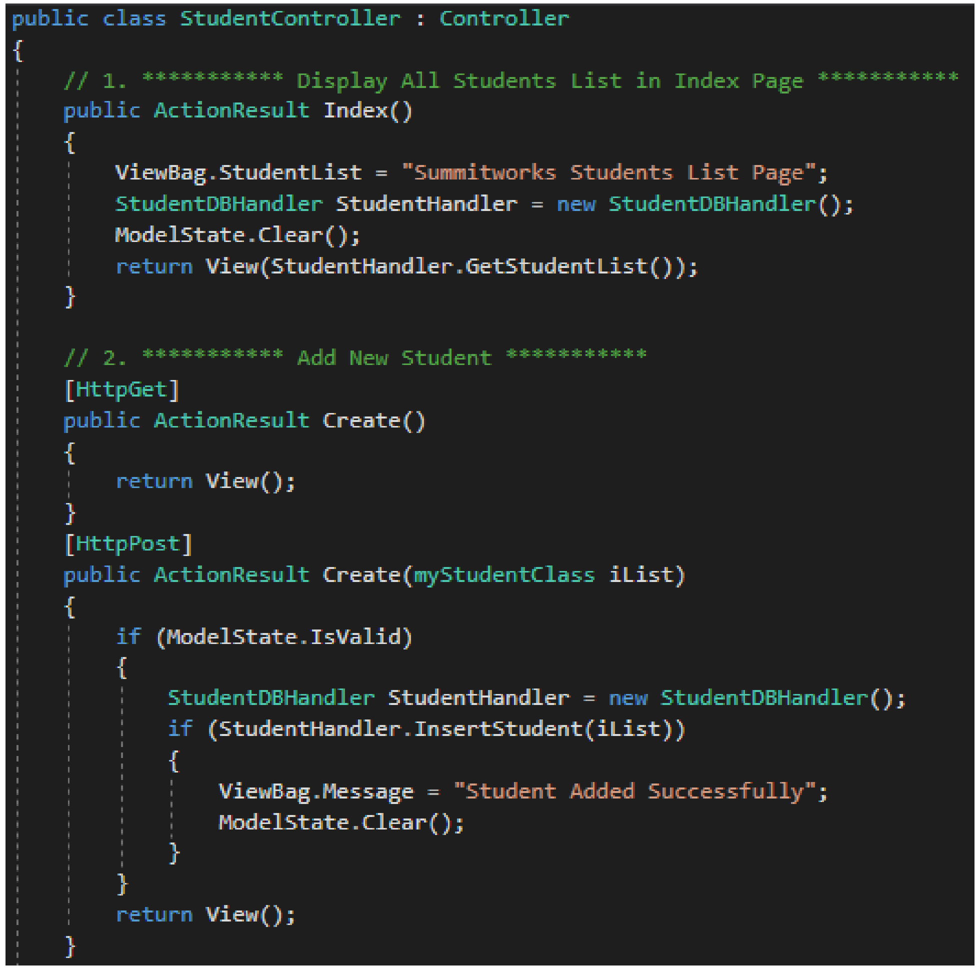
Add Controller

Controller name: StudentController

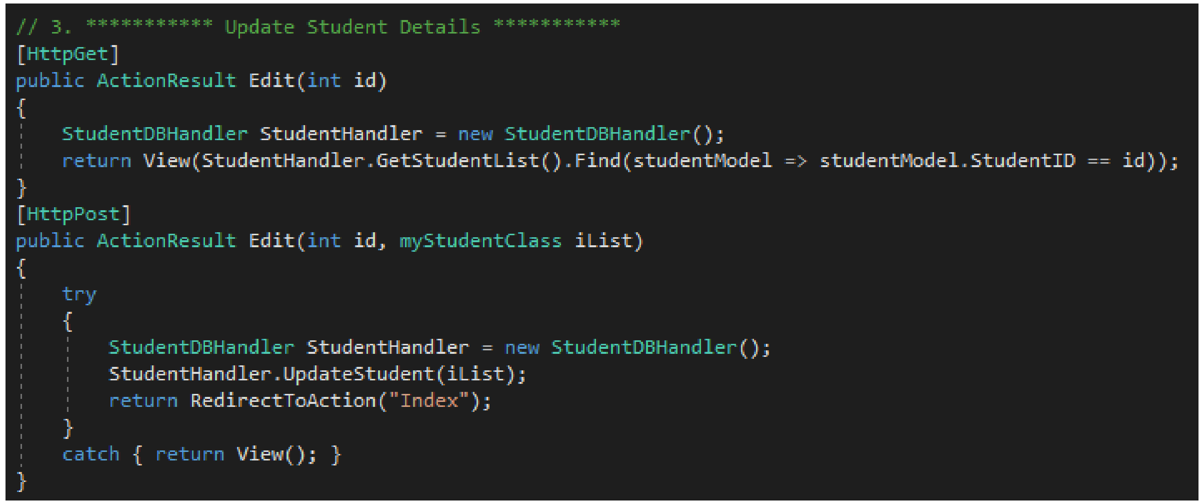
Add Cancel

Insert, Update, Delete without EF

Now Add following code in “StudentController.cs”

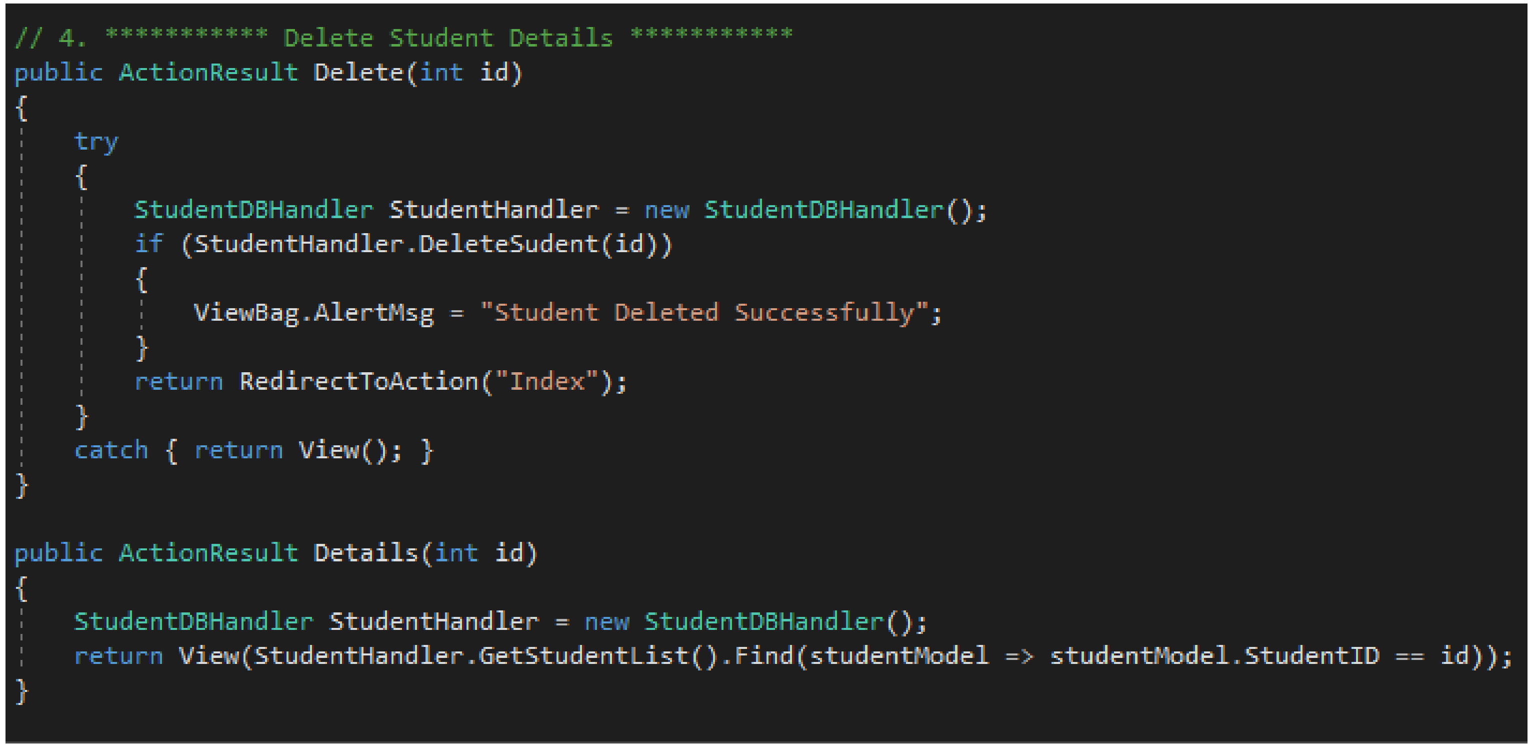


```
public class StudentController : Controller
{
    // 1. *********** Display All Students List in Index Page ***********
    public ActionResult Index()
    {
        ViewBag.StudentList = "Summitworks Students List Page";
        StudentDBHandler StudentHandler = new StudentDBHandler();
        ModelState.Clear();
        return View(StudentHandler.GetStudentList());
    }

    // 2. *********** Add New Student ***********
    [HttpGet]
    public ActionResult Create()
    {
        return View();
    }
    [HttpPost]
    public ActionResult Create(myStudentClass iList)
    {
        if (ModelState.IsValid)
        {
            StudentDBHandler StudentHandler = new StudentDBHandler();
            if (StudentHandler.InsertStudent(iList))
            {
                ViewBag.Message = "Student Added Successfully";
                ModelState.Clear();
            }
        }
        return View();
    }
```



```
// 3. *********** Update Student Details ***********
[HttpGet]
public ActionResult Edit(int id)
{
    StudentDBHandler StudentHandler = new StudentDBHandler();
    return View(StudentHandler.GetStudentList().Find(studentModel => studentModel.StudentID == id));
}
[HttpPost]
public ActionResult Edit(int id, myStudentClass iList)
{
    try
    {
        StudentDBHandler StudentHandler = new StudentDBHandler();
        StudentHandler.UpdateStudent(iList);
        return RedirectToAction("Index");
    }
    catch { return View(); }
}
```

Insert, Update, Delete without EF

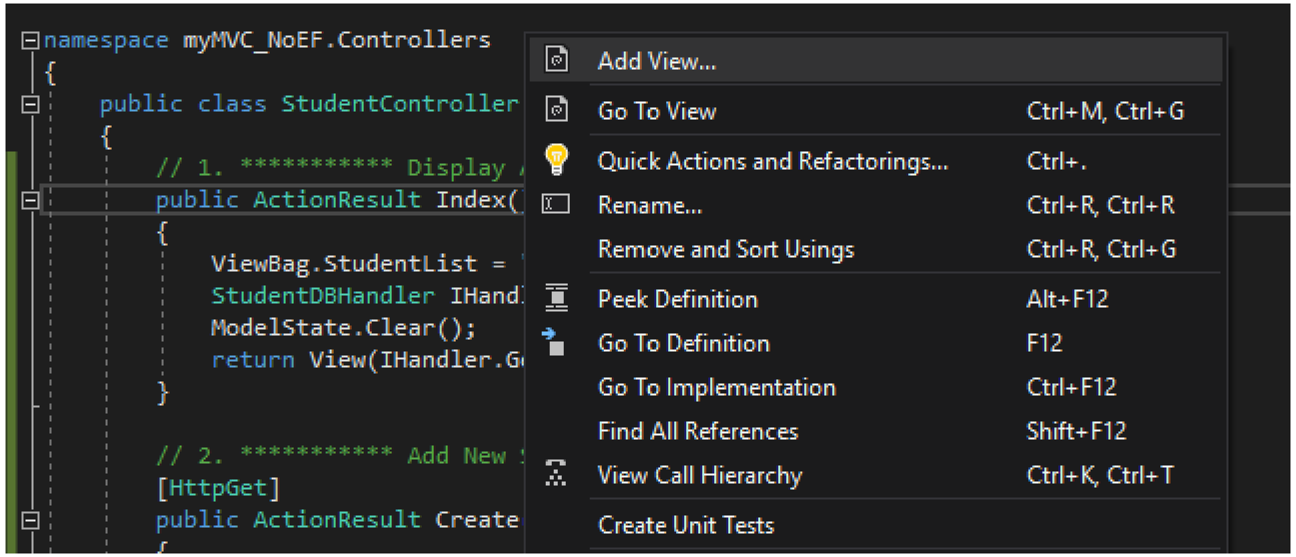


```
// 4. *********** Delete Student Details ***********
public ActionResult Delete(int id)
{
    try
    {
        StudentDBHandler StudentHandler = new StudentDBHandler();
        if (StudentHandler.DeleteSudent(id))
        {
            ViewBag.AlertMsg = "Student Deleted Successfully";
        }
        return RedirectToAction("Index");
    }
    catch { return View(); }
}

public ActionResult Details(int id)
{
    StudentDBHandler StudentHandler = new StudentDBHandler();
    return View(StudentHandler.GetStudentList().Find(studentModel => studentModel.StudentID == id));
}
```

Insert, Update, Delete without EF

Add View pages from Controller's Action Method

Go to "StudentController" and Right click on *Index()* Action Method then select Add View.



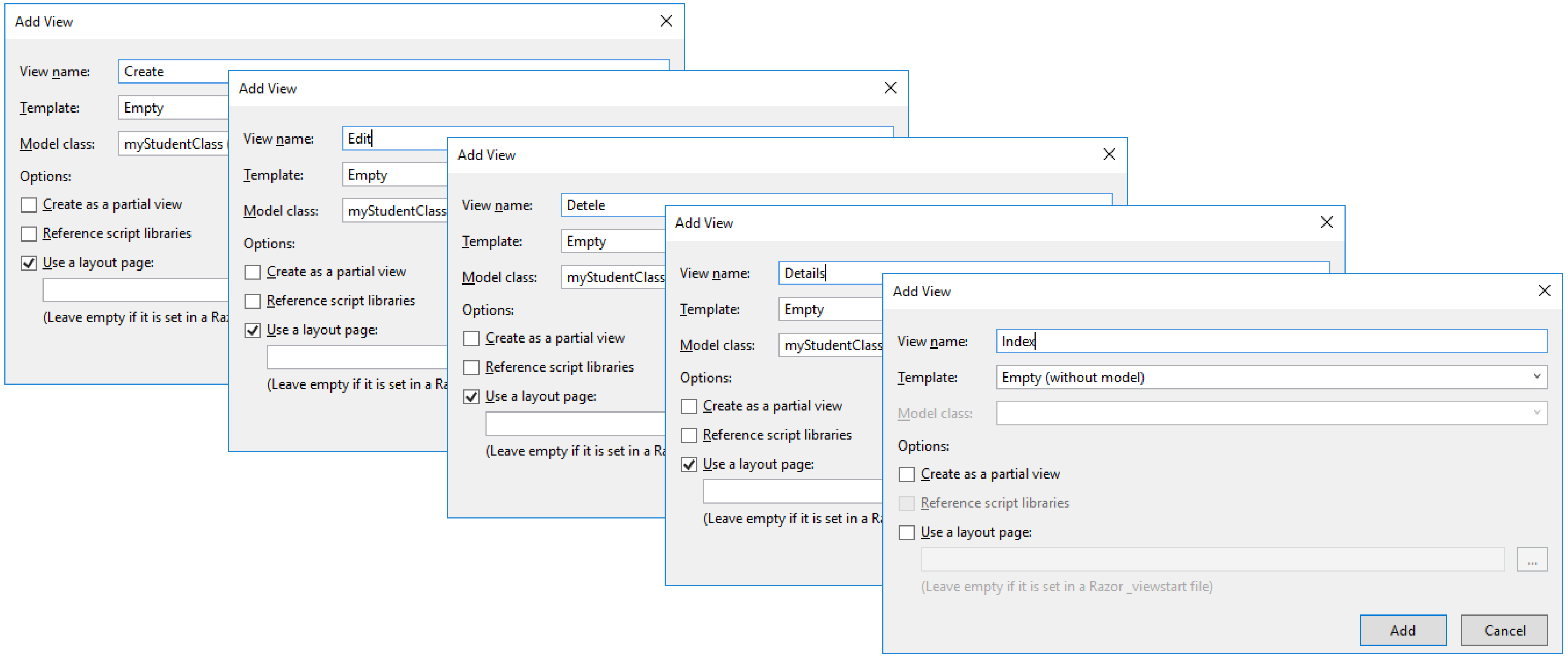
```
namespace myMVC_NoEF.Controllers
{
    public class StudentController
    {
        // 1. *********** Display
        public ActionResult Index(
        {
            ViewBag.StudentList =
            StudentDBHandler IHand
            ModelState.Clear();
            return View(IHandler.G
        }

        // 2. *********** Add New
        [HttpGet]
        public ActionResult Create
```

Add View...
Go To View Ctrl+M, Ctrl+G
Quick Actions and Refactorings... Ctrl+.
Rename... Ctrl+R, Ctrl+R
Remove and Sort Usings Ctrl+R, Ctrl+G
Peek Definition Alt+F12
Go To Definition F12
Go To Implementation Ctrl+F12
Find All References Shift+F12
View Call Hierarchy Ctrl+K, Ctrl+T
Create Unit Tests

Insert, Update, Delete without EF

Add View Dialog box will appear. Fill details as in this picture and click to Add button.

Using the same process Add View for Index(), Create(), Edit(), Delete() and Details() action method.



Add View

View name: Create

Template: Empty

Model class: myStudentClass (

Options:

☐ Create as a partial view

☐ Reference script libraries

☑ Use a layout page:

(Leave empty if it is set in a Ra

Add View

View name: Edit

Template: Empty

Model class: myStudentClass

Options:

☐ Create as a partial view

☐ Reference script libraries

☑ Use a layout page:

(Leave empty if it is set in a Ra

Add View

View name: Detele

Template: Empty

Model class: myStudentClass

Options:

☐ Create as a partial view

☐ Reference script libraries

☑ Use a layout page:

(Leave empty if it is set in a Ra

Add View

View name: Details

Template: Empty

Model class: myStudentClass

Options:

☐ Create as a partial view

☐ Reference script libraries

☑ Use a layout page:

(Leave empty if it is set in a Ra

Add View

View name: Index

Template: Empty (without model)

Model class:

Options:

☐ Create as a partial view

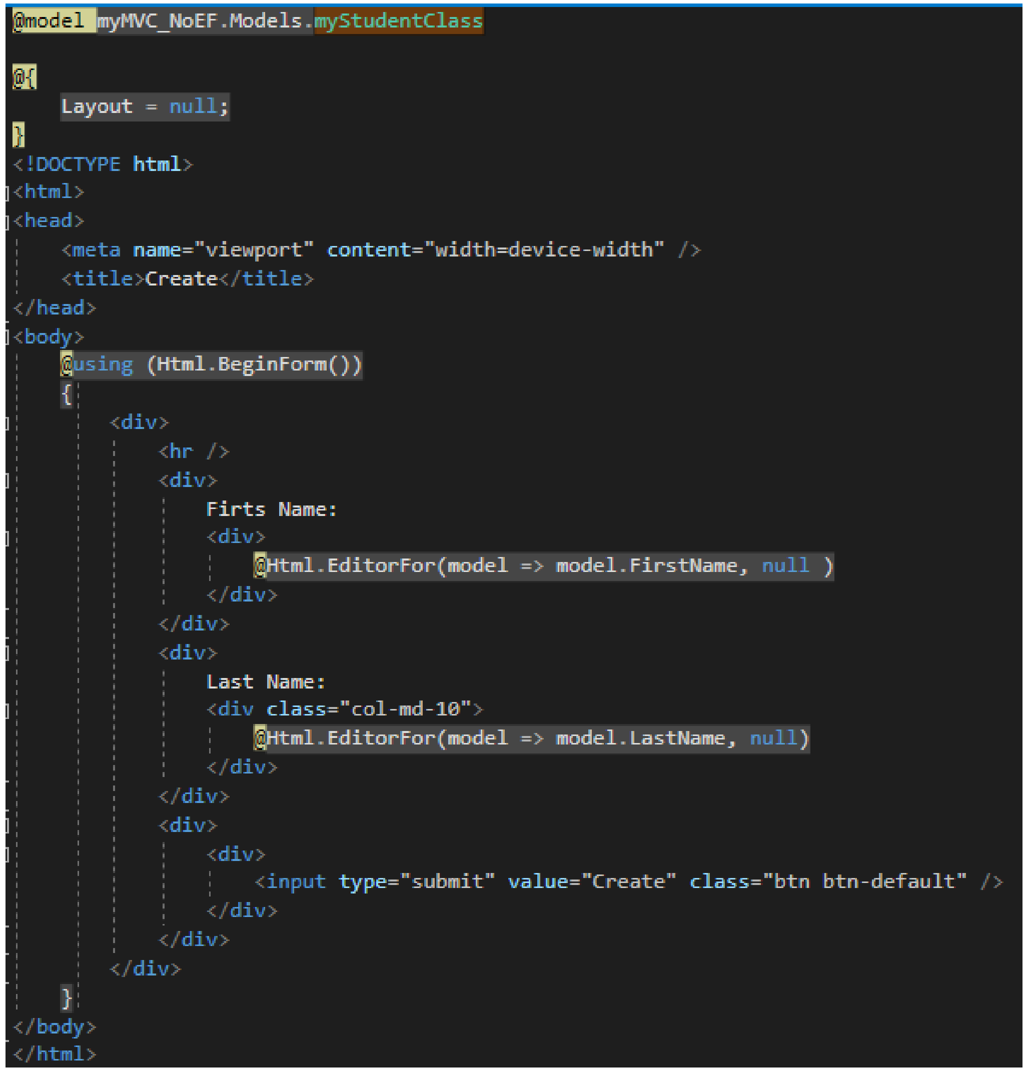
☐ Reference script libraries

☐ Use a layout page:

(Leave empty if it is set in a Razor _viewstart file)

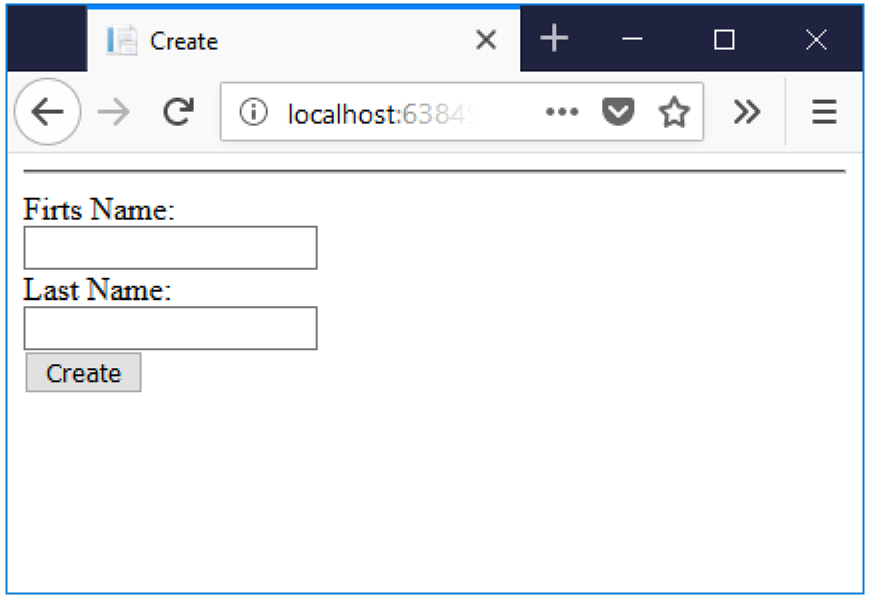
Insert, Update, Delete without EF

Now Add following code in “Create.cshtml”



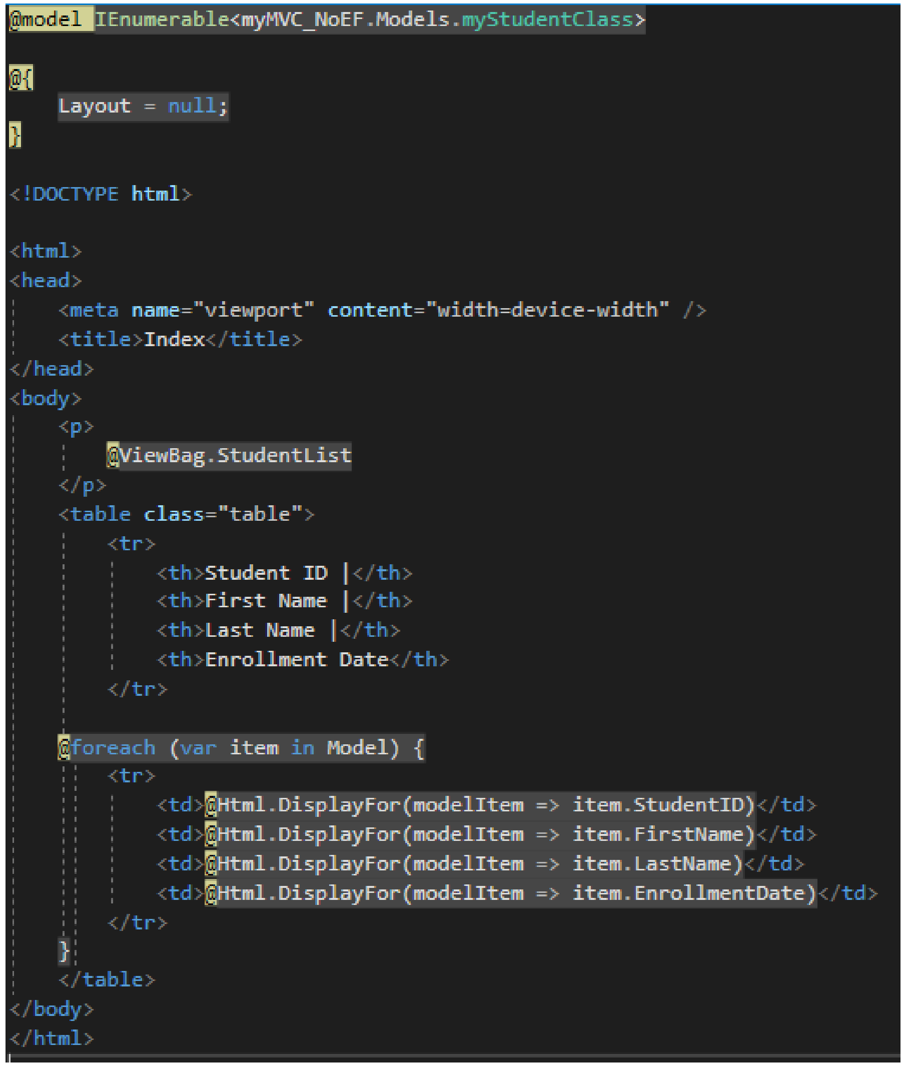
```
@model myMVC_NoEF.Models.myStudentClass

@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Create</title>
</head>
<body>
    @using (Html.BeginForm())
    {
        <div>
            <hr />
            <div>
                Firts Name:
                <div>
                    @Html.EditorFor(model => model.FirstName, null )
                </div>
            </div>
            <div>
                Last Name:
                <div class="col-md-10">
                    @Html.EditorFor(model => model.LastName, null)
                </div>
            </div>
            <div>
                <div>
                    <input type="submit" value="Create" class="btn btn-default" />
                </div>
            </div>
        </div>
    }
</body>
</html>
```

Press F5 to Run your project and navigate to *[/Student/Create]*



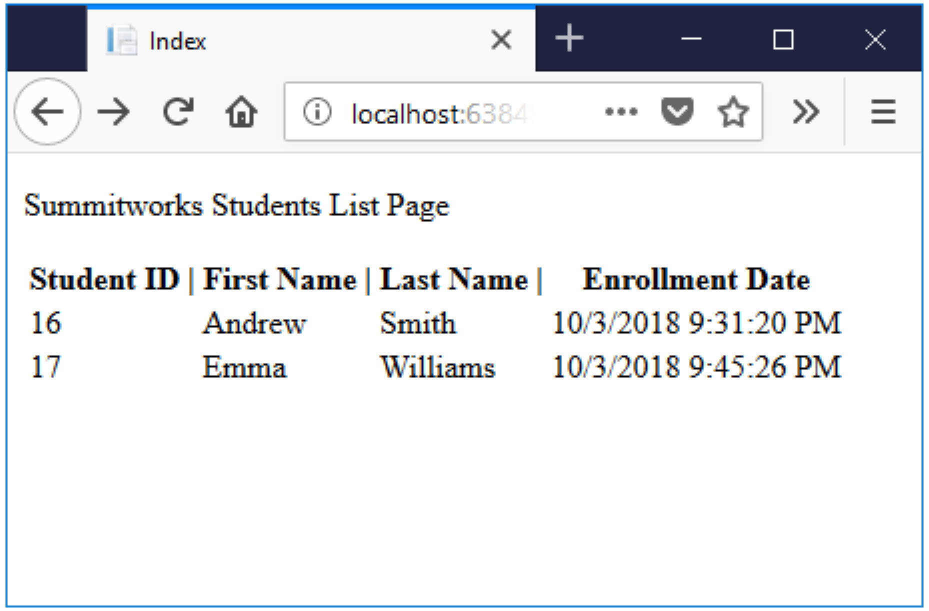
Insert, Update, Delete without EF

Now Add following code in “Index.cshtml”



```
@model IEnumerable<myMVC_NoEF.Models.myStudentClass>

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <p>
        @ViewBag.StudentList
    </p>
    <table class="table">
        <tr>
            <th>Student ID |</th>
            <th>First Name |</th>
            <th>Last Name |</th>
            <th>Enrollment Date</th>
        </tr>

    @foreach (var item in Model) {
        <tr>
            <td>@Html.DisplayFor(modelItem => item.StudentID)</td>
            <td>@Html.DisplayFor(modelItem => item.FirstName)</td>
            <td>@Html.DisplayFor(modelItem => item.LastName)</td>
            <td>@Html.DisplayFor(modelItem => item.EnrollmentDate)</td>
        </tr>
    }
    </table>
</body>
</html>
```

Press F5 to Run your project and navigate to */Student/Index*

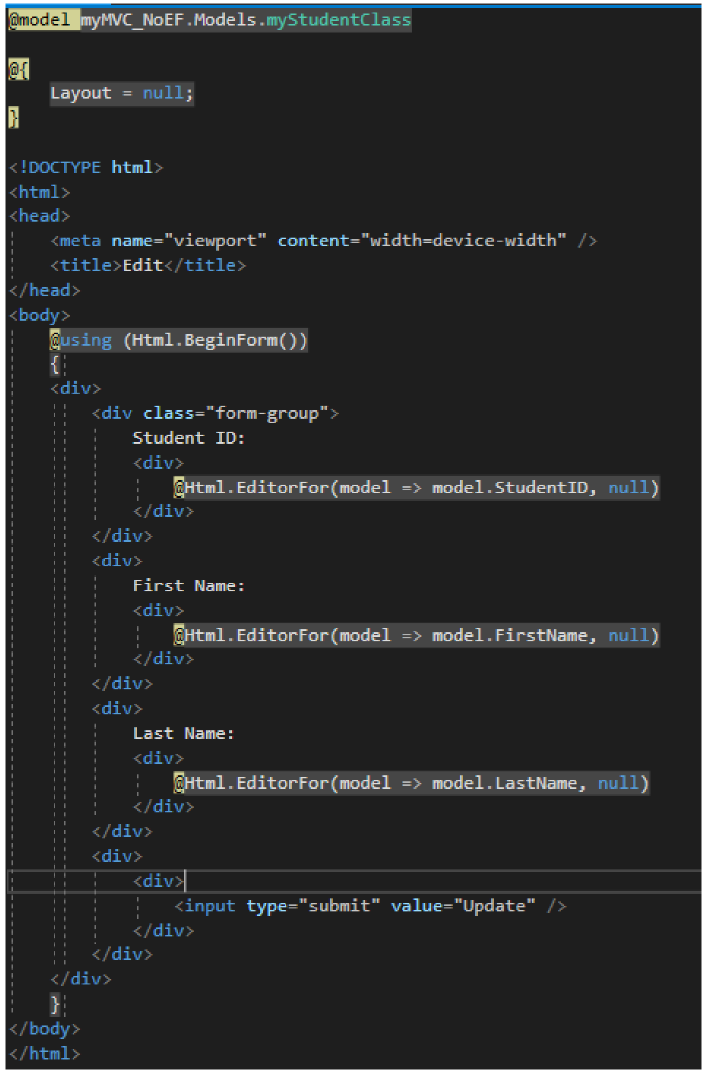


Summitworks Students List Page

Student ID |	First Name |	Last Name |	Enrollment Date
16	Andrew	Smith	10/3/2018 9:31:20 PM
17	Emma	Williams	10/3/2018 9:45:26 PM

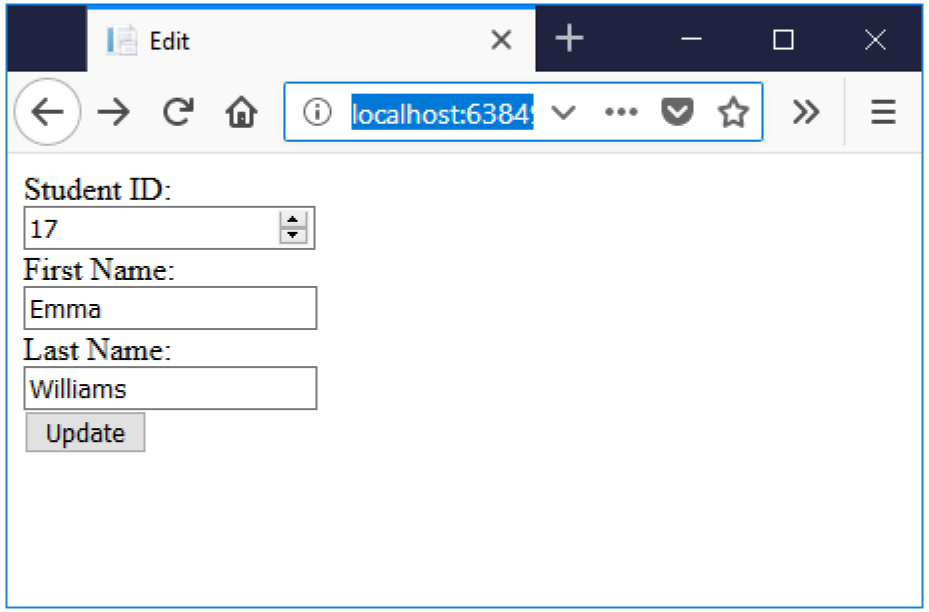
Insert, Update, Delete without EF

Now Add following code in “Update.cshtml”



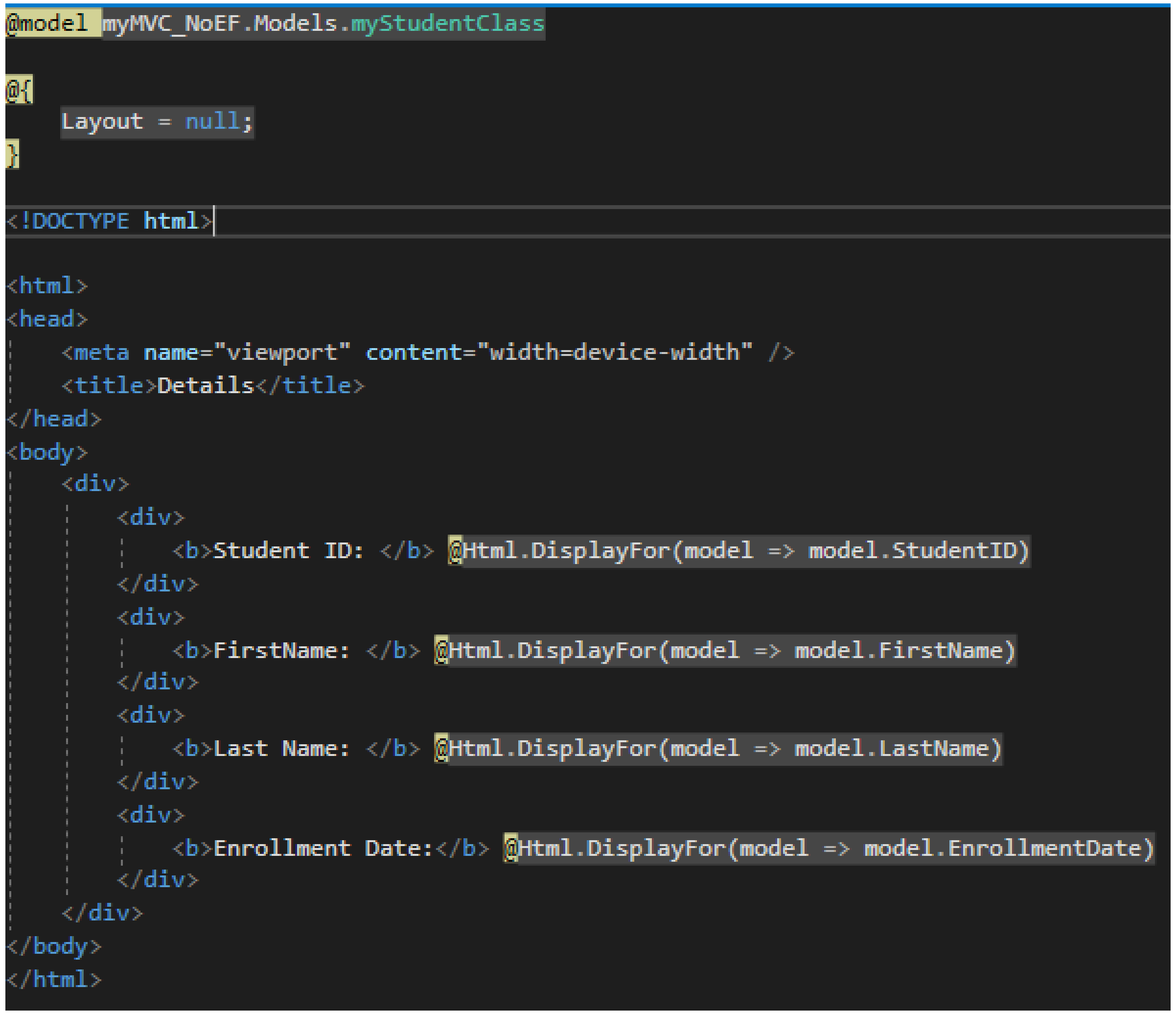
```
@model myMVC_NoEF.Models.myStudentClass

@{
    Layout = null;
}

<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Edit</title>
</head>
<body>
    @using (Html.BeginForm())
    {
    <div>
        <div class="form-group">
            Student ID:
            <div>
                @Html.EditorFor(model => model.StudentID, null)
            </div>
        </div>
        <div>
            First Name:
            <div>
                @Html.EditorFor(model => model.FirstName, null)
            </div>
        </div>
        <div>
            Last Name:
            <div>
                @Html.EditorFor(model => model.LastName, null)
            </div>
        </div>
        <div>
            <div>
                <input type="submit" value="Update" />
            </div>
        </div>
    </div>
    }
</body>
</html>
```

Press F5 to Run your project and navigate to */Student/Edit/{id}*



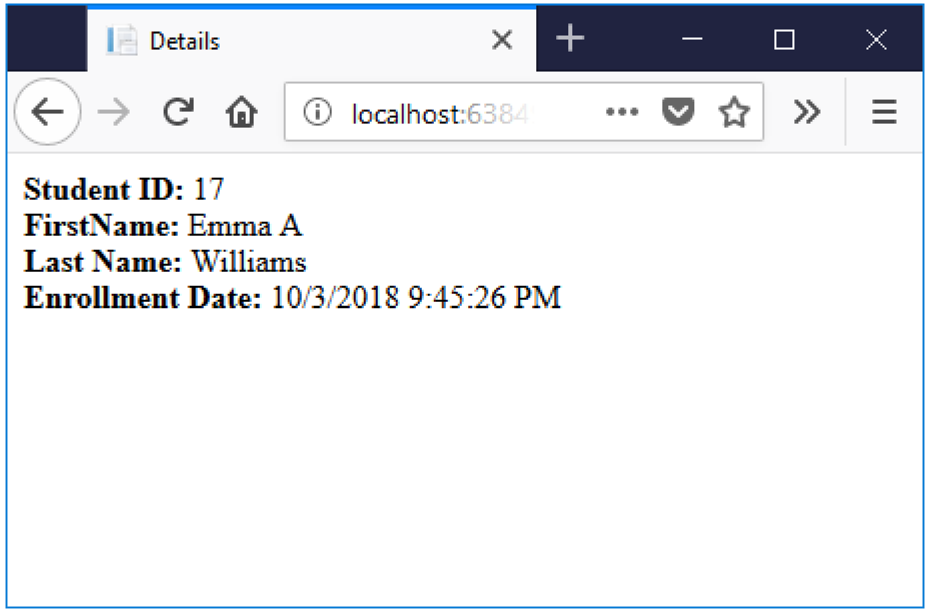
Insert, Update, Delete without EF

Now Add following code in "Details.cshtml"



```
@model myMVC_NoEF.Models.myStudentClass

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Details</title>
</head>
<body>
    <div>
        <div>
            <b>Student ID: </b> @Html.DisplayFor(model => model.StudentID)
        </div>
        <div>
            <b>FirstName: </b> @Html.DisplayFor(model => model.FirstName)
        </div>
        <div>
            <b>Last Name: </b> @Html.DisplayFor(model => model.LastName)
        </div>
        <div>
            <b>Enrollment Date:</b> @Html.DisplayFor(model => model.EnrollmentDate)
        </div>
    </div>
</body>
</html>
```

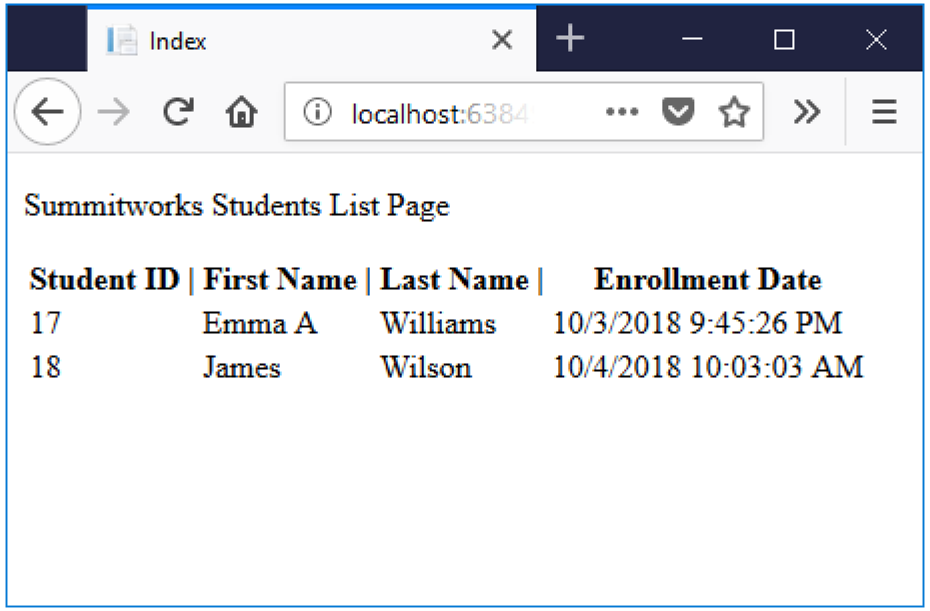
Press F5 to Run your project and navigate to *__/Student/Details/{id}__*



Student ID: 17
FirstName: Emma A
Last Name: Williams
Enrollment Date: 10/3/2018 9:45:26 PM

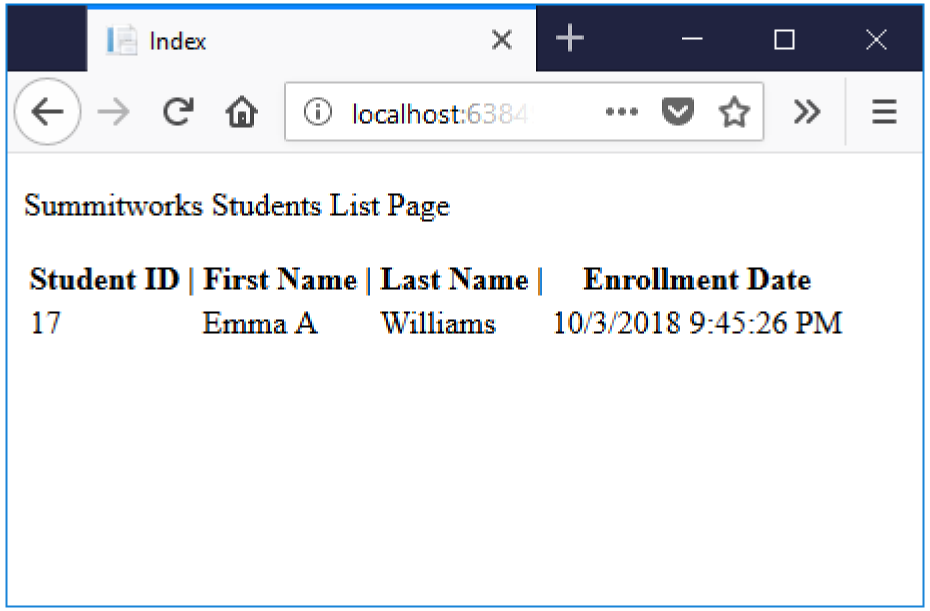
Insert, Update, Delete without EF

To delete a Student record navigate to ***/Student/Delete/{id}***



Summitworks Students List Page

Student ID	First Name	Last Name	Enrollment Date
17	Emma A	Williams	10/3/2018 9:45:26 PM
18	James	Wilson	10/4/2018 10:03:03 AM



Summitworks Students List Page

Student ID	First Name	Last Name	Enrollment Date
17	Emma A	Williams	10/3/2018 9:45:26 PM

Insert, Update, Delete using Entity Framework

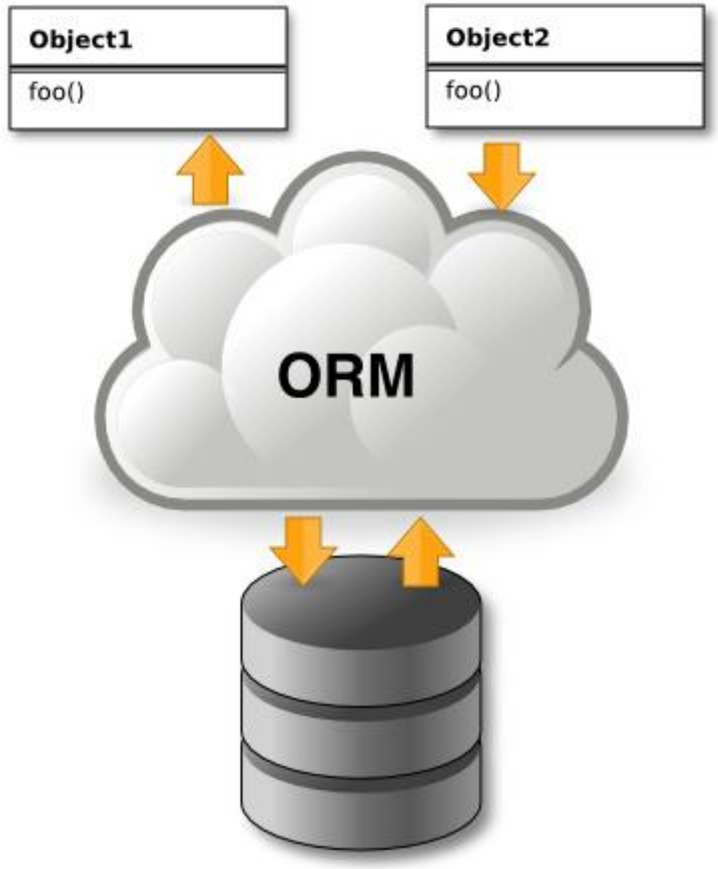
What is Entity Framework

Entity framework is an ORM (Object Relational Mapping) tool.

Object Relational Mapping (ORM) is a technique of accessing a relational database; .i.e., whatever has tables and store procedure these things we interact with database in class, method and property of the object format.

Features of ORM in entity framework

- Map our database types to our code types
- Avoid repetitive data access code
- Access code automatically based on the data model class
- Support a clean separation of concerns and independent development that allows parallel, simultaneous development of application
- Easily reuse the data object
- Application Maintainability



Advantages of EF

- It provides auto generated code
- It reduce development time
- It reduce development cost
- It enables developers to visually design models and mapping of database
- It provides capability of programming a conceptual model.
- It provides unique syntax (LINQ / Yoda) for all object queries whether it is database or not
- It allow multiple conceptual models to mapped to a single storage schema
- It’s easy to map business objects (with drag & drop tables).

Usage of Entity Framework

The Entity Framework provides three approaches to create an entity model.

- Code First Approach
- Database First Approach
- Model First Approach

Code First Approach

Some developers prefer to work with the Designer in Code while others would rather just work with their code. For those developers, Entity Framework has a modeling workflow referred to as Code First.

- Code First modeling workflow targets a database that doesn’t exist and Code First will create it.
- It can also be used if you have an empty database and then Code First will add new tables too.
- Code First allows you to define your model using C# or VB.Net classes.
- Additional configuration can optionally be performed using attributes on your classes and properties or by using a fluent API.

Usage of Entity Framework

Model First Approach

- Model First is great for when you're starting a new project where the database doesn't even exist yet.
- The model is stored in an EDMX file and can be viewed and edited in the Entity Framework Designer.
- In Model First, you define your model in an Entity Framework designer then generate SQL, which will create database schema to match your model and then you execute the SQL to create the schema in your database.
- The classes that you interact with in your application are automatically generated from the EDMX file.

Usage of Entity Framework

Database First Approach

- The Database First Approach provides an alternative to the Code First and Model First approaches to the Entity Data Model. It creates model codes (classes, properties, DbContext etc.) from the database in the project and those classes become the link between the database and controller.

- The Database First Approach creates the entity framework from an existing database. We use all other functionalities, such as the model/database sync and the code generation, in the same way we used them in the Model First approach.

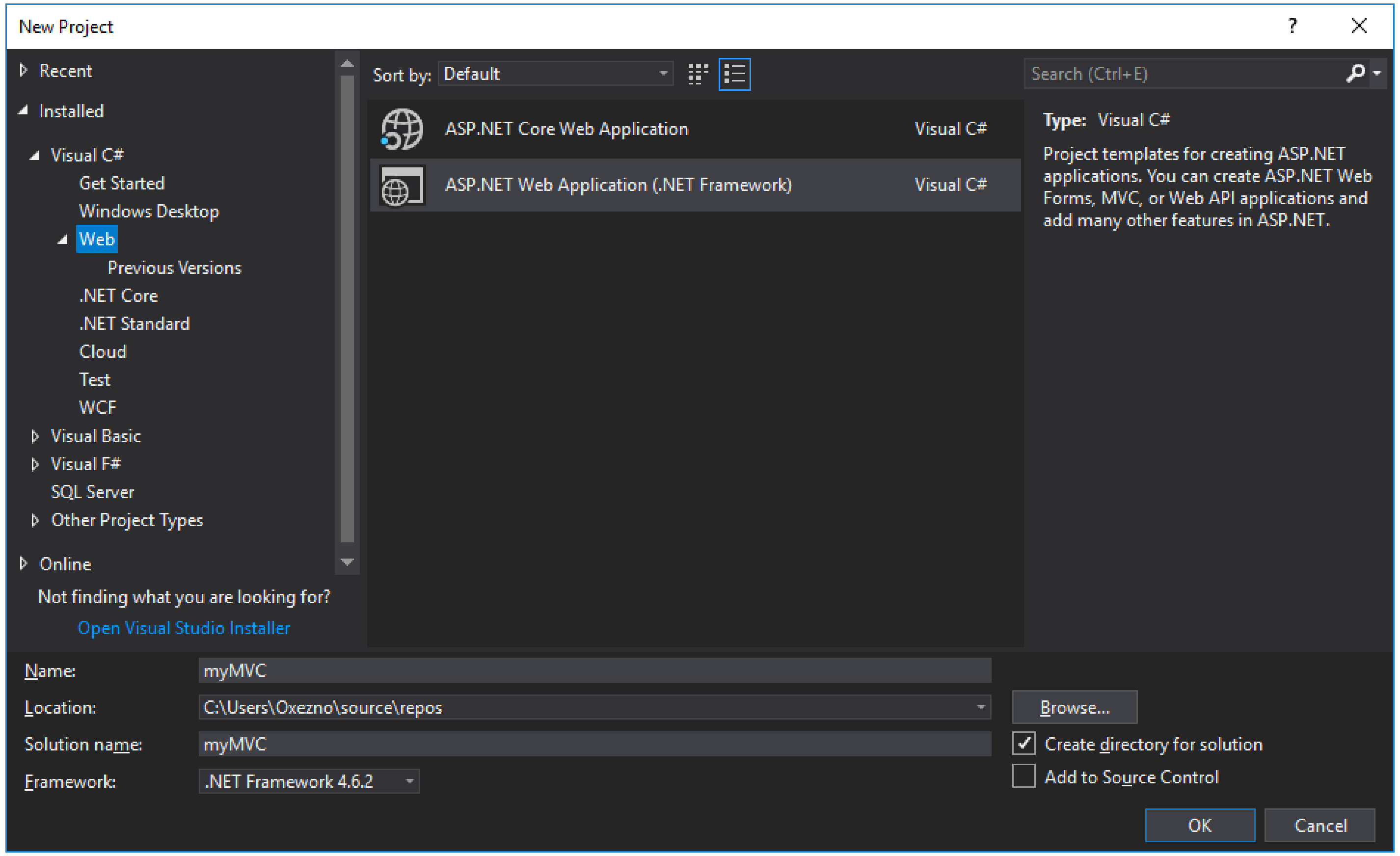
Insert, Update, Delete with EF

Create a new MVC Application

Start your Visual Studio and select:

File → New → Project. Select Web → ASP.NET Web Application (.NET Framework)

Name this project as **myMVC_EF**


New Project

Type: Visual C#

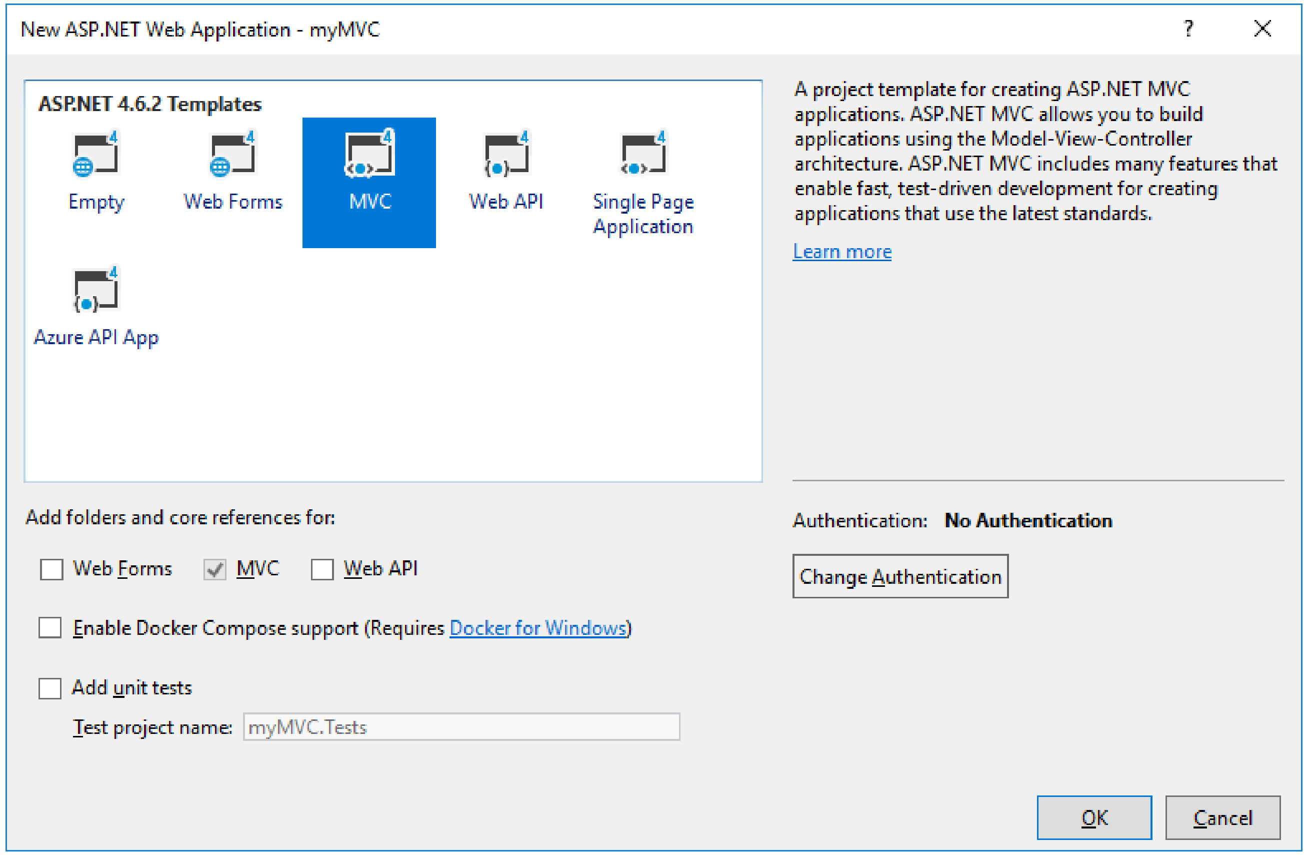
Project templates for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Name:	myMVC
Location:	C:\Users\Oxezno\source\repos
Solution name:	myMVC
Framework:	.NET Framework 4.6.2

[✓] Create directory for solution
[ ] Add to Source Control

Insert, Update, Delete with EF

Select MVC Template. Click OK.



New ASP.NET Web Application - myMVC

ASP.NET 4.6.2 Templates

A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

Learn more

Add folders and core references for:

☐ Web Forms ☑ MVC ☐ Web API

☐ Enable Docker Compose support (Requires Docker for Windows)

☐ Add unit tests

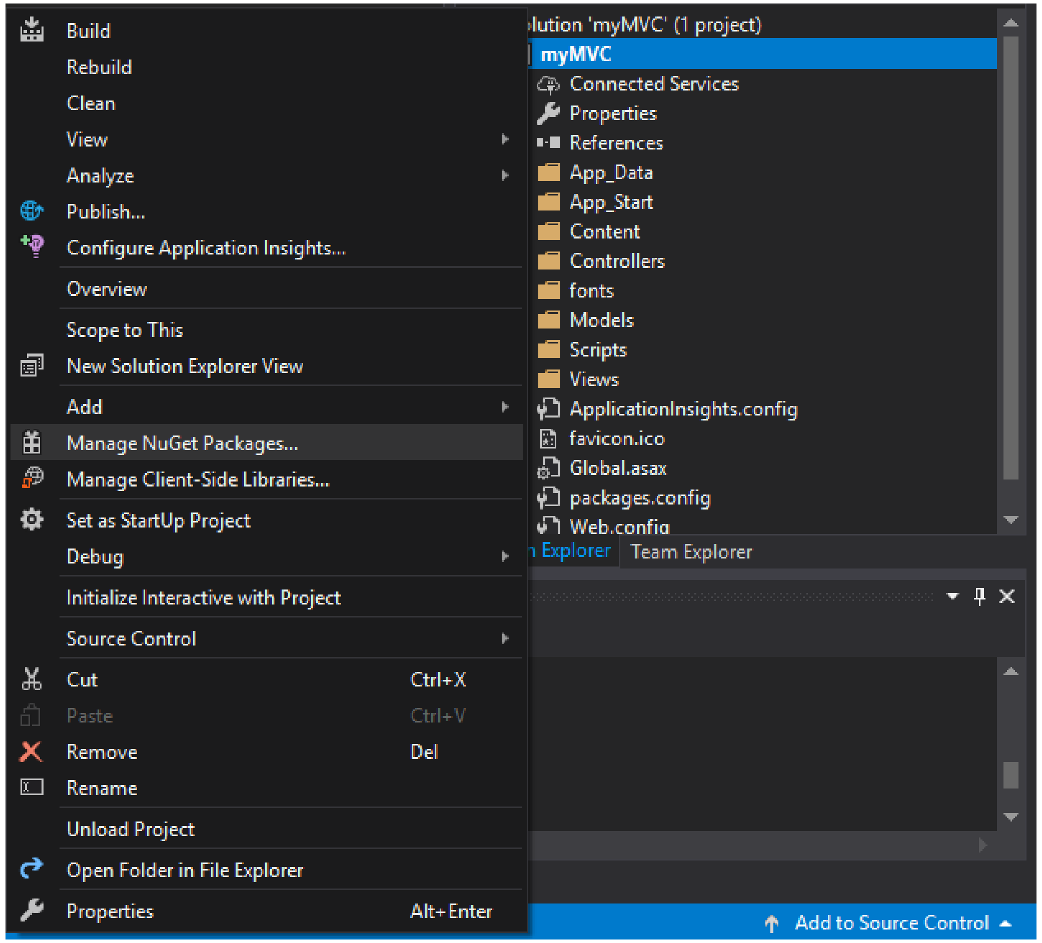
Test project name: myMVC.Tests

Authentication: **No Authentication**

Insert, Update, Delete with EF

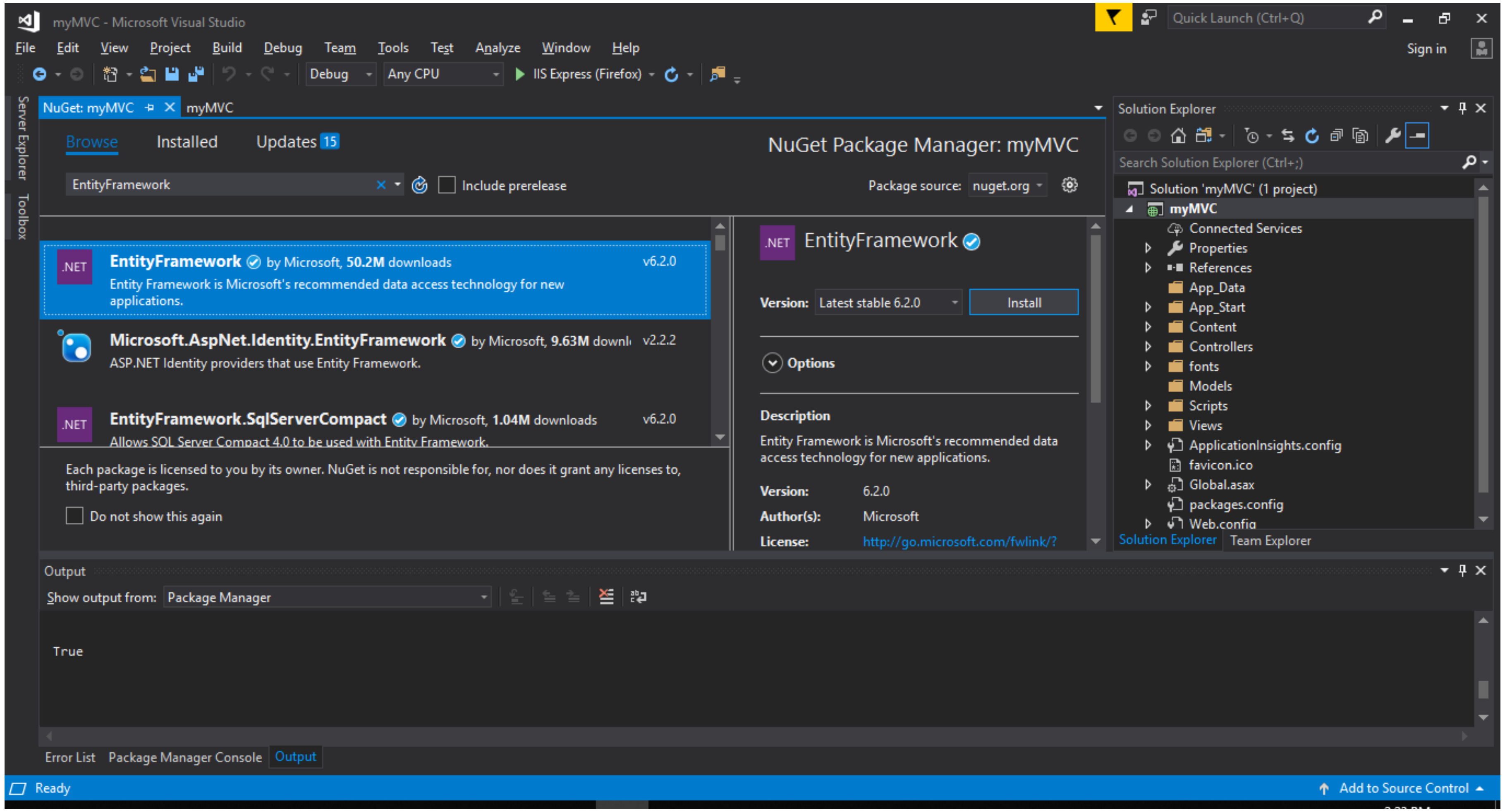
Install Entity Framework

Right Click on your project and select “Manage NuGet Packages”.



Insert, Update, Delete with EF

Search for “Entity Framework” and click install.



EntityFramework

EntityFramework by Microsoft, 50.2M downloads v6.2.0
Entity Framework is Microsoft's recommended data access technology for new applications.

Microsoft.AspNet.Identity.EntityFramework by Microsoft, 9.63M downl v2.2.2
ASP.NET Identity providers that use Entity Framework.

EntityFramework.SqlServerCompact by Microsoft, 1.04M downloads v6.2.0
Allows SQL Server Compact 4.0 to be used with Entity Framework.

Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages.

Do not show this again

EntityFramework

Version: Latest stable 6.2.0

Description
Entity Framework is Microsoft's recommended data access technology for new applications.

Version: 6.2.0
Author(s): Microsoft
License: http://go.microsoft.com/fwlink/?

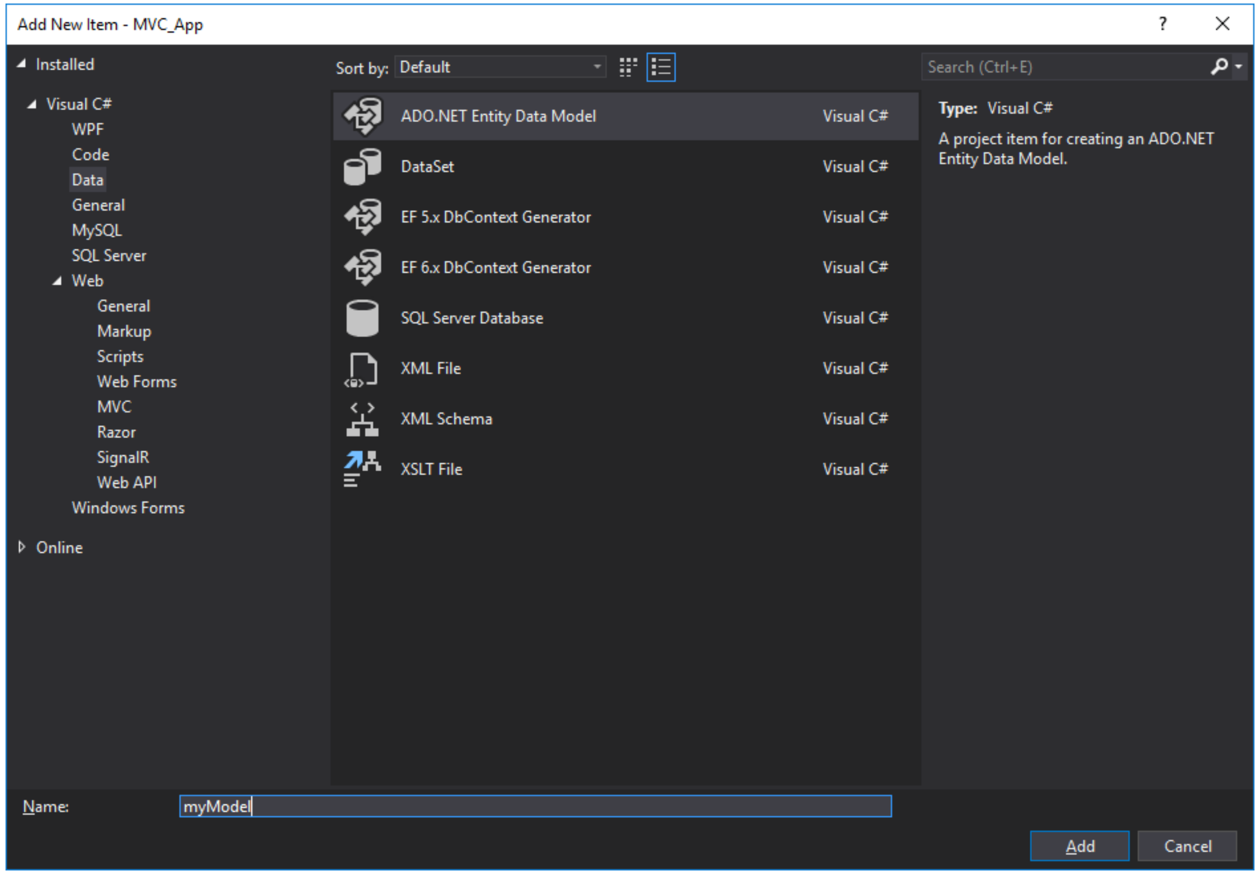
```
True
```

Insert, Update, Delete with EF

Generate Models Using Database Entities

Now you can go ahead and create a new Model. Right-click on the Models folder in your project and select:

Add → New Item.
Then, select ADO.NET Entity Data Model.



ADO.NET Entity Data Model	Visual C#
DataSet	Visual C#
EF 5.x DbContext Generator	Visual C#
EF 6.x DbContext Generator	Visual C#
SQL Server Database	Visual C#
XML File	Visual C#
XML Schema	Visual C#
XSLT File	Visual C#

Type: Visual C#

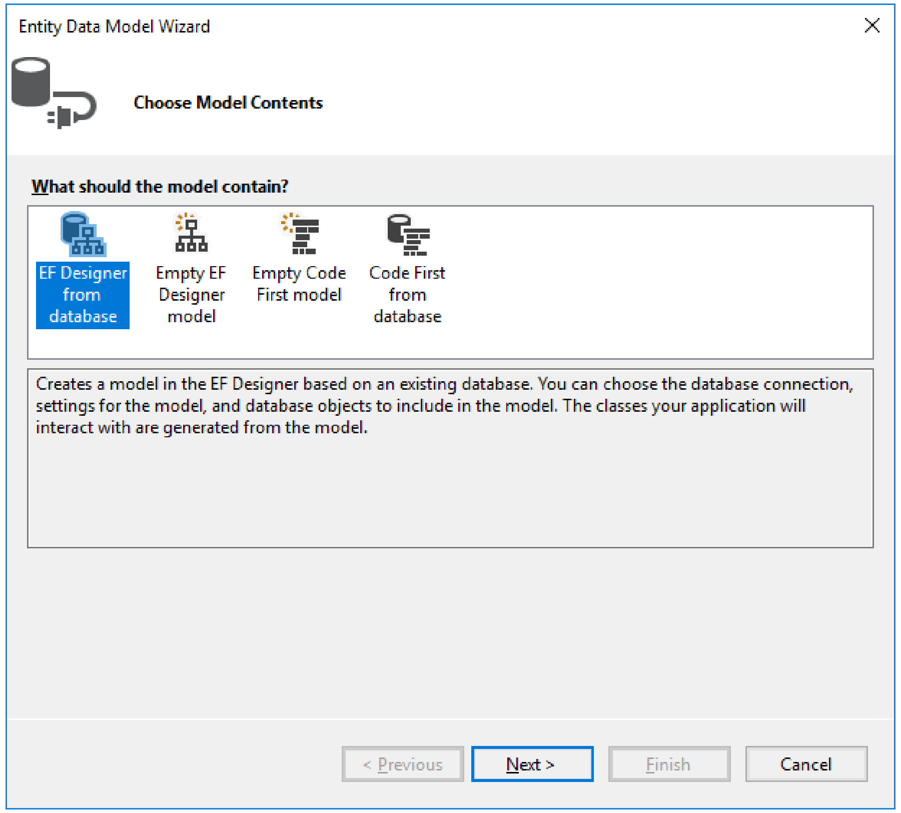
A project item for creating an ADO.NET Entity Data Model.

Name: myModel

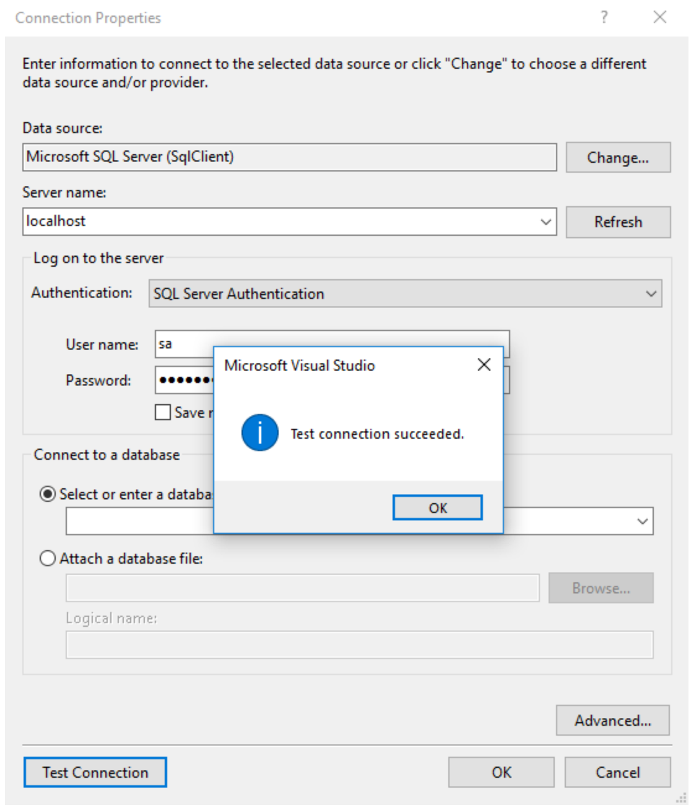
Insert, Update, Delete with EF

Choose EF Designer From Database and click Next. Set the Connection to your SQL database.

Select your database and click Test Connection. A screen similar to the following will follow. Click Next.



Creates a model in the EF Designer based on an existing database. You can choose the database connection, settings for the model, and database objects to include in the model. The classes your application will interact with are generated from the model.



Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source: Microsoft SQL Server (SqlClient)

Server name: localhost

Authentication: SQL Server Authentication

User name: sa

Password: ••••••

Save

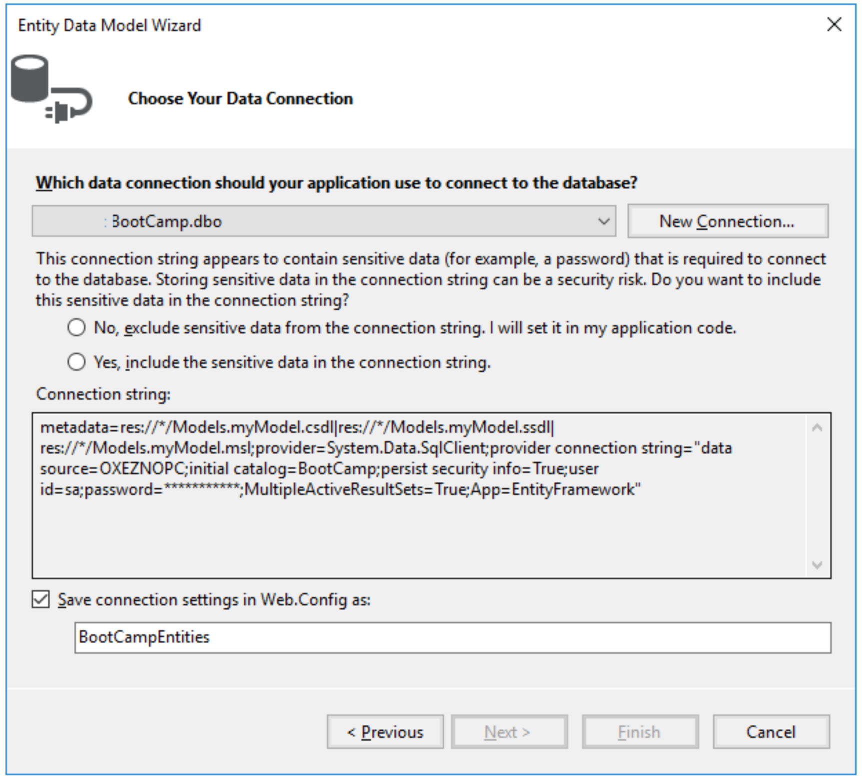
Select or enter a databa

Attach a database file:

Test connection succeeded.

Insert, Update, Delete with EF

Select "Yes, include the sensitive data…". Check the tables that you want to include and click "Finish".



Entity Data Model Wizard

Choose Your Data Connection

Which data connection should your application use to connect to the database?

BootCamp.dbo | New Connection...

This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

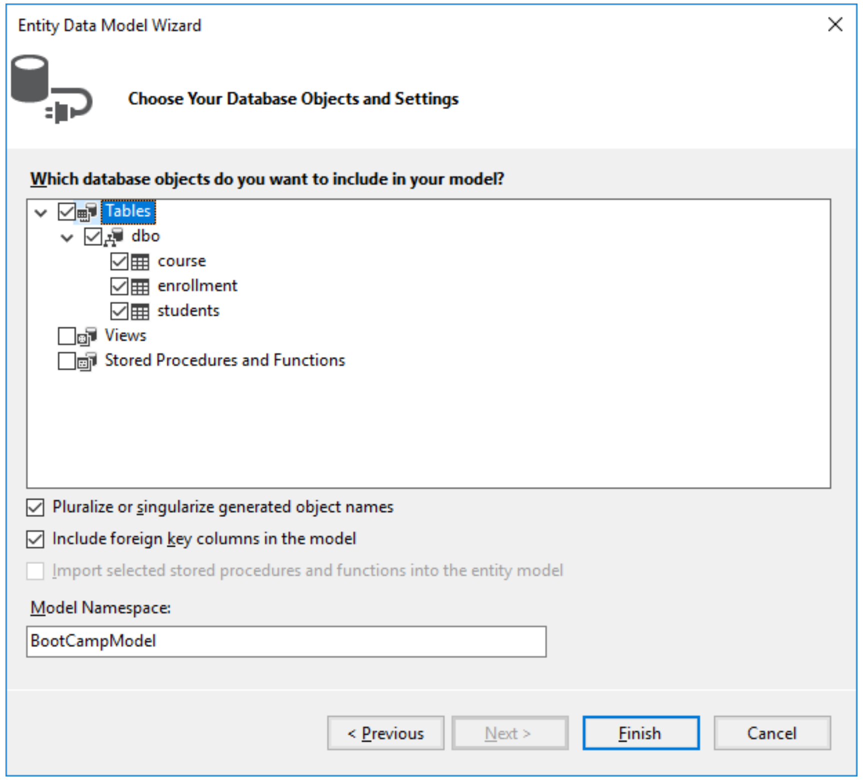
- ○ No, exclude sensitive data from the connection string. I will set it in my application code.
- ○ Yes, include the sensitive data in the connection string.

Connection string:

```
metadata=res://*/Models.myModel.csdl|res://*/Models.myModel.ssdl|
res://*/Models.myModel.msl;provider=System.Data.SqlClient;provider connection string="data
source=OXEZNOPC;initial catalog=BootCamp;persist security info=True;user
id=sa;password=***********;MultipleActiveResultSets=True;App=EntityFramework"
```

☑ Save connection settings in Web.Config as:

BootCampEntities



Entity Data Model Wizard

Choose Your Database Objects and Settings

Which database objects do you want to include in your model?

- ☑ Tables
  - ☑ dbo
    - ☑ course
    - ☑ enrollment
    - ☑ students
- ☐ Views
- ☐ Stored Procedures and Functions

☑ Pluralize or singularize generated object names

☑ Include foreign key columns in the model

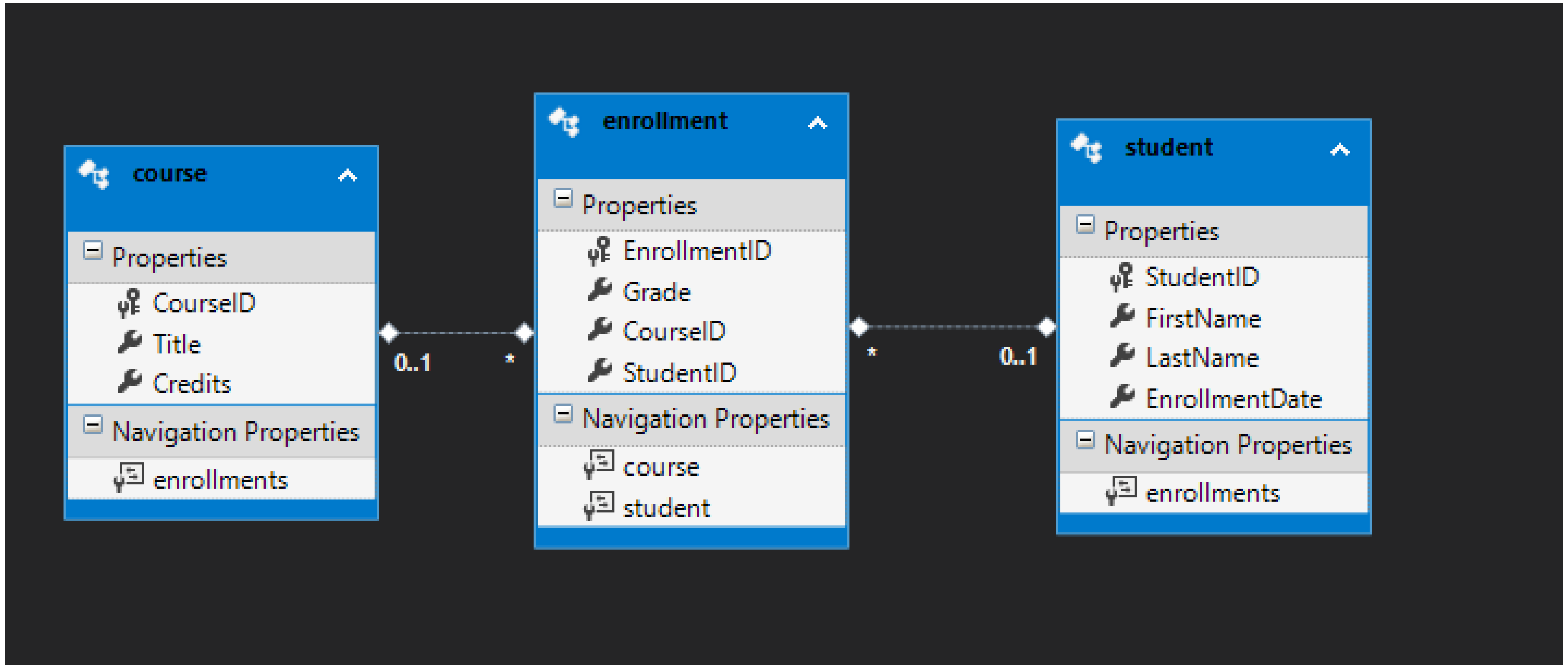
☐ Import selected stored procedures and functions into the entity model

Model Namespace:

BootCampModel

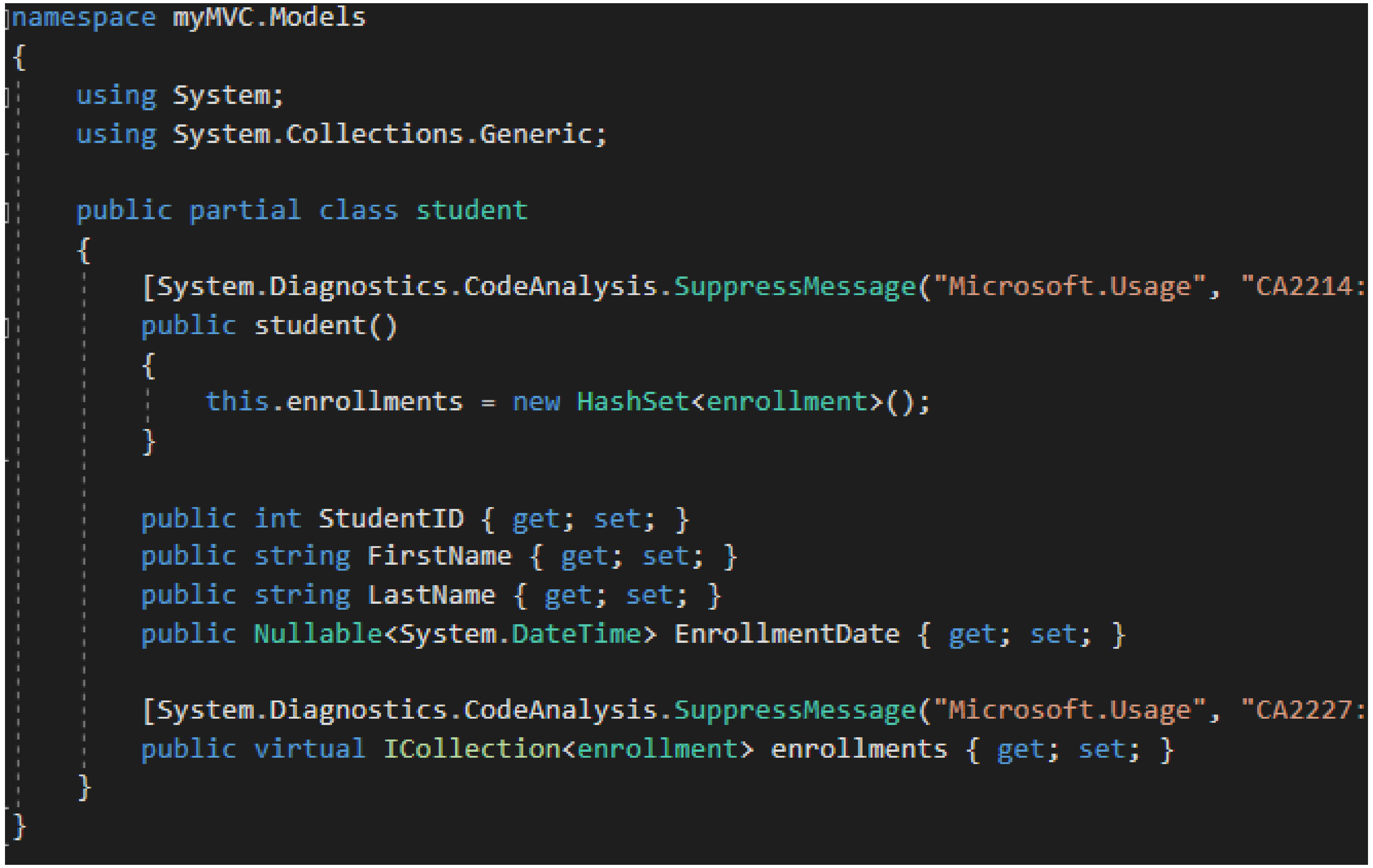
Insert, Update, Delete with EF

You will see the Model View created.



Insert, Update, Delete with EF

The above operations would automatically create a Model file for all the database entities. For example, the Student table that we created will result in a Model file “Student.cs” with the following code:



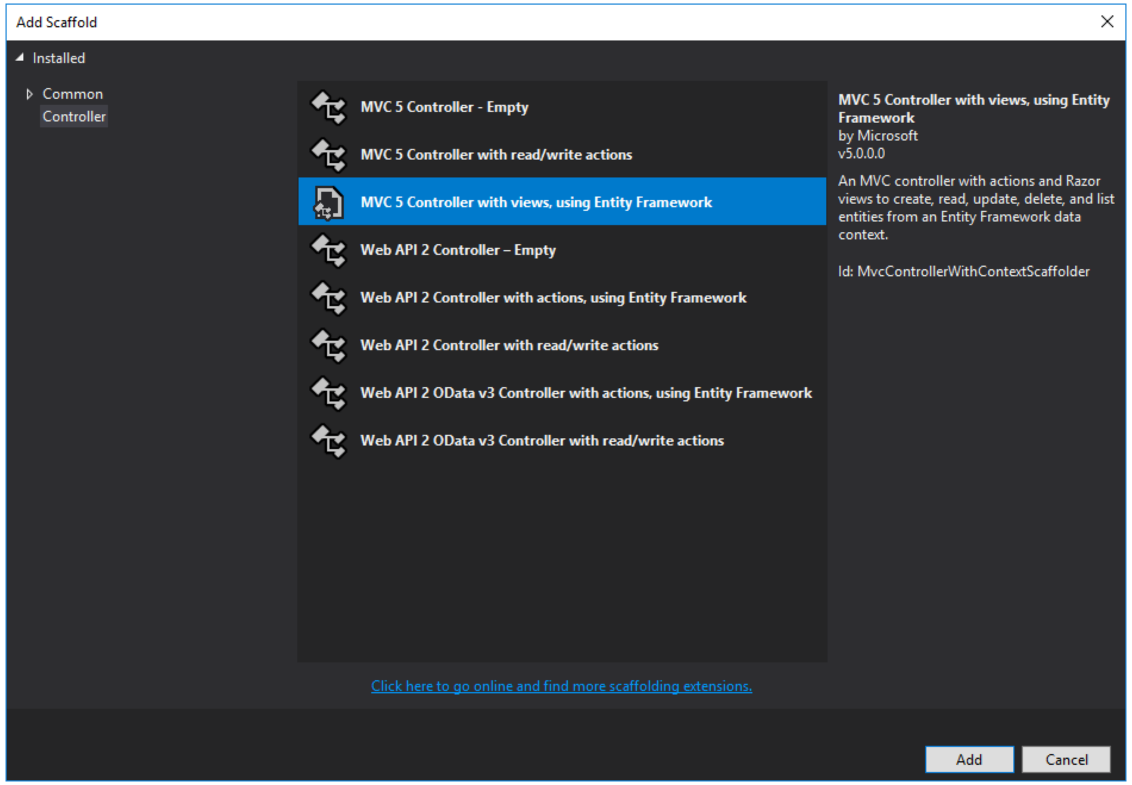
```
namespace myMVC.Models
{
    using System;
    using System.Collections.Generic;

    public partial class student
    {
        [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:
        public student()
        {
            this.enrollments = new HashSet<enrollment>();
        }

        public int StudentID { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public Nullable<System.DateTime> EnrollmentDate { get; set; }

        [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:
        public virtual ICollection<enrollment> enrollments { get; set; }
    }
}
```

Insert, Update, Delete with EF

Create a Controller "StudentController.cs"

Right Click on Controllers *Add → Controller*

Create MVC 5 Controller with views, using Entity Framework.



Add Scaffold

- Installed
  - Common
  - Controller

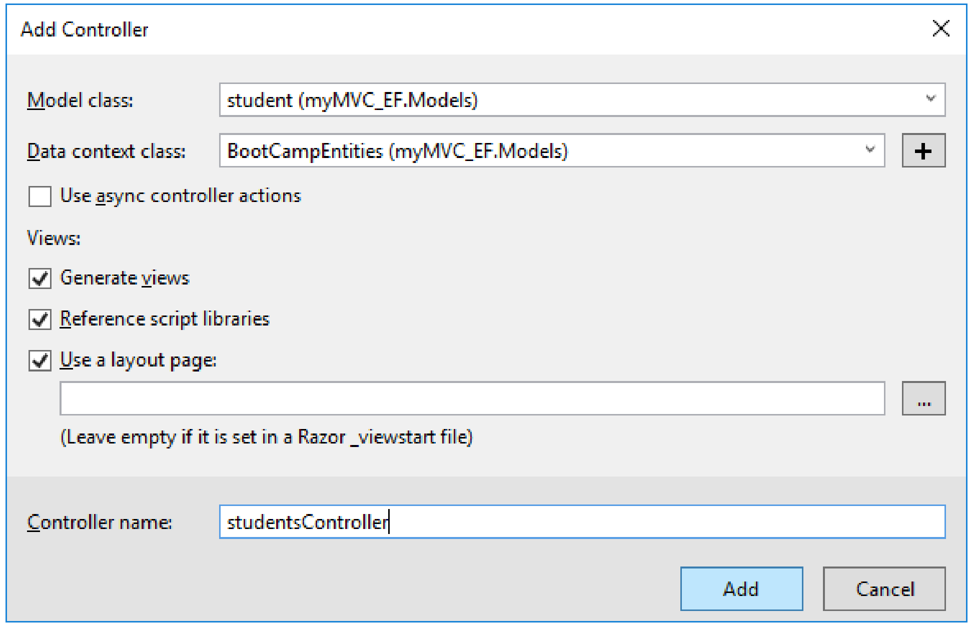
- MVC 5 Controller - Empty
- MVC 5 Controller with read/write actions
- MVC 5 Controller with views, using Entity Framework
- Web API 2 Controller – Empty
- Web API 2 Controller with actions, using Entity Framework
- Web API 2 Controller with read/write actions
- Web API 2 OData v3 Controller with actions, using Entity Framework
- Web API 2 OData v3 Controller with read/write actions

MVC 5 Controller with views, using Entity Framework
by Microsoft
v5.0.0.0

An MVC controller with actions and Razor views to create, read, update, delete, and list entities from an Entity Framework data context.

Id: MvcControllerWithContextScaffolder

Click here to go online and find more scaffolding extensions.



Add Controller

Model class: student (myMVC_EF.Models)

Data context class: BootCampEntities (myMVC_EF.Models)

☐ Use async controller actions

Views:

☑ Generate views

☑ Reference script libraries

☑ Use a layout page:

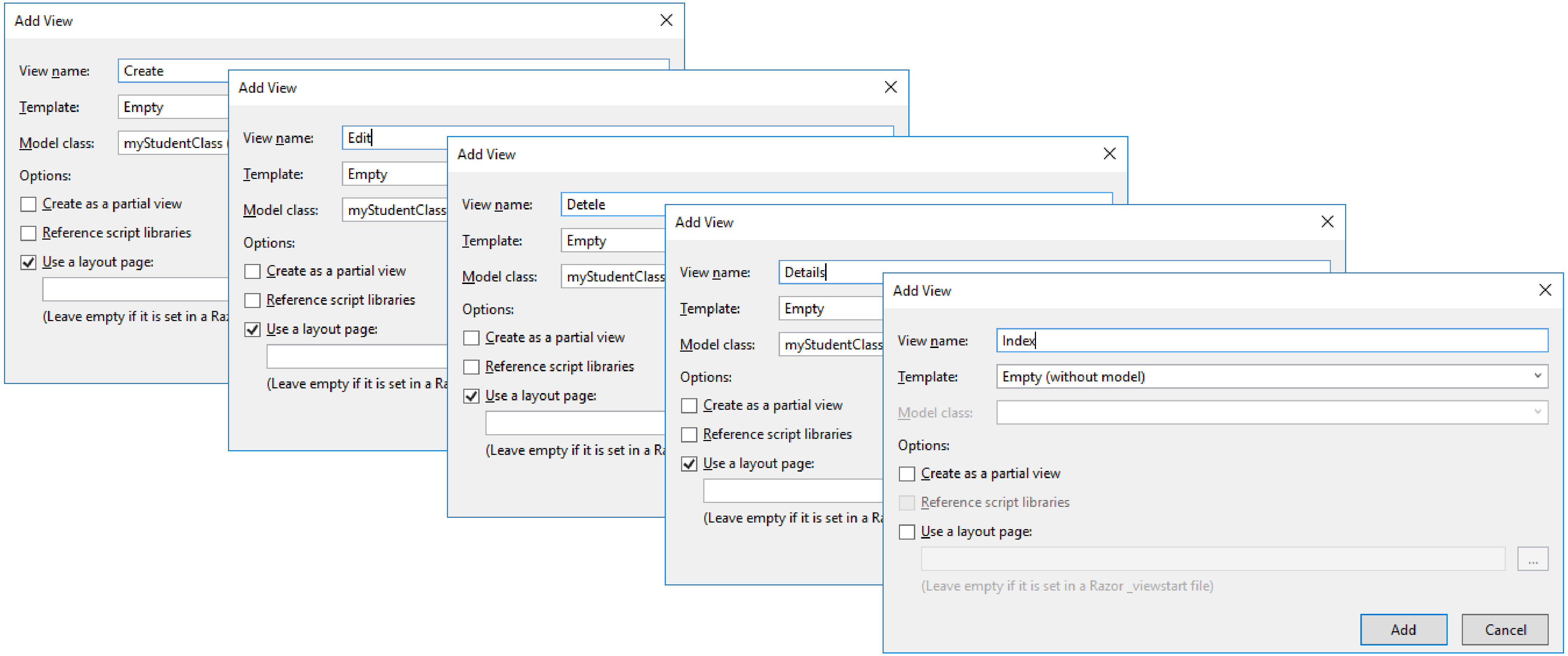
(Leave empty if it is set in a Razor _viewstart file)

Controller name: studentsController

Insert, Update, Delete with EF

Add View Dialog box will appear. Fill details as in this picture and click to Add button.

Using the same process Add View for Index(), Create(), Edit(), Delete() and Details() action method.



Add View

View name: Create

Template: Empty

Model class: myStudentClass

Options:

- [ ] Create as a partial view
- [ ] Reference script libraries
- [x] Use a layout page:

(Leave empty if it is set in a Ra

Add View

View name: Edit

Template: Empty

Model class: myStudentClass

Options:

- [ ] Create as a partial view
- [ ] Reference script libraries
- [x] Use a layout page:

(Leave empty if it is set in a Ra

Add View

View name: Detele

Template: Empty

Model class: myStudentClass

Options:

- [ ] Create as a partial view
- [ ] Reference script libraries
- [x] Use a layout page:

(Leave empty if it is set in a Ra

Add View

View name: Details

Template: Empty

Model class: myStudentClass

Options:

- [ ] Create as a partial view
- [ ] Reference script libraries
- [x] Use a layout page:

(Leave empty if it is set in a Ra

Add View

View name: Index

Template: Empty (without model)

Model class:

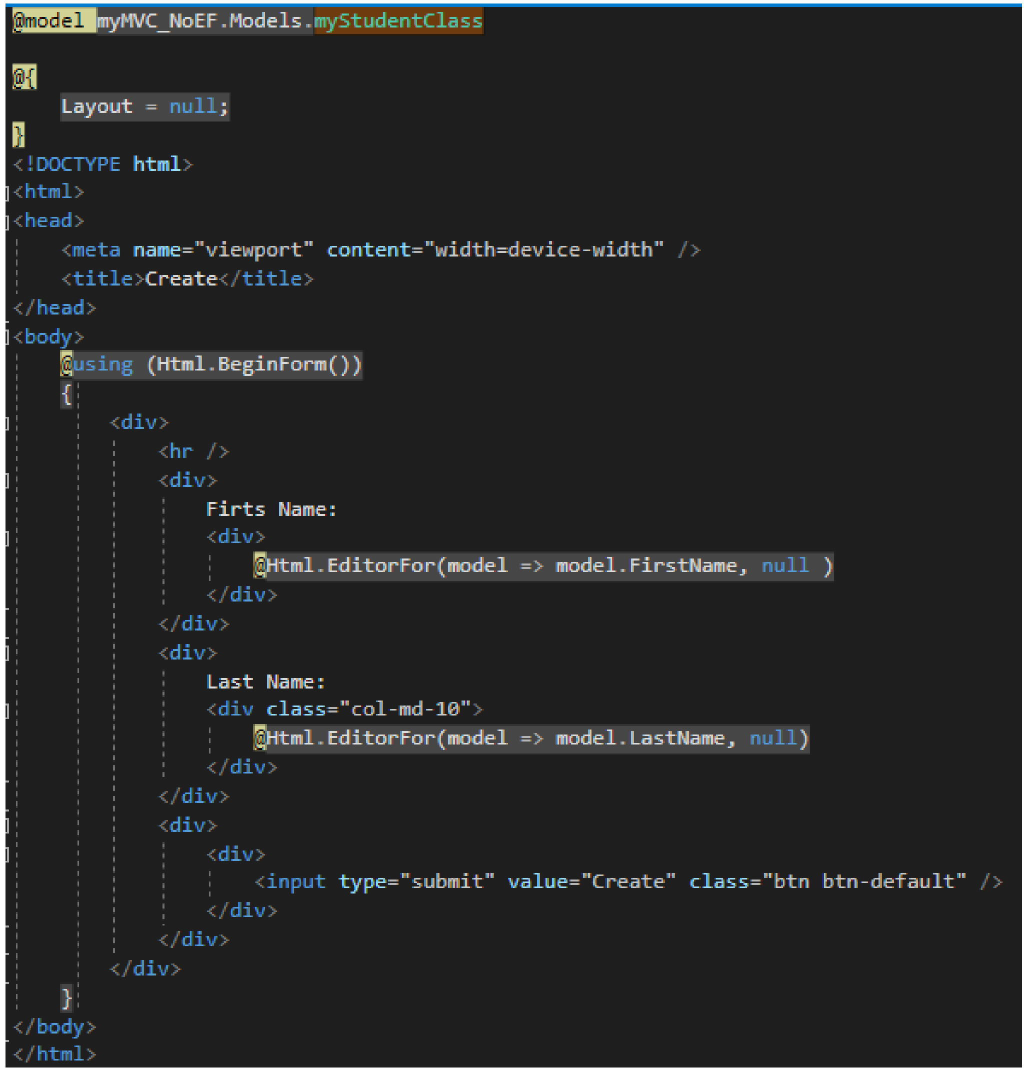
Options:

- [ ] Create as a partial view
- [ ] Reference script libraries
- [ ] Use a layout page:

(Leave empty if it is set in a Razor _viewstart file)

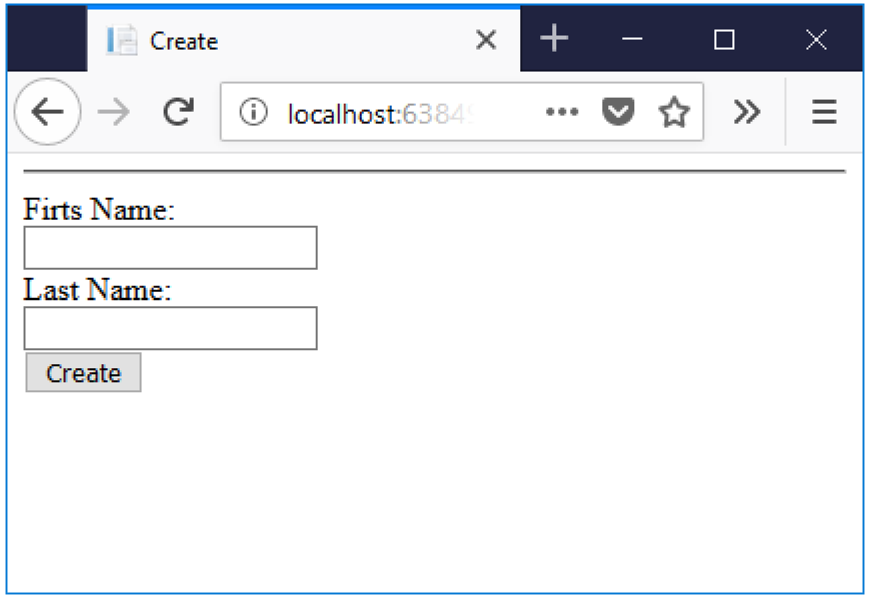
Insert, Update, Delete with EF

Now Add following code in "Create.cshtml"



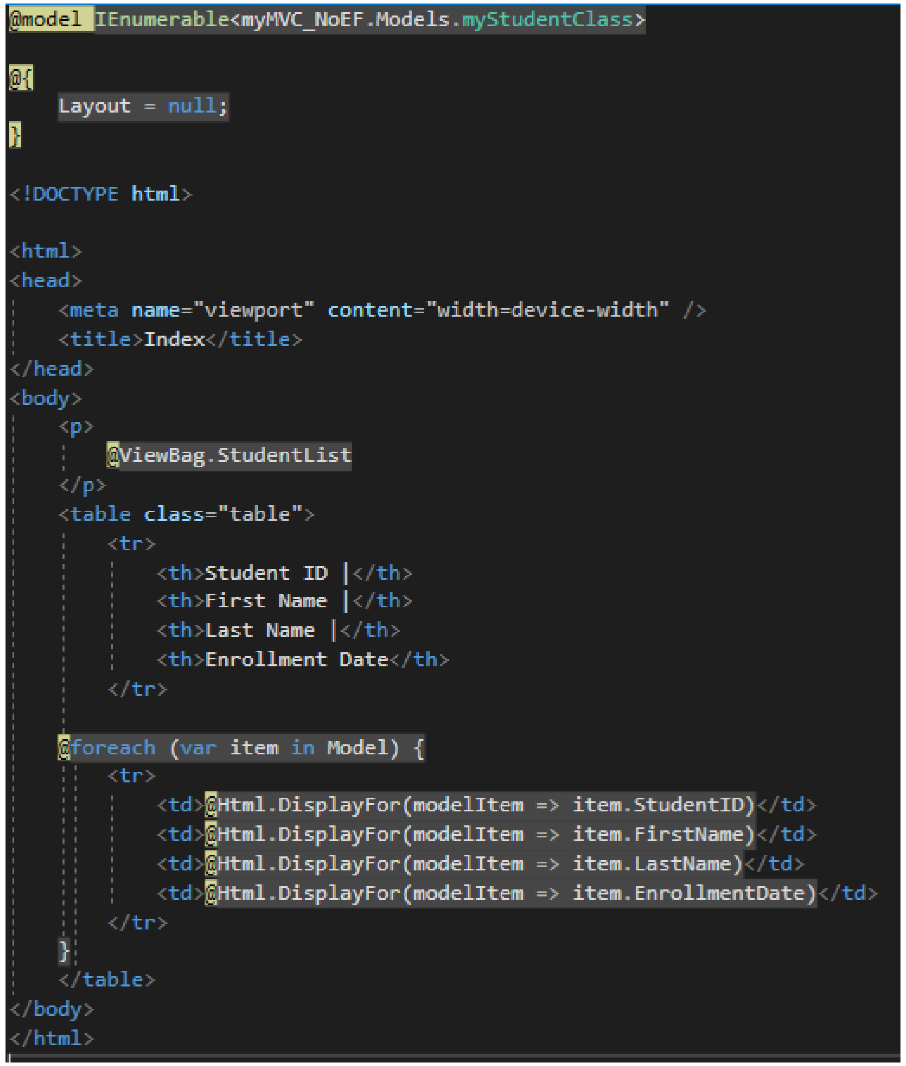
```
@model myMVC_NoEF.Models.myStudentClass

@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Create</title>
</head>
<body>
    @using (Html.BeginForm())
    {
        <div>
            <hr />
            <div>
                Firts Name:
                <div>
                    @Html.EditorFor(model => model.FirstName, null )
                </div>
            </div>
            <div>
                Last Name:
                <div class="col-md-10">
                    @Html.EditorFor(model => model.LastName, null)
                </div>
            </div>
            <div>
                <div>
                    <input type="submit" value="Create" class="btn btn-default" />
                </div>
            </div>
        </div>
    }
</body>
</html>
```

Press F5 to Run your project and navigate to *__/Student/Create__*



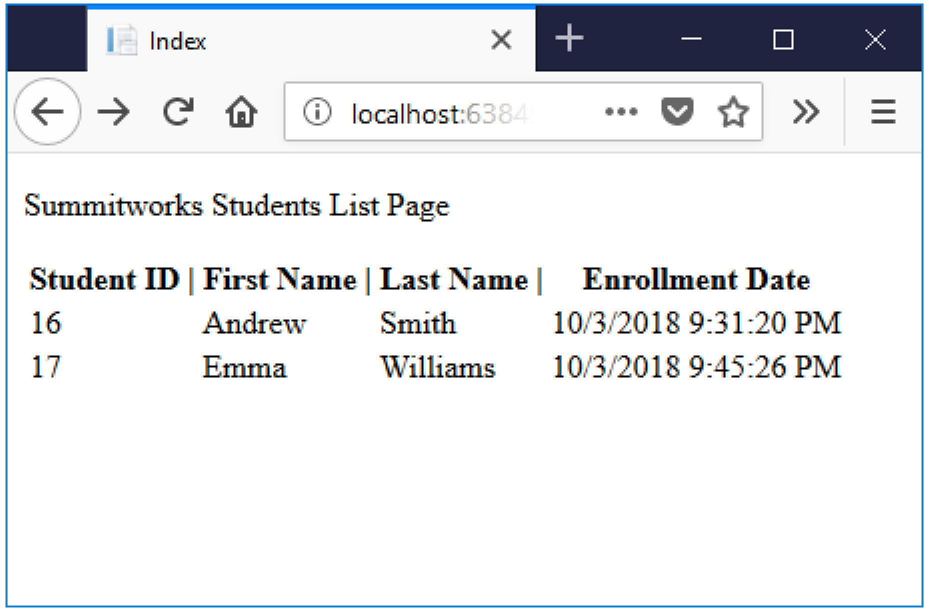
Insert, Update, Delete with EF

Now Add following code in “Index.cshtml”



```
@model IEnumerable<myMVC_NoEF.Models.myStudentClass>

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <p>
        @ViewBag.StudentList
    </p>
    <table class="table">
        <tr>
            <th>Student ID |</th>
            <th>First Name |</th>
            <th>Last Name |</th>
            <th>Enrollment Date</th>
        </tr>

    @foreach (var item in Model) {
        <tr>
            <td>@Html.DisplayFor(modelItem => item.StudentID)</td>
            <td>@Html.DisplayFor(modelItem => item.FirstName)</td>
            <td>@Html.DisplayFor(modelItem => item.LastName)</td>
            <td>@Html.DisplayFor(modelItem => item.EnrollmentDate)</td>
        </tr>
    }
    </table>
</body>
</html>
```

Press F5 to Run your project and navigate to *__/Student/Index__*

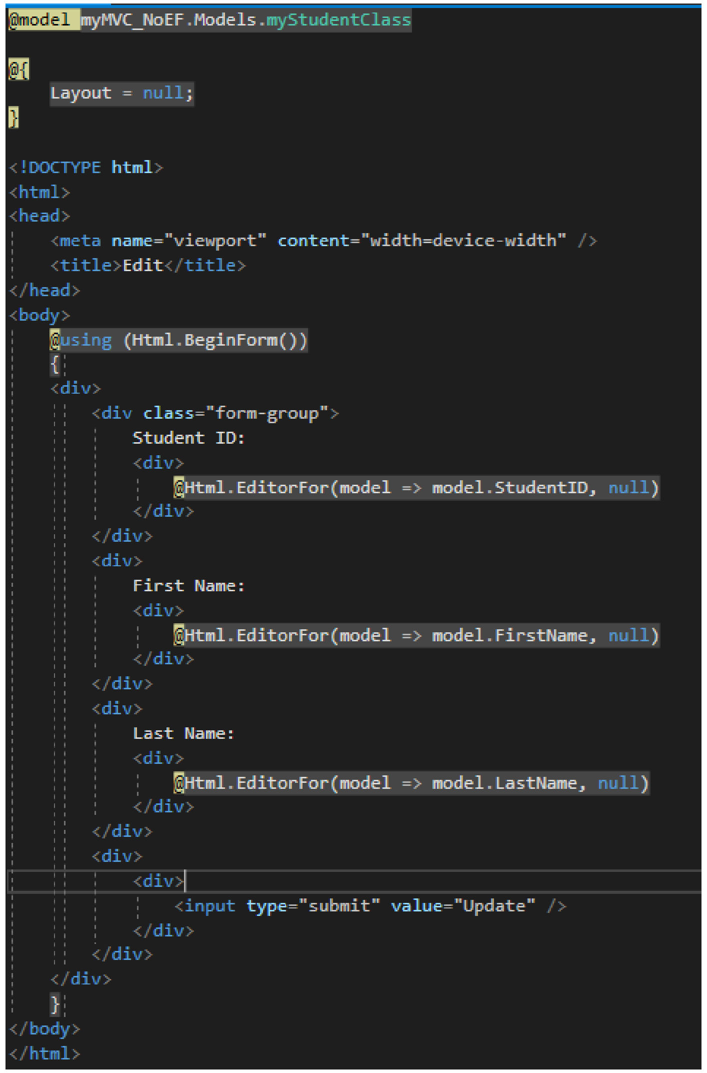


Summitworks Students List Page

Student ID |	First Name |	Last Name |	Enrollment Date
16	Andrew	Smith	10/3/2018 9:31:20 PM
17	Emma	Williams	10/3/2018 9:45:26 PM

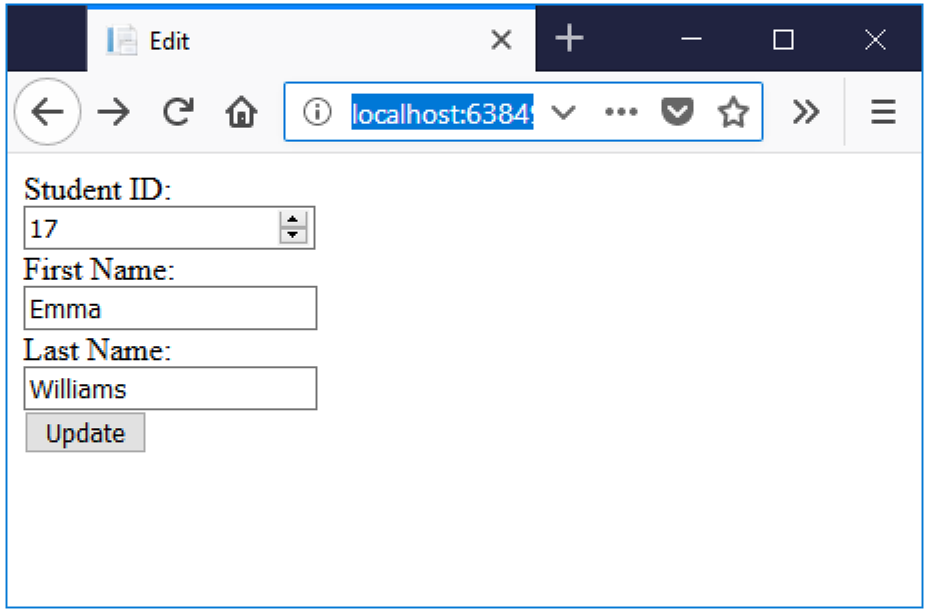
Insert, Update, Delete with EF

Now Add following code in “Update.cshtml”



```
@model myMVC_NoEF.Models.myStudentClass

@{
    Layout = null;
}

<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Edit</title>
</head>
<body>
    @using (Html.BeginForm())
    {
    <div>
        <div class="form-group">
            Student ID:
            <div>
                @Html.EditorFor(model => model.StudentID, null)
            </div>
        </div>
        <div>
            First Name:
            <div>
                @Html.EditorFor(model => model.FirstName, null)
            </div>
        </div>
        <div>
            Last Name:
            <div>
                @Html.EditorFor(model => model.LastName, null)
            </div>
        </div>
        <div>
            <div>
                <input type="submit" value="Update" />
            </div>
        </div>
    </div>
    }
</body>
</html>
```

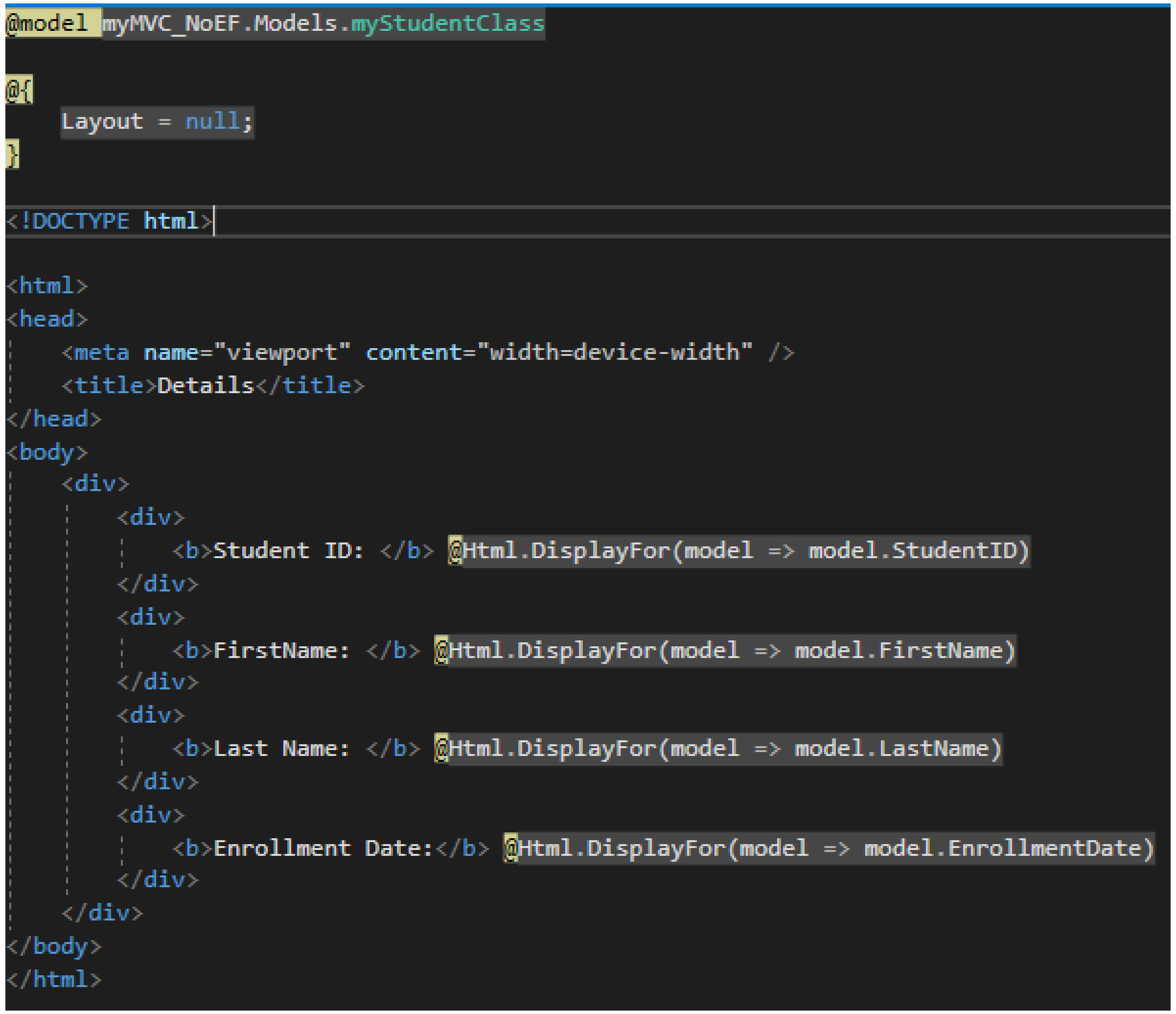
Press F5 to Run your project and navigate to *<span>/Student/Edit/{id}</span>*



Student ID:
17
First Name:
Emma
Last Name:
Williams
Update

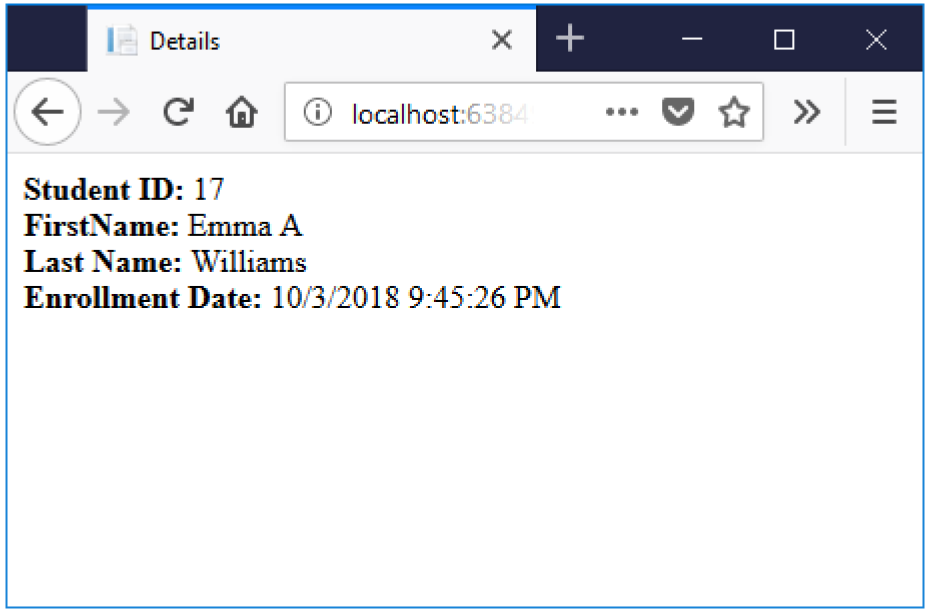
Insert, Update, Delete with EF

Now Add following code in “Details.cshtml”



```
@model myMVC_NoEF.Models.myStudentClass

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Details</title>
</head>
<body>
    <div>
        <div>
            <b>Student ID: </b> @Html.DisplayFor(model => model.StudentID)
        </div>
        <div>
            <b>FirstName: </b> @Html.DisplayFor(model => model.FirstName)
        </div>
        <div>
            <b>Last Name: </b> @Html.DisplayFor(model => model.LastName)
        </div>
        <div>
            <b>Enrollment Date:</b> @Html.DisplayFor(model => model.EnrollmentDate)
        </div>
    </div>
</body>
</html>
```

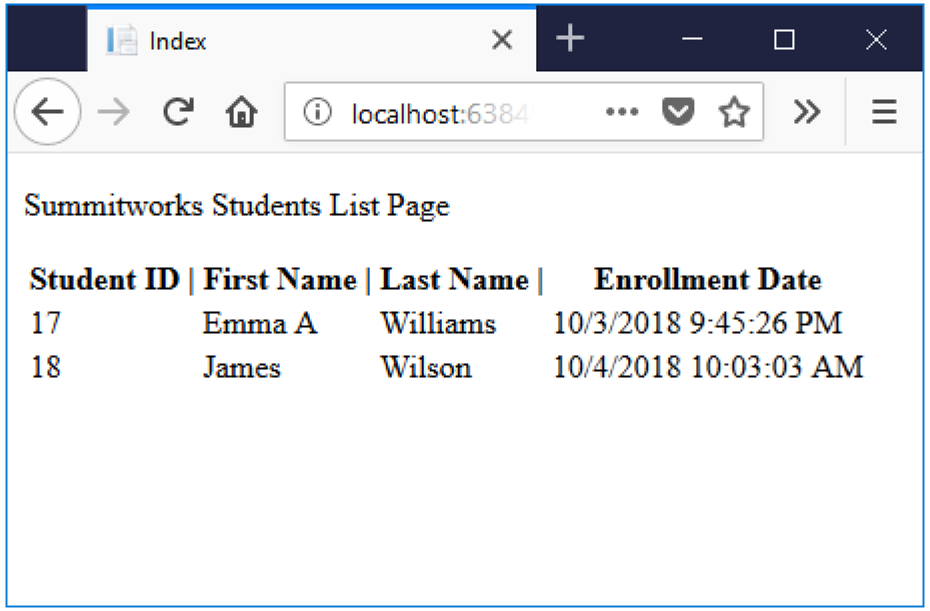
Press F5 to Run your project and navigate to
/Student/Details/{id}



Student ID: 17
FirstName: Emma A
Last Name: Williams
Enrollment Date: 10/3/2018 9:45:26 PM

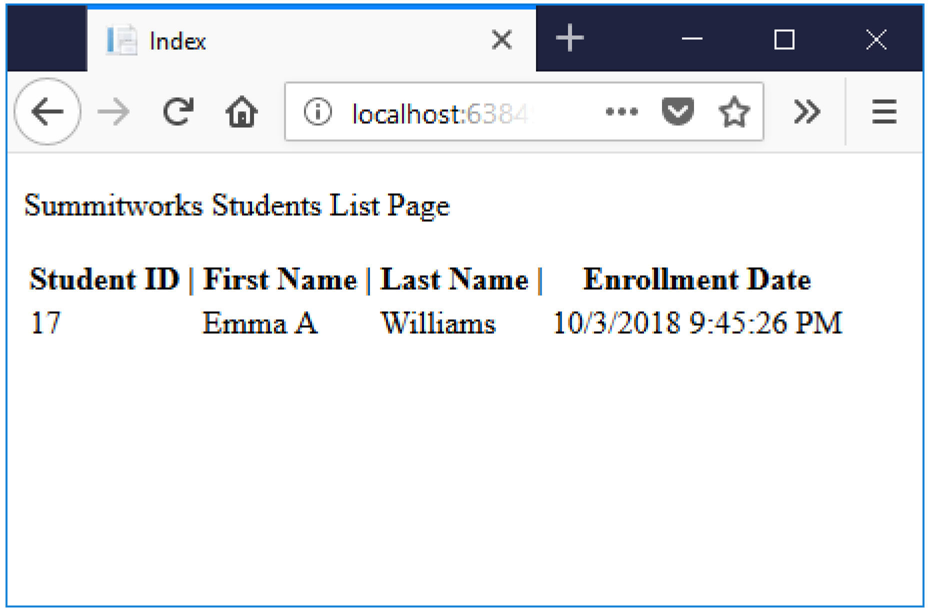
Insert, Update, Delete with EF

To delete a Student record navigate to ***/Student/Delete/{id}***



Summitworks Students List Page

```
Student ID | First Name | Last Name |    Enrollment Date
17           Emma A       Williams      10/3/2018 9:45:26 PM
18           James        Wilson        10/4/2018 10:03:03 AM
```



Summitworks Students List Page

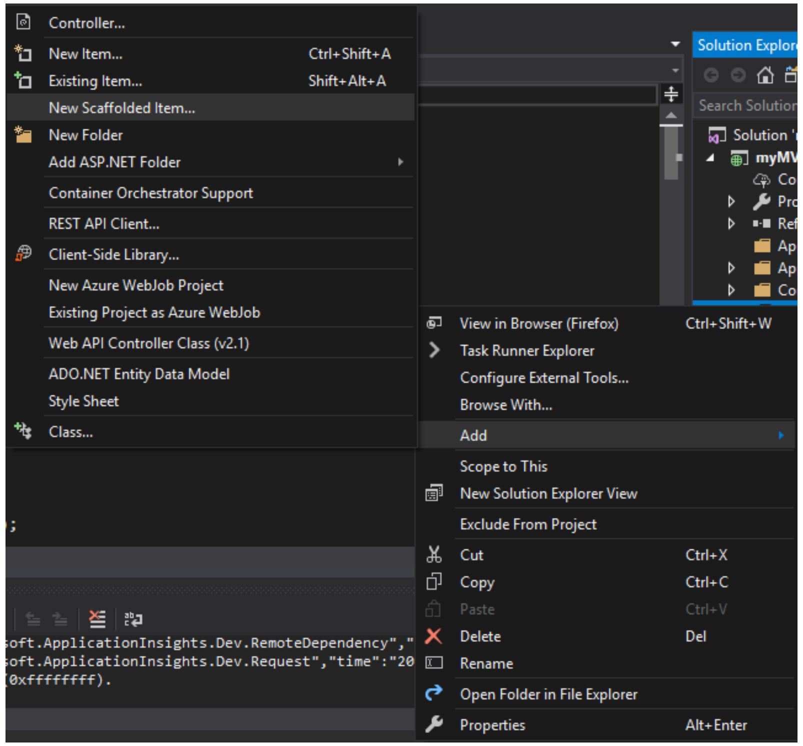
```
Student ID | First Name | Last Name |    Enrollment Date
17           Emma A       Williams      10/3/2018 9:45:26 PM
```

Scaffolding

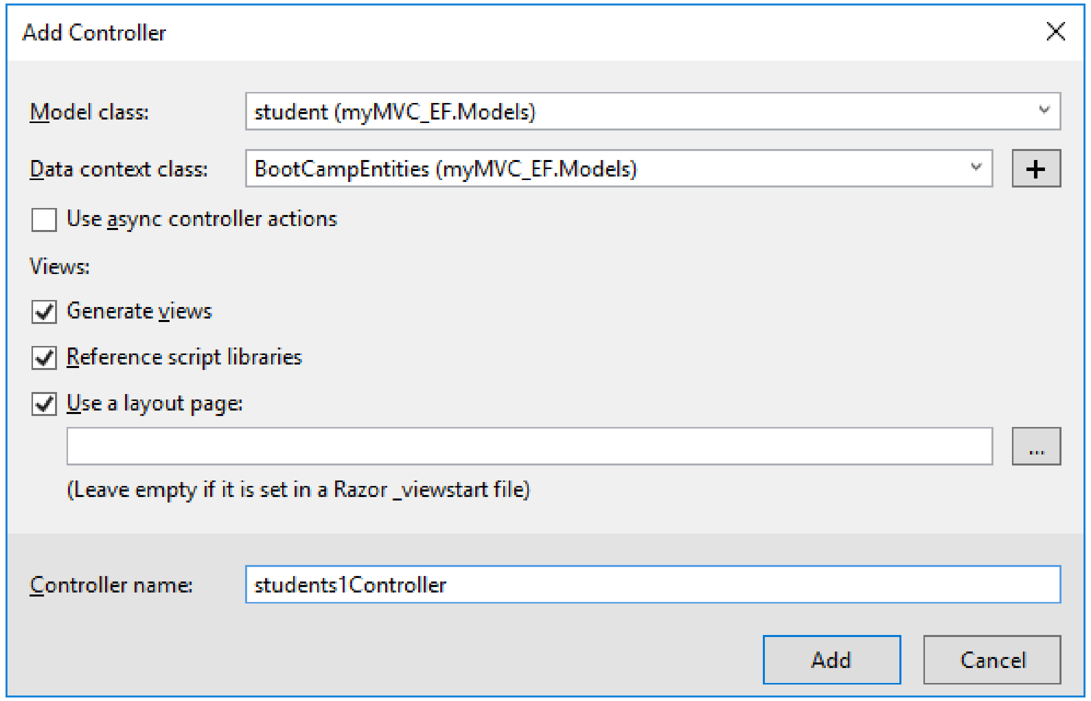
Scaffolding

- Scaffolding is a code generation framework that automatically adds codes and **creates view pages and controllers** for your project.

- **Scaffolding** makes developer job easier by creating controllers and view pages for the data model. It generates codes and pages for CRUD(Create, Read, Update and Delete) Operation.



```
soft.ApplicationInsights.Dev.RemoteDependency","
soft.ApplicationInsights.Dev.Request","time":"20
0xffffffff).
```


Add Controller

Model class: student (myMVC_EF.Models)

Data context class: BootCampEntities (myMVC_EF.Models) +

[ ] Use async controller actions

Views:

[✓] Generate views

[✓] Reference script libraries

[✓] Use a layout page:

...

(Leave empty if it is set in a Razor _viewstart file)

Controller name: students1Controller

Add Cancel