

---

# ASP.NET MVC 5

---

by Harsha Vardhan (.NET Expert)



## Contents

<b>Chapter 1: Entity Framework .....</b>	<b>10</b>
<b>Fundamentals of Entity Framework .....</b>	<b>10</b>
Introduction to Entity Framework .....	10
Basics of Entity Framework.....	11
Steps to Prepare First Example in EF .....	13
<b>Query Operations in EF .....</b>	<b>17</b>
ToList .....	17
Where .....	17
Where - Example.....	17
OrderBy .....	18
OrderBy - Example .....	19
OrderByDescending.....	20
OrderByDescending - Example.....	20
ThenBy.....	22
ThenBy - Example .....	22
Reverse.....	24
Reverse - Example .....	24
Select .....	25
Select - Example .....	25
Distinct .....	27
Distinct - Example .....	27
GroupBy.....	29
GroupBy - Example .....	29
Concat.....	31
Concat - Example .....	31
Union.....	33
Union - Example.....	33
Intersect .....	35
Intersect - Example .....	35
Except.....	36
Except - Example .....	37
Skip.....	38
Skip - Example.....	39
Take .....	40
Take - Example.....	40
First .....	42
First - Example .....	42
FirstOrDefault .....	43

FirstOrDefault - Example .....	43
Last.....	45
Last - Example.....	45
LastOrDefault.....	46
LastOrDefault - Example .....	46
Join .....	48
Join - Example.....	48
Aggregate Methods.....	50
Aggregate Methods - Example .....	50
Deferred Execution.....	52
Deferred Execution - Example.....	52
Non-Query Operations in EF .. . . . .	54
Insertion .....	54
Insertion - Example .....	55
Updation.....	56
Updation - Example.....	56
Deletion.....	58
Deletion - Example.....	58
Calling Stored Procedures using EF .....	60
Calling Stored Procedures using EF - Example .....	60
Stored Procedures with Parameters.....	61
Stored Procedures with Parameters - Example .....	62
Data Annotations .....	63
Data Annotations - Example .....	64
<b>Chapter 2 : ASP.NET MVC 5 .. . . . .</b>	<b>66</b>
<b>Fundamentals of MVC .. . . . .</b>	<b>66</b>
Introduction to ASP.NET MVC .....	66
Steps to Prepare First Example in MVC.....	69
<b>Controllers .. . . . .</b>	<b>85</b>
What is Controller .....	85
Action Methods .....	86
<b>URL Routing .. . . . .</b>	<b>86</b>
What is URL Routing .....	86
Multiple Routes - Example .....	86
Generic URL Routing.....	92
Generic Routing - Example .....	92
Route Parameters .....	97
Route Parameters - Example .....	97
Optional Parameters.....	99
Optional Parameters - Example.....	99

Default Parameters.....	100
Default Parameters - Example.....	101
Constraints .....	102
Constraints - Example.....	102
HTTP .....	105
What is HTTP.....	105
The "Request" object.....	106
Request - Example.....	107
The "Response" object .....	108
Response - Example .....	109
ActionResult .....	110
ContentResult .....	111
ContentResult - Example.....	111
FileResult .....	112
FileResult - Example.....	112
RedirectToRouteResult .....	113
RedirectToRouteResult - Example .....	114
Server.Transfer.....	115
Server.Transfer - Example .....	115
Views .....	116
Views - Example.....	117
View Name .....	118
View Name - Example .....	118
Razor .....	119
What is View Engine.....	119
Razor - Code Blocks - Example.....	120
Razor - Expressions - Example .....	121
Razor - If - Example.....	122
Razor - For - Example .....	123
Razor - Foreach - Example .....	124
Razor – Literal - Example .....	126
Razor – Comments - Example.....	127
State Management .....	128
What is State Management.....	128
ViewData.....	128
ViewData - Example .....	128
Models .....	129
ViewData with Models - Example.....	130
ViewBag.....	131
ViewBag - Example.....	131
ViewBag with Collections - Example.....	133

TempData.....	134
TempData - Example.....	134
TempData – Third Result - Example .....	136
TempData – Keep - Example .....	138
TempData – Peek - Example.....	140
Session.....	142
Session - Example .....	142
Session – With Objects - Example.....	143
ViewData (vs) ViewBag (vs) TempData (vs) Session.....	145
Application.....	145
Application - Example.....	146
Cache .....	147
Cache - Example.....	147
Cookies.....	148
Cookies - Example .....	148
<b>Shared Views .....</b>	<b>149</b>
What is Shared View .....	149
Shared Views - Example.....	149
<b>Layout Views .....</b>	<b>151</b>
What is Layout View.....	151
Layout Views - Example.....	152
_ViewStart.cshtml .....	154
_ViewStart.cshtml - Example .....	154
Sections in Layout Views .....	157
Sections in Layout Views - Example .....	157
<b>Partial Views .....</b>	<b>160</b>
Partial Views.....	160
Partial Views - Example .....	160
<b>Bundling and Minification .....</b>	<b>162</b>
What is Bundling and Minification .....	162
Bundling and Minification - Example .....	162
<b>Filters .....</b>	<b>168</b>
OutputCache.....	169
Output Caching - Example.....	169
ActionName .....	170
ActionName - Example .....	170
NonAction .....	171
NonAction - Example .....	171
ChildActionOnly .....	172
ChildActionOnly - Example.....	173

<b>Custom Filters .....</b>	<b>174</b>
Introduction to Custom Filters .....	174
IAuthorizationFilter - Example .....	175
IActionFilter - Example .....	176
IResultFilter - Example .....	178
IExceptionFilter - Example .....	179
Global Filters.....	181
Global Action Filters - Example .....	181
<b>Exception Handling in MVC .....</b>	<b>185</b>
Try Catch .....	185
TryCatch - Example .....	186
OnException .....	187
OnException - Example.....	187
IExceptionFilter .....	189
IExceptionFilter - Example .....	189
HandleError.....	192
HandleError - Example.....	192
HTTP Errors .....	194
HTTP Errors - Example .....	194
Application_Error .....	196
Application_Error - Example .....	196
<b>Attribute Routing .....</b>	<b>198</b>
Intro to Attribute Routing.....	198
Attribute Routing - Example .....	198
RoutePrefix.....	201
RoutePrefix - Example .....	201
Route Parameters .....	204
Route Parameters - Example .....	204
<b>Areas .....</b>	<b>206</b>
Introduction to Areas.....	206
Areas - Example .....	207
<b>Integrations .....</b>	<b>209</b>
Npm Integration .....	209
Npm - Example.....	210
jQuery Integration.....	211
jQuery - Example .....	211
Bootstrap Integration.....	213
Bootstrap - Example .....	213
AngularJS Integration.....	215
AngularJS - Example .....	215

Angular 5 Integration .....	217
Angular 5 - Example .....	217
Gulp Integration .....	220
Gulp - Example .....	221
Grunt Integration .....	226
Grunt - Example .....	227
Bower Integration .....	230
Bower - Example .....	231
WCF Integration .....	232
WCF - Example .....	233
AutoMapper Integration .....	235
AutoMapper - Example .....	236
<b>Working with Forms .....</b>	<b>237</b>
Forms .....	237
Forms - Example .....	238
Action Parameters .....	239
Action Parameters - Example.....	240
Automatic Model Binding.....	242
Automatic Model Binding - Example.....	242
<b>Strongly Typed Views .....</b>	<b>244</b>
What is Strongly Typed View .....	244
Strongly Typed Views - Example .....	245
Strongly Typed Views With Collections.....	246
Strongly Typed Views With Collections - Example.....	246
View Model.....	248
View Models - Example.....	248
<b>Entity Framework .....</b>	<b>250</b>
Introduction to Entity Framework .....	250
Entity Framework - Example .....	251
Entity Framework – Search - Example.....	254
Entity Framework – Stored Procedure - Example.....	256
Entity Framework – Insert - Example .....	260
Entity Framework – Update - Example.....	262
Entity Framework – Delete - Example.....	264
Entity Framework – CRUD - Example .....	267
Code First Approach .....	272
Code First Approach - Example.....	272
<b>Scaffolding Templates .....</b>	<b>275</b>
What are Scaffolding Templates .....	275
Scaffolding Templates - Example .....	275
<b>Repository Pattern .....</b>	<b>286</b>

What is Repository Pattern .....	286
Repository - Example .....	287
Generic Repository.....	294
Repository - Example .....	295
<b>Dependency Injection .....</b>	<b>302</b>
What is Dependency Injection .....	302
Dependency Injection - Example .....	302
<b>AJAX .....</b>	<b>310</b>
Introduction to AJAX.....	310
jQuery AJAX.....	311
jQuery AJAX - Simple - Example .....	311
jQuery AJAX - Get - Example.....	313
jQuery AJAX - Search - Example .....	315
jQuery AJAX - Post - Example .....	318
jQuery AJAX - Put - Example.....	321
jQuery AJAX - Delete - Example .....	324
AngularJS AJAX.....	326
AngularJS AJAX - Simple - Example .....	327
AngularJS AJAX - Get - Example.....	329
AngularJS AJAX - Search - Example .....	332
AngularJS AJAX - Post - Example .....	335
AngularJS AJAX - Put - Example .....	338
AngularJS AJAX - Delete - Example .....	341
<b>Web API .....</b>	<b>343</b>
Introduction to Web API.....	343
Web API - jQuery AJAX - Simple – Example .....	344
Web API - jQuery AJAX - Get - Example .....	346
Web API - jQuery AJAX - Search - Example.....	349
Web API - jQuery AJAX - Post - Example .....	353
Web API - jQuery AJAX - Put - Example .....	356
Web API - jQuery AJAX - Delete - Example .....	360
Web API - AngularJS AJAX - Simple - Example.....	364
Web API - AngularJS AJAX - Get - Example .....	366
Web API - AngularJS AJAX - Search - Example.....	370
Web API - AngularJS AJAX - Post - Example .....	373
Web API - AngularJS AJAX - Put - Example .....	377
Web API - AngularJS AJAX - Delete - Example .....	380
<b>HTML Helpes and Validations .....</b>	<b>384</b>
Introduction to HTML Helpers.....	384
HTML Helpers - Example .....	385
Custom HTML Helpers - Example .....	388

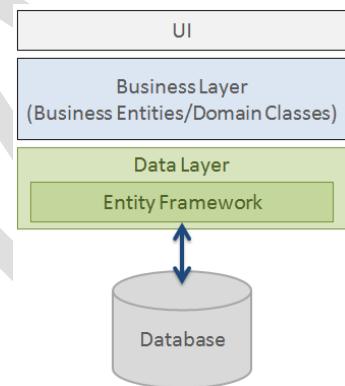
Custom HTML Helpers .....	391
<b>Validations .....</b>	<b>391</b>
Introduction to Validations.....	391
Custom Validations.....	392
Validations - Example .....	393
Client Side Validations - Example.....	398
<b>Security .....</b>	<b>404</b>
Forms Based Authentication.....	404
Forms Authentication - Example.....	404
CSRF.....	410
CSRF - Example .....	411
CORS .....	415
CORS – Example - Server Application.....	416
CORS – Example – Client Application.....	419
<b>Request Life Cycle .....</b>	<b>422</b>
<b>Unit Testing .....</b>	<b>423</b>
Basic Unit Testing - Example .....	423
View Name Testing - Example .....	424
View Bag Testing - Example .....	426
Model Testing - Example .....	428
Repository Testing - Example .....	431
<b>Deployment .....</b>	<b>434</b>
Deployment to IIS.....	434
Deployment into IIS - Example .....	434
<b>SignalR .....</b>	<b>442</b>
Introduction to SignalR.....	442
SignalR - Example .....	445
Method Invocation - Example .....	447
Group Chat - Example .....	449
Private Chat - Example .....	453
Method Returns - Example .....	457
HubName and HubMethodName - Example.....	459
Logging - Example .....	462

# Chapter 1: Entity Framework

## FUNDAMENTALS OF ENTITY FRAMEWORK

### Introduction to Entity Framework

- Entity Framework (EF) is a database technology, which is built based on ADO.NET, that supports ORM (Object-Relational Mapping) pattern, which is mostly preferred by ASP.NET MVC. "EF" is a part of .NET Framework, introduced in .NET 4.0. EF supports ORM (Object-Relational Mapping) pattern, which maps a database table with a model class. The table's columns are treated as properties in the model class. In this way, it bridges between "Relational world" to the "Object Oriented world".
- EF replaces manual coding in Data Layer. In other words EF is used in Data Layer, which directly communicates with database. On the top of Data Layer, we will develop Business Layer, which contains business logic that is specific to your application. On the top of Business Layer, we will develop UI Layer, which contains page design logic / presentation logic.



#### Features of Entity Framework

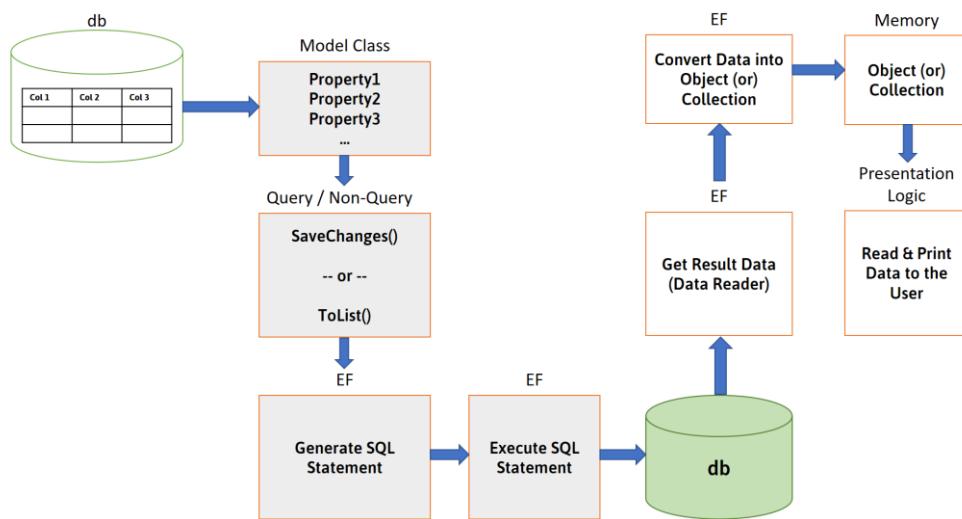
- Modeling:** EF Model classes represent structure of tables, so the developers can easily understand the fields and manipulate them in code.
- Query / Non-Query:** EF supports both querying (selecting data from database) and non-querying (inserting, updating, deleting) also.
- Transactions:** EF supports transactions, using which you can execute transactions. Transaction is a group of non-query operations. When one non-query operation is failed, all the previously executed non-query operations of the transaction will be cancelled automatically.
- Code First / Database First:** EF supports both "Code First" and "Database First" approaches. "Code First" approach means, the developer creates model class first; database will be automatically created at first time run. "Database First" approach means, the developer creates database tables first, model class needs to be created manually / automatically.

#### Versions of Entity Framework

Entity Framework Version	Year of Release	.NET Framework Version
Entity Framework 6	2013	.NET 4.6 with VS 2015 / VS 2017
Entity Framework 5	2012	.NET 4.0 with VS 2013
Entity Framework 4.3	2011	.NET 4.0 with VS 2012
Entity Framework 4.0	2010	.NET 4.0 with VS 2010
Entity Framework 3.5	2008	.NET 3.5 SP1 with VS 2008

## Basics of Entity Framework

### Work Flow in Entity Framework



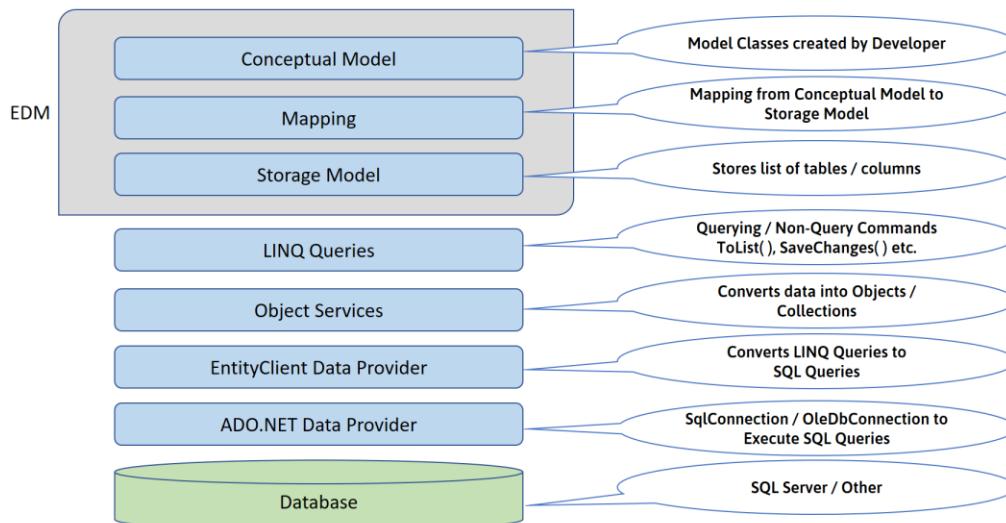
### Understanding Work Flow in EF

- Database Developer creates tables with columns and rows in the database.
- EF Developer creates a model class for each table. The model class reflects structure of the table, treating columns as properties.
- EF Developer calls a Query (ToList) or Non-Query (SaveChanges).
- EF converts the EF query / non-query into equivalent SQL statement.
- EF connects to database using ADO.NET internally and it executes the SQL statement at database.
- Database provides the result data in the form of DataReader.
- EF reads data from data reader and converts the data into Object or Collection.
- Developer reads and prints data from Object or Collection.

### Architecture of EF

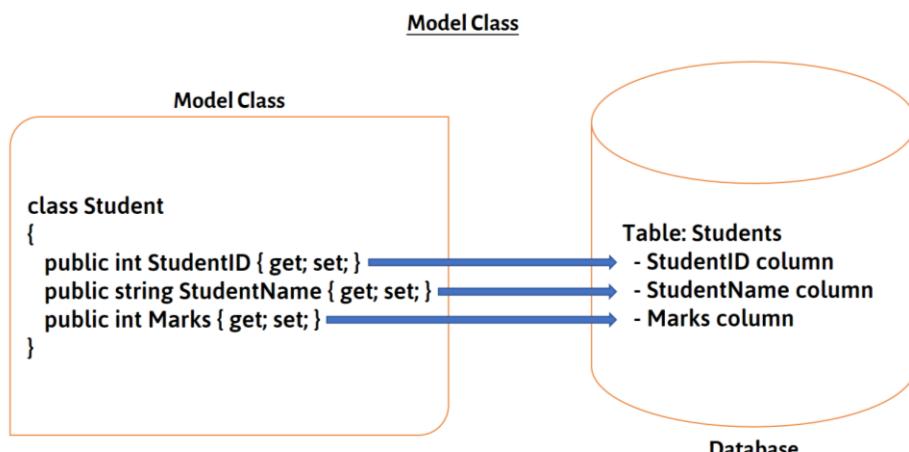
- **Conceptual Model:** It contains model classes created by developer. The class names / property names may differ from the table names / column names.
- **Storage Model:** It contains actual table names and column names.
- **Mapping:** Mapping between conceptual model and storage model.
- **LINQ Queries:** Contains actual LINQ queries like ToList(), SaveChanges(), Where, OrderBy() etc.
- **Object Services:** Converts data into objects / collections.
- **EntityClient Data Provider:** Converts LINQ Queries to equivalent SQL Queries.
- **ADO.NET Data Provider:** It is actual ADO.NET classes such SqlConnection, SqlCommand, SqlDataReader etc.
- **Database:** Stores actual tables.

### Architecture of Entity Framework



#### Model Class

- Model class reflects structure of the table.
- Table is treated as a class.
- Column is treated as a property.



#### Syntax of Model Class

```

using System.ComponentModel.DataAnnotations;
class modelclassname
{
    [Key]
    public datatype propertlename { get; set; }
    public datatype propertlename { get; set; }
    public datatype propertlename { get; set; }
    ...
}

```

#### DbContext Class

- DbContext class represents a collection of DbSet's.
- DbSet is a collection that represents a table. DbSet provides methods to retrieve, insert, update and delete data.

- The DbContext class should be child class of System.Data.Entity.DbContext class.
- The constructor of DbContext class will be called automatically when the developer creates an object for the DbContext class. The developer should initialize the constructor in the constructor of DbContext class, with "base" keyword.

### Syntax of DbContext

```
using System.Data.Entity;
class classname : DbContext
{
    public classname() : base(@"data source=servername; user id=sa; password=123; initial catalog=databasename")
    {
    }
    public DbSet<modelclassname> collectionname { get; set; }
    public DbSet<modelclassname> collectionname { get; set; }
    ...
}
```

### EntityFramework NuGet Package

- The complete entity framework is available as a NuGet package called "EntityFramework".
- It should be installed in the current project, in order to use EntityFramework in the current project.
- To install this package, go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console" and apply the following command:  
`install-package EntityFramework`
- When we install this package, Visual Studio automatically adds references to the following DLL files to the current project:
  - EntityFramework.dll
  - EntityFramework.SqlServer.dll
  - System.ComponentModel.DataAnnotations.dll

## Steps to Prepare First Example in EF

### Creating Database

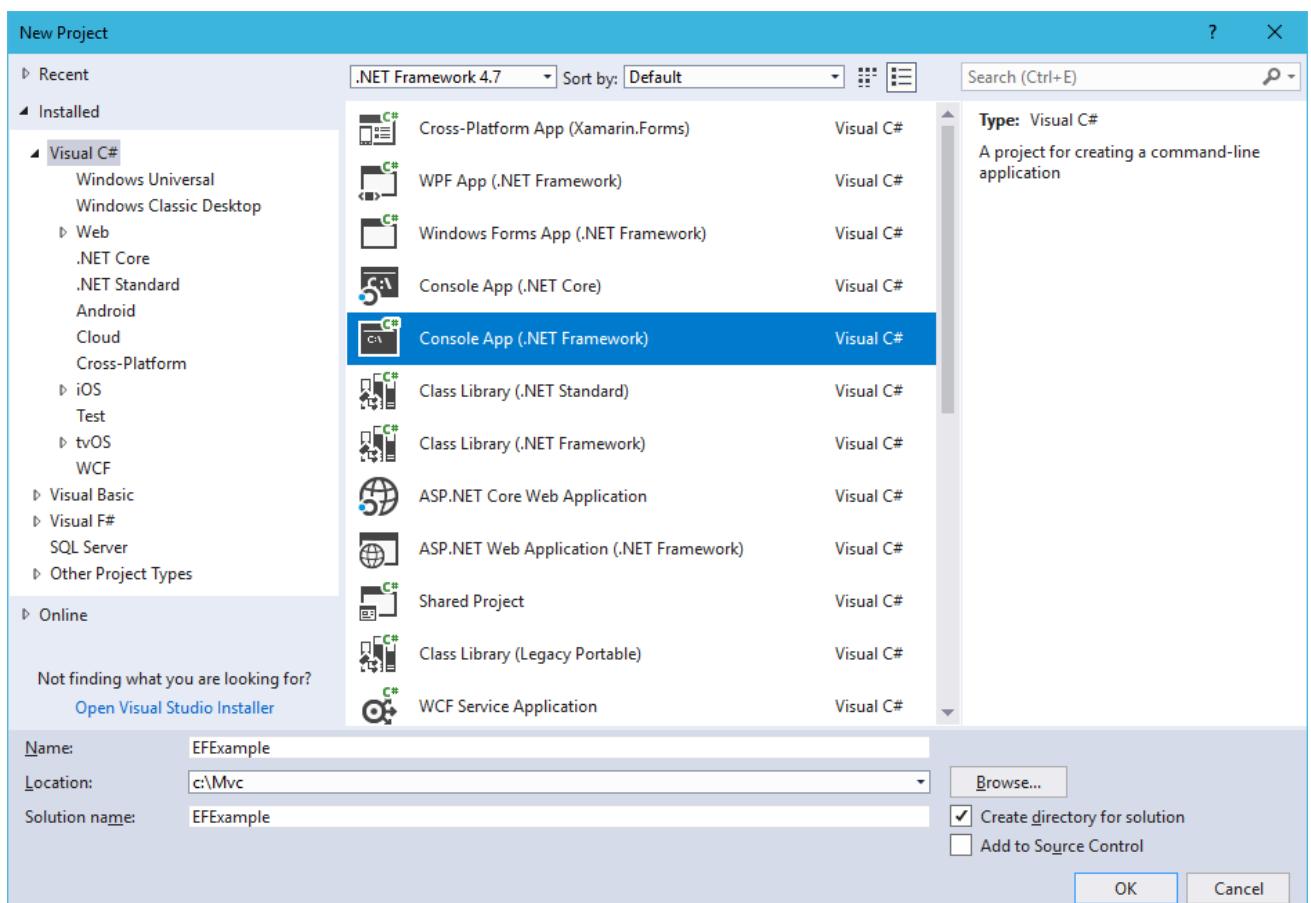
- Open SQL Server Management Studio. Select "Windows Authentication".
- Click on "Connect".
- Ignore this step if "company" database already exists.
- Click on "New Query".
- Type the following code:

```
create database company
go
use company
go
create table Employees(EmpID int primary key, EmpName nvarchar(max), Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### Creating Project

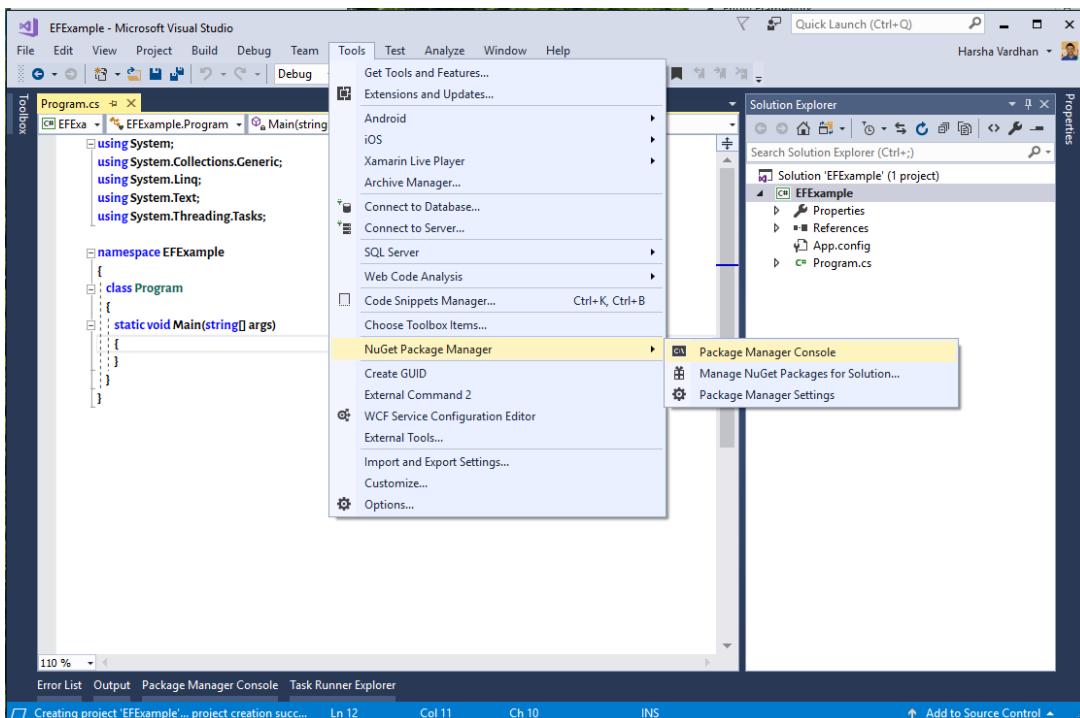
- Open Visual Studio 2017.
- Go to "File" - "New" - "Project".
- "New Project" dialogbox appears.



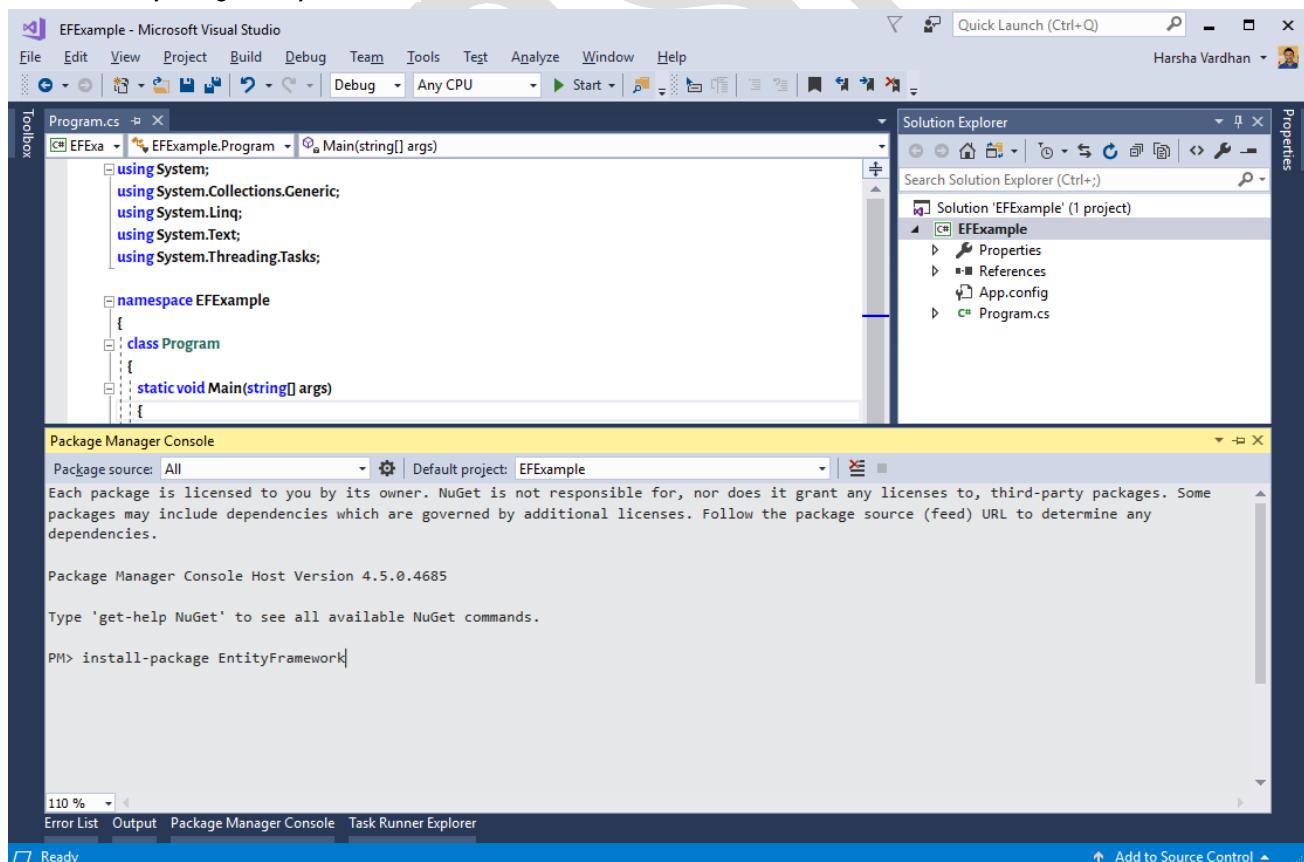
- Select ".NET Framework 4.7". Select "Visual C#". Select "Console App (.NET Framework)". Type the project name "EFExample". Type the location as "C:\Mvc". Type the solution name as "EFExample". Click on OK.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".



- "Package Manager Console" window will be opened.
- Type the following command in "Package Manager Console" and press Enter.  
`install-package EntityFramework`



- The following code will be automatically generated for "packages.config".

**Code for "packages.config" (Automatically generated)**

```
<?xml version="1.0" encoding="utf-8"?>
<packages>
<package id="EntityFramework" version="6.2.0" targetFramework="net47" />
</packages>
```

**Code for "Program.cs"**

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

namespace ToListExample
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }

        public DbSet<Employee> Employees { get; set; }
    }

    class Program
    {
        static void Main()
        {
            CompanyDbContext db = new CompanyDbContext();
            List<Employee> emps = db.Employees.ToList();
            foreach (Employee emp in emps)
            {
                Console.Write(emp.EmpID);
                Console.Write(",");
                Console.Write(emp.EmpName);
                Console.Write(",");
                Console.Write(emp.Salary);
                Console.WriteLine();
            }
            Console.ReadKey();
        }
    }
}
```

**Compile the Project**

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B". The project will be compiled.

**Run the Project**

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".

Output:

```
c:\Mvc\EFExample\EFExample\bin\Debug\EFExample.exe
1,Scott,4000
2,Allen,2500
3,Jones,5200
4,James,4400
5,Smith,7600
```

### QUERY OPERATIONS IN EF

#### ToList

- The `ToList()` method is used to select all columns and all rows of the table. It executes the query immediately and returns corresponding data as a "collection of objects of model class".
- Syntax:** `DbContextclass referencevariable = new DbContextclass();  
referencevariable.Dbsetname.ToList();`

#### Where

- The `Where()` method is used to select all columns and specific rows of the table. It retrieves the rows that are matching with specified condition. The `Where()` method receives a lambda expression as argument, which contains an argument (`temp`) that represents a row of the table, checks the condition and returns true / false. If true is returned, the row will be taken into the result; otherwise the row will not be taken into the result.
- Syntax:** `DbContextclass referencevariable = new DbContextclass();  
referencevariable.Dbsetname.Where(temp => condition).ToList();`

#### Where - Example

##### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```
create database company
go
use company
go
create table Employees(EmpID int primary key, EmpName nvarchar(max), Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

##### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "Console App (.NET Framework)". Type the project name "WhereExample". Type the location as "C:\mvc". Type the solution name as "WhereExample". Click on OK.

##### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.  
`install-package EntityFramework`

##### Code for "Program.cs"

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
```

```

using System.Collections.Generic;
using System.Linq;
namespace WhereExample
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }
    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }
        public DbSet<Employee> Employees { get; set; }
    }
    class Program
    {
        static void Main()
        {
            CompanyDbContext db = new CompanyDbContext();
            List<Employee> emps = db.Employees.Where(temp => temp.Salary > 7000).ToList();
            foreach (Employee emp in emps)
            {
                Console.Write(emp.EmpID);
                Console.Write(",");
                Console.Write(emp.EmpName);
                Console.Write(",");
                Console.Write(emp.Salary);
                Console.WriteLine();
            }
            Console.ReadKey();
        }
    }
}

```

#### Compile the Project

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B". The project will be compiled.

#### Run the Project

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".

#### Output:

```
C:\mvc\WhereExample\WhereExample\bin\Debug\WhereExample.exe
5,Smith,7600
```

## OrderBy

- The OrderBy() method is used to select sort rows based on the specified column. It performs sorting in ascending order. The OrderBy() method receives a lambda expression as argument, which contains an argument (temp) that represents a row of the table, that returns a column (property), based on which the data has to be sorted.
- Syntax:** `DbContextclass referencevariable = new DbContextclass();  
referencevariable.Dbsetname.OrderBy(temp => temp.columnname).ToList();`

## OrderBy - Example

### **Creating Database**

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```
create database company
go
use company
go
create table Employees(EmpID int primary key, EmpName nvarchar(max), Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### **Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "Console App (.NET Framework)". Type the project name "OrderByExample". Type the location as "C:\mvc". Type the solution name as "OrderByExample". Click on OK.

### **Installing Packages**

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
```

### **Code for "Program.cs"**

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
using System.Collections.Generic;
using System.Linq;
namespace OrderByExample
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }
        public DbSet<Employee> Employees { get; set; }
    }
    class Program
    {
        static void Main()
        {
            CompanyDbContext db = new CompanyDbContext();
            List<Employee> emps = db.Employees.OrderBy(temp => temp.Salary).ToList();
            foreach (Employee emp in emps)
```

```
{  
    Console.WriteLine(emp.EmpID);  
    Console.Write(",");  
    Console.WriteLine(emp.EmpName);  
    Console.Write(",");  
    Console.WriteLine(emp.Salary);  
    Console.WriteLine();  
}  
Console.ReadKey();  
}  
}  
}
```

### Compile the Project

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B". The project will be compiled.

### Run the Project

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".

### Output:



```
C:\mvc\OrderByExample\OrderByExample\bin\Debug\OrderByExample.exe  
2,Allen,2500  
1,Scott,4000  
4,James,4400  
3,Jones,5200  
5,Smith,7600
```

## OrderByDescending

- The OrderByDescending() method is used to select sort rows based on the specified column. It performs sorting in descending order. The OrderByDescending() method receives a lambda expression as argument, which contains an argument (temp) that represents a row of the table, that returns a column (property), based on which the data has to be sorted.
- **Syntax:** DbContextclass referencevariable = new DbContextclass();  
referencevariable.Dbsetname.OrderByDescending(temp => temp.columnname).ToList();

## OrderByDescending - Example

### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```
create database company  
go  
use company  
go  
create table Employees(EmpID int primary key, EmpName nvarchar(max), Salary decimal)  
go  
insert into Employees values(1, 'Scott', 4000)  
insert into Employees values(2, 'Allen', 2500)  
insert into Employees values(3, 'Jones', 5200)  
insert into Employees values(4, 'James', 4400)  
insert into Employees values(5, 'Smith', 7600)  
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "Console App (.NET Framework)". Type the project name "OrderByDescendingExample". Type the location as "C:\mvc". Type the solution name as "OrderByDescendingExample". Click on OK.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.  
`install-package EntityFramework`

### Code for "Program.cs"

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
using System.Collections.Generic;
using System.Linq;
namespace OrderByDescendingExample
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }
        public DbSet<Employee> Employees { get; set; }
    }
    class Program
    {
        static void Main()
        {
            CompanyDbContext db = new CompanyDbContext();
            List<Employee> emps = db.Employees.OrderBy(temp => temp.Salary).ToList();
            foreach (Employee emp in emps)
            {
                Console.Write(emp.EmpID);
                Console.Write(",");
                Console.Write(emp.EmpName);
                Console.Write(",");
                Console.Write(emp.Salary);
                Console.WriteLine();
            }
            Console.ReadKey();
        }
    }
}
```

### Compile the Project

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B". The project will be compiled.

### Run the Project

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".

Output:

```
C:\mvc\OrderByDescendingExample\OrderByDescendingExample\bin\Debug\OrderByDescendingExample.exe
5,Smith,6000
3,Jones,5200
4,James,4400
1,Scott,4000
2,Allen,2500
```

### ThenBy

- The ThenBy() method is used to specify secondary column name, whereas OrderBy() method is used to specify primary sorting column name. It performs sorting in ascending order. The ThenBy() method receives a lambda expression as argument, which contains an argument (temp) that represents a row of the table, that returns a column (property), based on which the data has to be sorted.
- Syntax:** `DbContextclass referencevariable = new DbContextclass();  
referencevariable.Dbsetname.ThenBy(temp => temp.columnname).ToList();`

### ThenBy - Example

#### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "departmentsandemployees" database already exists.
- Click on "New Query". Type the following code:

```
create database departmentsandemployees
go
use departmentsandemployees
go
create table Departments(
DeptNo int primary key,
DeptName nvarchar(max),
Loc nvarchar(max))
go
create table Employees(
EmpID int primary key,
EmpName varchar(max),
Salary decimal,
DeptNo int references Departments(DeptNo))
go
insert into Departments values(10, 'Accounting', 'New York')
insert into Departments values(20, 'Operations', 'New Delhi')
insert into Departments values(30, 'Sales', 'New Jersey')
insert into Employees values(1, 'Scott', 2000, 10)
insert into Employees values(2, 'Allen', 6500, 10)
insert into Employees values(3, 'Jones', 4577, 20)
insert into Employees values(4, 'James', 2500, 20)
insert into Employees values(5, 'Smith', 3345, 30)
insert into Employees values(6, 'Harry', 3600, 30)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialog box appears. Select ".NET Framework 4.7". Select "Visual C#". Select "Console App (.NET Framework)". Type the project name "ThenByExample". Type the location as "C:\mvc". Type the solution name as "ThenByExample". Click on OK.

#### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
```

**Code for "Program.cs"**

```

using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
using System.Collections.Generic;
using System.Linq;

namespace ThenByExample
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
        public int DeptNo { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=departmentsandemployees")
        {

        }

        public DbSet<Employee> Employees { get; set; }
    }

    class Program
    {
        static void Main()
        {
            CompanyDbContext db = new CompanyDbContext();
            List<Employee> emps = db.Employees.OrderBy(temp => temp.DeptNo).ThenBy(temp => temp.Salary).ToList();
            foreach (Employee emp in emps)
            {
                Console.Write(emp.EmpID);
                Console.Write(",");
                Console.Write(emp.EmpName);
                Console.Write(",");
                Console.Write(emp.Salary);
                Console.Write(",");
                Console.Write(emp.DeptNo);
                Console.WriteLine();
            }
            Console.ReadKey();
        }
    }
}

```

**Compile the Project**

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B". The project will be compiled.

**Run the Project**

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".

**Output:**

```
D:\OneDrive\01..NET\04. ASP.NET MVC 5\34. Entity Framework\06. ThenBy\ThenByExample\ThenByExample\bin\Debug\ThenByExample.exe
1,Scott,2000,10
2,Allen,6500,10
4,James,2500,20
3,Jones,4577,20
5,Smith,3345,30
6,Harry,3600,30
```

### Reverse

- The Reverse() method is used to reverse the rows.
- **Syntax:**

```
DbContextclass referencevariable = new DbContextclass();
List<modelclassname> collectionname = referencevariable.Dbsetname.ToList();
collectionname.Reverse();
```

### Reverse - Example

#### **Creating Database**

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```
create database company
go
use company
go
create table Employees(EmpID int primary key, EmpName nvarchar(max), Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

#### **Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "Console App (.NET Framework)". Type the project name "ReverseExample". Type the location as "C:\mvc". Type the solution name as "ReverseExample". Click on OK.

#### **Installing Packages**

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.  
`install-package EntityFramework`

#### **Code for "Program.cs"**

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
using System.Collections.Generic;
using System.Linq;
namespace ReverseExample
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }
}
```

```

public class CompanyDbContext : DbContext
{
    public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
    {
    }
    public DbSet<Employee> Employees { get; set; }
}
class Program
{
    static void Main()
    {
        CompanyDbContext db = new CompanyDbContext();
        List<Employee> emps = db.Employees.ToList();
        emps.Reverse();
        foreach (Employee emp in emps)
        {
            Console.Write(emp.EmpID);
            Console.Write(",");
            Console.Write(emp.EmpName);
            Console.Write(",");
            Console.Write(emp.Salary);
            Console.WriteLine();
        }
        Console.ReadKey();
    }
}

```

**Compile the Project**

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B".
- The project will be compiled.

**Run the Project**

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".

**Output:**

```

C:\mvc\ReverseExample\ReverseExample\bin\Debug\ReverseExample.exe
5,Smith,7600
4,James,4400
3,Jones,5200
2,Allen,2500
1,Scott,4000

```

**Select**

- The Select() method is used to retrieve the specified columns, instead of retrieving all columns. The Select() method receives a lambda expression as argument, which contains an argument (temp) that represents a row of the table, that returns a column or a set of columns that are to be retrieved.
- **Syntax:** `DbContextclass referencevariable = new DbContextclass();  
referencevariable.Dbsetname.Select(temp => temp.columnname).ToList();  
referencevariable.Dbsetname.Select(temp => new { temp.columnname1, temp.columnname2, ... }).ToList();`

**Select - Example****Creating Database**

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```
create database company
go
use company
go
create table Employees(EmpID int primary key, EmpName nvarchar(max), Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "Console App (.NET Framework)". Type the project name "SelectExample". Type the location as "C:\mvc". Type the solution name as "SelectExample". Click on OK.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
```

### Code for "Program.cs"

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

namespace SelectExample
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }

        public DbSet<Employee> Employees { get; set; }
    }

    class Program
    {
        static void Main()
        {
            CompanyDbContext db = new CompanyDbContext();
            var emps = db.Employees.Select(temp => new { temp.EmpName, temp.Salary });
            foreach (var emp in emps)
            {
                Console.Write(emp.EmpName);
                Console.Write(",");
                Console.WriteLine(emp.Salary);
            }
        }
    }
}
```

```
        Console.WriteLine();
    }
    Console.ReadKey();
}
}
```

## Compile the Project

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B". The project will be compiled.

## Run the Project

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".

## Output:

Scott	,4000
Allen	,2500
Jones	,5200
James	,4400
Smith	,7600

## Distinct

- The `Distinct()` method is used to select unique (non-duplicate) values of a specific column in the table. It also sorts the column in ascending order. The `Distinct()` method doesn't receive any argument. We must call `Select()` method to select a single column, based on which you want to apply `Distinct`.

- **Syntax:**    `DbContextclass referencevariable = new DbContextclass();  
referencevariable.DbSetName.Select(temp => temp.ColumnName).Distinct().ToList();`

## Distinct - Example

## Creating Database

- Open SQL Server Management Studio. Select “Windows Authentication”. Click on “Connect”. Ignore this step if “departmentsandemployees” database already exists. Click on “New Query”. Type the following code:

```
create database departmentsandemployees
go
use departmentsandemployees
go
create table Departments(DeptNo int primary key, DeptName nvarchar(max), Loc nvarchar(max))
go
create table Employees(EmpID int primary key, EmpName varchar(max), Salary decimal, DeptNo int references
Departments(DeptNo))
go
insert into Departments values(10, 'Acunting', 'New York')
insert into Departments values(20, 'Operations', 'New Delhi')
insert into Departments values(30, 'Sales', 'New Jersy')
insert into Employees values(1, 'Scott', 2000, 10)
insert into Employees values(2, 'Allen', 6500, 10)
insert into Employees values(3, 'Jones', 4577, 20)
insert into Employees values(4, 'James', 2500, 20)
insert into Employees values(5, 'Smith', 3345, 30)
insert into Employees values(6, 'Harry', 3600, 30)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "Console App (.NET Framework)". Type the project name "DistinctExample". Type the location as "C:\mvc". Type the solution name as "DistinctExample". Click on OK.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.  
**install-package EntityFramework**

### Code for "Program.cs"

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

namespace DistinctExample
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
        public int DeptNo { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=departmentsandemployees")
        {
        }

        public DbSet<Employee> Employees { get; set; }
    }

    class Program
    {
        static void Main()
        {
            CompanyDbContext db = new CompanyDbContext();
            List<int> deptnos = db.Employees.Select(temp => temp.DeptNo).Distinct().ToList();
            foreach (int dno in deptnos)
            {
                Console.WriteLine(dno);
            }
            Console.ReadKey();
        }
    }
}
```

### Compile the Project

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B". The project will be compiled.

### Run the Project

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".

### Output:



```
C:\mvc\DistinctExample\DistinctExample\bin\Debug\DistinctExample.exe
10
20
30
```

### GroupBy

- The GroupBy() method is used to group-up the rows based on a specific column in the table. It splits rows into groups; the collection of groups is represented as `IQueryable<IGrouping<keydatatype, modelclassname>>` type; each group is represented as `IGrouping<keydatatype, modelclassname>`. Each group contains a key with a group of rows. The GroupBy() method receives a lambda expression as argument, which contains an argument (`temp`) that represents a row of the table, that returns a column based on which grouping is to be applied.

**Syntax:** `DbContextclass referencevariable = new DbContextclass();  
referencevariable.Dbsetname.GroupBy(temp => temp.columnname);`

### GroupBy - Example

#### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication".
- Click on "Connect".
- Ignore this step if "departmentsandemployees" database already exists.
- Click on "New Query".
- Type the following code:

```
create database departmentsandemployees
go
use departmentsandemployees
go
create table Departments(DeptNo int primary key, DeptName nvarchar(max), Loc nvarchar(max))
go
create table Employees(EmpID int primary key, EmpName varchar(max), Salary decimal, DeptNo int references
Departments(DeptNo))
go
insert into Departments values(10, 'Accounting', 'New York')
insert into Departments values(20, 'Operations', 'New Delhi')
insert into Departments values(30, 'Sales', 'New Jersy')
insert into Employees values(1, 'Scott', 2000, 10)
insert into Employees values(2, 'Allen', 6500, 10)
insert into Employees values(3, 'Jones', 4577, 20)
insert into Employees values(4, 'James', 2500, 20)
insert into Employees values(5, 'Smith', 3345, 30)
insert into Employees values(6, 'Harry', 3600, 30)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "Console App (.NET Framework)". Type the project name "GroupByExample". Type the location as "C:\mvc". Type the solution name as "GroupByExample". Click on OK.

#### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.  
`install-package EntityFramework`

#### Code for "Program.cs"

```
using System;
using System.ComponentModel.DataAnnotations;
```

```

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

namespace GroupByExample
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
        public int DeptNo { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=departmentsandemployees")
        {
        }

        public DbSet<Employee> Employees { get; set; }
    }

    class Program
    {
        static void Main()
        {
            CompanyDbContext db = new CompanyDbContext();
            var emps = db.Employees.GroupBy(temp => temp.DeptNo);
            foreach (var item in emps)
            {
                Console.WriteLine("Department: " + item.Key);
                foreach (var emp in item)
                {
                    Console.Write(emp.EmpID);
                    Console.Write(",");
                    Console.Write(emp.EmpName);
                    Console.Write(",");
                    Console.Write(emp.Salary);
                    Console.Write(",");
                    Console.Write(emp.DeptNo);
                    Console.WriteLine();
                }
                Console.WriteLine();
            }
            Console.ReadKey();
        }
    }
}

```

**Compile the Project**

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B".
- The project will be compiled.

**Run the Project**

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".

**Output:**

```
C:\mvc\GroupByExample\GroupByExample\bin\Debug\GroupByExample.exe
Department: 10
1,Scott,2000,10
2,Allen,6500,10

Department: 20
3,Jones,4577,20
4,James,2500,20

Department: 30
5,Smith,3345,30
6,Harry,3600,30
```

### Concat

- The Concat() method is used to concatenate two lists of same model class, into a single list. It attaches the second list at the end of first list. The Concat() method receives the second list as argument
- Syntax:**

```
DbContextclass referencevariable = new DbContextclass();
List<modelclassname> firstlist = referencevariable.Dbsetname.ToList();
List<modelclassname> secondlist = referencevariable.Dbsetname.ToList();
List<modelclassname> finalist = firstlist.Concat(secondlist).ToList();
```

### Concat - Example

#### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication".
- Click on "Connect".
- Ignore this step if "departmentsandemployees" database already exists.
- Click on "New Query".
- Type the following code:

```
create database departmentsandemployees
go
use departmentsandemployees
go
create table Departments(DeptNo int primary key, DeptName nvarchar(max), Loc nvarchar(max))
go
create table Employees(EmpID int primary key, EmpName varchar(max), Salary decimal, DeptNo int references
Departments(DeptNo))
go
insert into Departments values(10, 'Acouting', 'New York')
insert into Departments values(20, 'Operations', 'New Delhi')
insert into Departments values(30, 'Sales', 'New Jersy')
insert into Employees values(1, 'Scott', 2000, 10)
insert into Employees values(2, 'Allen', 6500, 10)
insert into Employees values(3, 'Jones', 4577, 20)
insert into Employees values(4, 'James', 2500, 20)
insert into Employees values(5, 'Smith', 3345, 30)
insert into Employees values(6, 'Harry', 3600, 30)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "Console App (.NET Framework)". Type the project name "ConcatExample". Type the location as "C:\mvc". Type the solution name as "ConcatExample". Click on OK.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.  
`install-package EntityFramework`

### Code for 'Program.cs'

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

namespace ConcatExample
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
        public int DeptNo { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=departmentsandemployees")
        {
        }

        public DbSet<Employee> Employees { get; set; }
    }

    class Program
    {
        static void Main()
        {
            CompanyDbContext db = new CompanyDbContext();
            List<Employee> emps1 = db.Employees.Where(temp => temp.DeptNo == 10).ToList();
            List<Employee> emps2 = db.Employees.Where(temp => temp.Salary > 3500).ToList();
            List<Employee> emps = emps1.Concat(emps2).ToList();
            foreach (Employee emp in emps)
            {
                Console.Write(emp.EmpID);
                Console.Write(",");
                Console.Write(emp.EmpName);
                Console.Write(",");
                Console.Write(emp.Salary);
                Console.Write(",");
                Console.Write(emp.DeptNo);
                Console.WriteLine();
            }
            Console.ReadKey();
        }
    }
}
```

### Compile the Project

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B". The project will be compiled.

### Run the Project

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".

**Output:**

```
C:\mvcc\ConcatExample\ConcatExample\bin\Debug\ConcatExample.exe
1,Scott,2000,10
2,Allen,6500,10
2,Allen,6500,10
3,Jones,4577,20
6,Harry,3600,30
```

**Union**

- The Union() method is used to combines two lists of same model class, into a single list. It removes duplicate rows, if any. The Union() method receives the second list as argument.

**Syntax:**

```
DbContextclass referencevariable = new DbContextclass();
List<modelclassname> firstlist = referencevariable.Dbsetname.ToList();
List<modelclassname> secondlist = referencevariable.Dbsetname.ToList();
List<modelclassname> finalist = firstlist.Union(secondlist).ToList();
```

**Union - Example****Creating Database**

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "departmentsandemployees" database already exists. Click on "New Query". Type the following code:

```
create database departmentsandemployees
go
use departmentsandemployees
go
create table Departments(
DeptNo int primary key,
DeptName nvarchar(max),
Loc nvarchar(max))
go
create table Employees(
EmpID int primary key,
EmpName varchar(max),
Salary decimal,
DeptNo int references Departments(DeptNo))
go
insert into Departments values(10, 'Acouting', 'New York')
insert into Departments values(20, 'Operations', 'New Delhi')
insert into Departments values(30, 'Sales', 'New Jersy')
insert into Employees values(1, 'Scott', 2000, 10)
insert into Employees values(2, 'Allen', 6500, 10)
insert into Employees values(3, 'Jones', 4577, 20)
insert into Employees values(4, 'James', 2500, 20)
insert into Employees values(5, 'Smith', 3345, 30)
insert into Employees values(6, 'Harry', 3600, 30)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

**Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "Console App (.NET Framework)". Type the project name "UnionExample". Type the location as "C:\mvcc". Type the solution name as "UnionExample". Click on OK.

**Installing Packages**

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
```

### Code for "Program.cs"

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

namespace UnionExample
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
        public int DeptNo { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=departmentsandemployees")
        {
        }

        public DbSet<Employee> Employees { get; set; }
    }

    class Program
    {
        static void Main()
        {
            CompanyDbContext db = new CompanyDbContext();
            List<Employee> emps1 = db.Employees.Where(temp => temp.DeptNo == 10).ToList();
            List<Employee> emps2 = db.Employees.Where(temp => temp.Salary > 3500).ToList();
            List<Employee> emps = emps1.Union(emps2).ToList();
            foreach (Employee emp in emps)
            {
                Console.Write(emp.EmpID);
                Console.Write(",");
                Console.Write(emp.EmpName);
                Console.Write(",");
                Console.Write(emp.Salary);
                Console.Write(",");
                Console.Write(emp.DeptNo);
                Console.WriteLine();
            }
            Console.ReadKey();
        }
    }
}
```

### Compile the Project

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B". The project will be compiled.

### Run the Project

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".

### Output:

```
C:\mvc\UnionExample\UnionExample\bin\Debug\UnionExample.exe
1,Scott,2000,10
2,Allen,6500,10
3,Jones,4577,20
6,Harry,3600,30
```

### Intersect

- The Intersect() method is used to combines two lists of same model class, into a single list. It only selects only common (duplicate) rows, if any. The Intersect() method receives the second list as argument.
- Syntax:**

```
DbContextclass referencevariable = new DbContextclass();
List<modelclassname> firstlist = referencevariable.Dbsetname.ToList();
List<modelclassname> secondlist = referencevariable.Dbsetname.ToList();
List<modelclassname> finalist = firstlist.Intersect(secondlist).ToList();
```

### Intersect - Example

#### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "departmentsandemployees" database already exists. Click on "New Query". Type the following code:

```
create database departmentsandemployees
go
use departmentsandemployees
go
create table Departments(DeptNo int primary key, DeptName nvarchar(max), Loc nvarchar(max))
go
create table Employees(EmpID int primary key, EmpName varchar(max), Salary decimal, DeptNo int references
Departments(DeptNo))
go
insert into Departments values(10, 'Accounting', 'New York')
insert into Departments values(20, 'Operations', 'New Delhi')
insert into Departments values(30, 'Sales', 'New Jersey')
insert into Employees values(1, 'Scott', 2000, 10)
insert into Employees values(2, 'Allen', 6500, 10)
insert into Employees values(3, 'Jones', 4577, 20)
insert into Employees values(4, 'James', 2500, 20)
insert into Employees values(5, 'Smith', 3345, 30)
insert into Employees values(6, 'Harry', 3600, 30)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "Console App (.NET Framework)". Type the project name "IntersectExample". Type the location as "C:\mvc". Type the solution name as "IntersectExample". Click on OK.

#### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.  
`install-package EntityFramework`

#### Code for "Program.cs"

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

```

namespace IntersectExample
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
        public int DeptNo { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=departmentsandemployees")
        {
        }

        public DbSet<Employee> Employees { get; set; }
    }

    class Program
    {
        static void Main()
        {
            CompanyDbContext db = new CompanyDbContext();
            List<Employee> emps1 = db.Employees.Where(temp => temp.DeptNo == 10).ToList();
            List<Employee> emps2 = db.Employees.Where(temp => temp.Salary > 3500).ToList();
            List<Employee> emps = emps1.Intersect(emps2).ToList();
            foreach (Employee emp in emps)
            {
                Console.Write(emp.EmpID);
                Console.Write(",");
                Console.Write(emp.EmpName);
                Console.Write(",");
                Console.Write(emp.Salary);
                Console.Write(",");
                Console.Write(emp.DeptNo);
                Console.WriteLine();
            }
            Console.ReadKey();
        }
    }
}

```

**Compile the Project**

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B". The project will be compiled.

**Run the Project**

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".

**Output:**

```
C:\mvc\IntersectExample\IntersectExample\bin\Debug\IntersectExample.exe
2,Allen,6500,10
```

**Except**

- The Except() method is used to select all rows of the first line, which are not present in the second list.
- The Intersect() method receives the second list as argument.

- Syntax:**

```
DbContextclass referencevariable = new DbContextclass();
List<modelclassname> firstlist = referencevariable.Dbsetname.ToList();
List<modelclassname> secondlist = referencevariable.Dbsetname.ToList();
List<modelclassname> finalist = firstlist.Except(secondlist).ToList();
```

### Except - Example

#### **Creating Database**

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "departmentsandemployees" database already exists. Click on "New Query". Type the following code:

```
create database departmentsandemployees
go
use departmentsandemployees
go
create table Departments(
DeptNo int primary key,
DeptName nvarchar(max),
Loc nvarchar(max))
go
create table Employees(
EmpID int primary key,
EmpName varchar(max),
Salary decimal,
DeptNo int references Departments(DeptNo))
go
insert into Departments values(10, 'Acouting', 'New York')
insert into Departments values(20, 'Operations', 'New Delhi')
insert into Departments values(30, 'Sales', 'New Jersy')
insert into Employees values(1, 'Scott', 2000, 10)
insert into Employees values(2, 'Allen', 6500, 10)
insert into Employees values(3, 'Jones', 4577, 20)
insert into Employees values(4, 'James', 2500, 20)
insert into Employees values(5, 'Smith', 3345, 30)
insert into Employees values(6, 'Harry', 3600, 30)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

#### **Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "Console App (.NET Framework)". Type the project name "ExceptExample". Type the location as "C:\mvcc". Type the solution name as "ExceptExample". Click on OK.

#### **Installing Packages**

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
```

#### **Code for "Program.cs"**

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

namespace ExceptExample
{
    public class Employee
    {
        [Key]
```

```

public int EmpID { get; set; }
public string EmpName { get; set; }
public decimal Salary { get; set; }
public int DeptNo { get; set; }
}

public class CompanyDbContext : DbContext
{
    public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=departmentsandemployees")
    {
    }

    public DbSet<Employee> Employees { get; set; }
}
class Program
{
    static void Main()
    {
        CompanyDbContext db = new CompanyDbContext();
        List<Employee> emps1 = db.Employees.Where(temp => temp.Salary > 3500).ToList();
        List<Employee> emps2 = db.Employees.Where(temp => temp.DeptNo == 10).ToList();
        List<Employee> emps = emps1.Except(emps2).ToList();
        foreach (Employee emp in emps)
        {
            Console.Write(emp.EmpID);
            Console.Write(",");
            Console.Write(emp.EmpName);
            Console.Write(",");
            Console.Write(emp.Salary);
            Console.Write(",");
            Console.Write(emp.DeptNo);
            Console.WriteLine();
        }
        Console.ReadKey();
    }
}
}

```

**Compile the Project**

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B". The project will be compiled.

**Run the Project**

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".

**Output:**

```
C:\mvc\ExceptExample\ExceptExample\bin\Debug\ExceptExample.exe
3,Jones,4577,20
6,Harry,3600,30
```

**Skip**

- The Skip() method is used to skip the specified no. of rows and select the remaining rows. The Skip() method receives the no. of rows as argument.
- **Syntax:** DbContextclass referencevariable = new DbContextclass();
referencevariable.Dbsetname.Skip(number of rows);

## Skip - Example

### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```
create database company
go
use company
go
create table Employees(EmpID int primary key, EmpName nvarchar(max), Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "Console App (.NET Framework)". Type the project name "SkipExample". Type the location as "C:\mvc". Type the solution name as "SkipExample". Click on OK.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
```

### Code for "Program.cs"

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

namespace SkipExample
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }
        public DbSet<Employee> Employees { get; set; }
    }
}

class Program
{
    static void Main()
    {
        CompanyDbContext db = new CompanyDbContext();
        List<Employee> emps = db.Employees.ToList().Skip(2).ToList();
        foreach (Employee emp in emps)
```

```
{  
    Console.WriteLine(emp.EmpID);  
    Console.Write(",");  
    Console.WriteLine(emp.EmpName);  
    Console.Write(",");  
    Console.WriteLine(emp.Salary);  
    Console.WriteLine();  
}  
Console.ReadKey();  
}  
}
```

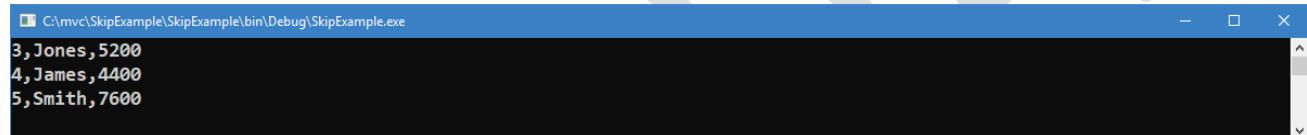
### Compile the Project

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B". The project will be compiled.

### Run the Project

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".

### Output:



```
C:\mvc\SkipExample\SkipExample\bin\Debug\SkipExample.exe  
3, Jones, 5200  
4, James, 4400  
5, Smith, 7600
```

## Take

- The Take() method is used to take (select) the specified no. of rows from the beginning of the table. The Take() method receives the no. of rows as argument.
- **Syntax:** DbContextclass referencevariable = new DbContextclass();  
referencevariable.Dbsetname.Take(number of rows);

## Take - Example

### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query".
- Type the following code:

```
create database company  
go  
use company  
go  
create table Employees(EmpID int primary key, EmpName nvarchar(max), Salary decimal)  
go  
insert into Employees values(1, 'Scott', 4000)  
insert into Employees values(2, 'Allen', 2500)  
insert into Employees values(3, 'Jones', 5200)  
insert into Employees values(4, 'James', 4400)  
insert into Employees values(5, 'Smith', 7600)  
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "Console App (.NET Framework)". Type the project name "TakeExample". Type the location as "C:\mvc". Type the solution name as "TakeExample". Click on OK.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.  
`install-package EntityFramework`

### Code for 'Program.cs'

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

namespace TakeExample
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }

        public DbSet<Employee> Employees { get; set; }
    }

    class Program
    {
        static void Main()
        {
            CompanyDbContext db = new CompanyDbContext();
            List<Employee> emps = db.Employees.Take(2).ToList();
            foreach (Employee emp in emps)
            {
                Console.Write(emp.EmpID);
                Console.Write(",");
                Console.Write(emp.EmpName);
                Console.Write(",");
                Console.Write(emp.Salary);
                Console.WriteLine();
            }
            Console.ReadKey();
        }
    }
}
```

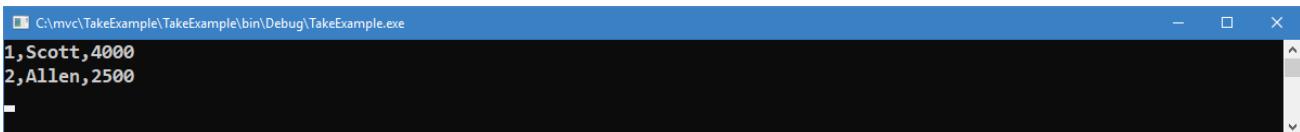
### Compile the Project

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B". The project will be compiled.

### Run the Project

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".

### Output:



```
C:\mvc\TakeExample\TakeExample\bin\Debug\TakeExample.exe
1,Scott,4000
2,Allen,2500
```

### First

- The First() method is used to select the first row of the table. The First() method doesn't receive any argument. It returns exception, if no rows are found in the table.
- Syntax:** DbContextclass referencevariable = new DbContextclass();  
referencevariable.Dbsetname.First();

### First - Example

#### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```
create database company
go
use company
go
create table Employees(EmpID int primary key, EmpName nvarchar(max), Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "Console App (.NET Framework)". Type the project name "FirstExample". Type the location as "C:\mvc". Type the solution name as "FirstExample". Click on OK.

#### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
```

#### Code for "Program.cs"

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

namespace FirstExample
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }
}
```

```

public class CompanyDbContext : DbContext
{
    public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
    {
    }
    public DbSet<Employee> Employees { get; set; }
}
class Program
{
    static void Main()
    {
        CompanyDbContext db = new CompanyDbContext();
        Employee emp = db.Employees.First();
        Console.WriteLine(emp.EmpID);
        Console.WriteLine(",");
        Console.WriteLine(emp.EmpName);
        Console.WriteLine(",");
        Console.WriteLine(emp.Salary);
        Console.ReadKey();
    }
}

```

**Compile the Project**

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B".
- The project will be compiled.

**Run the Project**

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".

**Output:**


```
C:\mvc\FirstExample\FirstExample\bin\Debug\FirstExample.exe
1,Scott,4000
```

**FirstOrDefault**

- The `FirstOrDefault()` method is used to select the first row of the table.
- The `FirstOrDefault()` method doesn't receive any argument.
- It returns null, if no rows are found in the table.
- **Syntax:** `DbContextclass referencevariable = new DbContextclass();  
referencevariable.Dbsetname.FirstOrDefault();`

**FirstOrDefault - Example****Creating Database**

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```

create database company
go
use company
go
create table Employees(
    EmpID int primary key,
    EmpName nvarchar(max),
    Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)

```

```
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "Console App (.NET Framework)". Type the project name "FirstOrDefaultExample". Type the location as "C:\mvc". Type the solution name as "FirstOrDefaultExample". Click on OK.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
```

### Code for "Program.cs"

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

namespace FirstOrDefaultExample
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }
        public DbSet<Employee> Employees { get; set; }
    }

    class Program
    {
        static void Main()
        {
            CompanyDbContext db = new CompanyDbContext();
            Employee emp = db.Employees.Where(temp => temp.Salary > 10000).FirstOrDefault();
            if (emp == null)
            {
                Console.WriteLine("No data found");
            }
            else
            {
                Console.Write(emp.EmpID);
                Console.Write(",");
                Console.Write(emp.EmpName);
                Console.Write(",");
                Console.Write(emp.Salary);
            }
        }
    }
}
```

```
        Console.ReadKey();
    }
}
```

### Compile the Project

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B". The project will be compiled.

### Run the Project

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".

#### Output:



```
C:\mvc\FirstOrDefaultExample\FirstOrDefaultExample\bin\Debug\FirstOrDefaultExample.exe
No data found
```

## Last

- The `Last()` method is used to select the last row of the table. The `Last()` method doesn't receive any argument. It returns exception, if no rows are found in the table.
- **Syntax:** `DbContextclass referencevariable = new DbContextclass();  
referencevariable.Dbsetname.Last();`

## Last - Example

### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```
create database company
go
use company
go
create table Employees(EmpID int primary key, EmpName nvarchar(max), Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "Console App (.NET Framework)". Type the project name "LastExample". Type the location as "C:\mvc". Type the solution name as "LastExample". Click on OK.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
```

### Code for "Program.cs"

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
```

```

using System.Collections.Generic;
using System.Linq;

namespace LastExample
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }

        public DbSet<Employee> Employees { get; set; }
    }

    class Program
    {
        static void Main()
        {
            CompanyDbContext db = new CompanyDbContext();
            Employee emp = db.Employees.ToList().Last();
            Console.WriteLine(emp.EmpID);
            Console.Write(",");
            Console.WriteLine(emp.EmpName);
            Console.Write(",");
            Console.WriteLine(emp.Salary);
            Console.ReadKey();
        }
    }
}

```

**Compile the Project**

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B". The project will be compiled.

**Run the Project**

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".

Output:

```
C:\mvc\LastExample\LastExample\bin\Debug\LastExample.exe
5,Smith,7600
```

**LastOrDefault**

- The `LastOrDefault()` method is used to select the last row of the table. The `LastOrDefault()` method doesn't receive any argument. It returns null, if no rows are found in the table.
- **Syntax:** `DbContextclass referencevariable = new DbContextclass();  
referencevariable.Dbsetname.LastOrDefault();`

**LastOrDefault - Example****Creating Database**

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```
create database company
```

```
go
use company
go
create table Employees(EmpID int primary key, EmpName nvarchar(max), Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "Console App (.NET Framework)". Type the project name "LastOrDefaultExample". Type the location as "C:\mvc". Type the solution name as "LastOrDefaultExample". Click on OK.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.  
`install-package EntityFramework`

### Code for "Program.cs"

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

namespace LastOrDefaultExample
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }
        public DbSet<Employee> Employees { get; set; }
    }

    class Program
    {
        static void Main()
        {
            CompanyDbContext db = new CompanyDbContext();
            Employee emp = db.Employees.Where(temp => temp.Salary > 10000).ToList().LastOrDefault();
            if (emp == null)
            {
                Console.WriteLine("No data found");
            }
            else
            {

```

```
Console.WriteLine(emp.EmpID);
Console.WriteLine(",");
Console.WriteLine(emp.EmpName);
Console.WriteLine(",");
Console.WriteLine(emp.Salary);
}

Console.ReadKey();
}
}
}
```

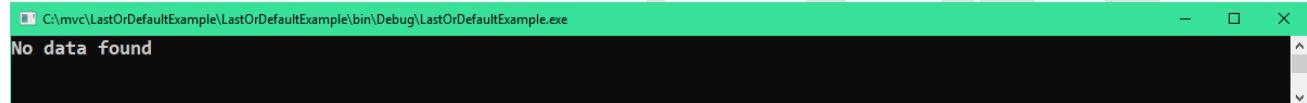
### Compile the Project

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B".
- The project will be compiled.

### Run the Project

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".

#### Output:



```
C:\mvc\LastOrDefaultExample\LastOrDefaultExample\bin\Debug\LastOrDefaultExample.exe
No data found
```

## Join

- The Join() method is used to combine two tables based on primary key and reference key. The first model class represents primary key table. The second model class represents reference key table. The third model class represents joined table. The Join() method receives four arguments:
  - argument 1: Second db set.
  - argument 2: Primary key column.
  - argument 3: Reference key column
  - argument 4: Lambda expression that represents list of columns to retrieve.
- **Syntax:**

```
DbContextclass referencevariable = new DbContextclass();
List<thirdmodelclass> joineddata = referencevariable.FirstDbsetname
    .Join( referencevariable.SecondDbSetname, temp => temp.Primarykey, temp2 => temp2.Referencekey, (temp1, temp2)
    => new thirdmodelclass() { Column1 = temp1.Column1, Column2 = temp2.Column2, ... } ).ToList();
```

## Join - Example

### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "departmentsandemployees" database already exists. Click on "New Query". Type the following code:

```
create database departmentsandemployees
go
use departmentsandemployees
go
create table Departments(DeptNo int primary key, DeptName nvarchar(max), Loc nvarchar(max))
go
create table Employees(EmpID int primary key, EmpName varchar(max), Salary decimal, DeptNo int references
Departments(DeptNo))
go
insert into Departments values(10, 'Accounting', 'New York')
insert into Departments values(20, 'Operations', 'New Delhi')
insert into Departments values(30, 'Sales', 'New Jersy')
insert into Employees values(1, 'Scott', 2000, 10)
insert into Employees values(2, 'Allen', 6500, 10)
insert into Employees values(3, 'Jones', 4577, 20)
insert into Employees values(4, 'James', 2500, 20)
insert into Employees values(5, 'Smith', 3345, 30)
```

```
insert into Employees values(6, 'Harry', 3600, 30)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "Console App (.NET Framework)". Type the project name "JoinExample". Type the location as "C:\mvc". Type the solution name as "JoinExample". Click on OK.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
```

### Code for "Program.cs"

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

namespace JoinsExample
{
    public class Department
    {
        [Key]
        public int DeptNo { get; set; }
        public string DeptName { get; set; }
        public string Loc { get; set; }
    }
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
        public int DeptNo { get; set; }
    }
    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=departmentsandemployees")
        {
        }
        public DbSet<Department> Departments { get; set; }
        public DbSet<Employee> Employees { get; set; }
    }
    public class JoinedModel
    {
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
        public int DeptNo { get; set; }
        public string DeptName { get; set; }
        public string Loc { get; set; }
    }
    class Program
    {
        static void Main()
```

```

{
    CompanyDbContext db = new CompanyDbContext();
    List<JoinedModel> emps = db.Departments.Join(db.Employees, d => d.DeptNo, e => e.DeptNo, (d, e) => new JoinedModel() {
        EmpID = e.EmpID, EmpName = e.EmpName, Salary = e.Salary, DeptNo = d.DeptNo, DeptName = d.DeptName, Loc = d.Loc
    }).ToList();
    foreach (JoinedModel emp in emps)
    {
        Console.Write(emp.EmpID);
        Console.Write(",");
        Console.Write(emp.EmpName);
        Console.Write(",");
        Console.Write(emp.Salary);
        Console.Write(",");
        Console.Write(emp.DeptNo);
        Console.WriteLine();
    }
    Console.ReadKey();
}
}
}

```

**Compile the Project**

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B". The project will be compiled.

**Run the Project**

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".

**Output:**

```

C:\mvc\JoinsExample\JoinsExample\bin\Debug\JoinsExample.exe
1,Scott,2000,10,Accounting,New York
2,Allen,6500,10,Accounting,New York
3,Jones,4577,20,Operations,New Delhi
4,James,2500,20,Operations,New Delhi
5,Smith,3345,30,Sales,New Jersey
6,Harry,3600,30,Sales,New Jersey

```

**Aggregate Methods**

- The aggregate methods combines all values of a column and returns a single value. The Sum() method returns sum of all values of the column. The Average() method returns average of all values of the column. The Min() method returns minimum value of all values of the column. The Max() method returns maximum value of all values of the column. The Count() method returns count of all values of the column.

- **Syntax:** DbContextclass referencevariable = new DbContextclass();
 decimal variable = referencevariable.Dbsetname.Select(temp => temp.columnname).Sum();
 decimal variable = referencevariable.Dbsetname.Select(temp => temp.columnname).Average();
 decimal variable = referencevariable.Dbsetname.Select(temp => temp.columnname).Min();
 decimal variable = referencevariable.Dbsetname.Select(temp => temp.columnname).Max();
 decimal variable = referencevariable.Dbsetname.Select(temp => temp.columnname).Count();

**Aggregate Methods - Example****Creating Database**

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "departmentsandemployees" database already exists. Click on "New Query". Type the following code:

```

create database departmentsandemployees
go
use departmentsandemployees

```

```

go
create table Departments(DeptNo int primary key, DeptName nvarchar(max), Loc nvarchar(max))
go
create table Employees(EmpID int primary key, EmpName varchar(max), Salary decimal, DeptNo int references
Departments(DeptNo))
go
insert into Departments values(10, 'Acunting', 'New York')
insert into Departments values(20, 'Operations', 'New Delhi')
insert into Departments values(30, 'Sales', 'New Jersy')
insert into Employees values(1, 'Scott', 2000, 10)
insert into Employees values(2, 'Allen', 6500, 10)
insert into Employees values(3, 'Jones', 4577, 20)
insert into Employees values(4, 'James', 2500, 20)
insert into Employees values(5, 'Smith', 3345, 30)
insert into Employees values(6, 'Harry', 3600, 30)
go

```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### **Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "Console App (.NET Framework)". Type the project name "AggregateOperatorsExample". Type the location as "C:\mvc". Type the solution name as "AggregateOperatorsExample". Click on OK.

### **Installing Packages**

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
```

### **Code for "Program.cs"**

```

using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
using System.Collections.Generic;
using System.Linq;

namespace AggregateOperatorsExample
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }

        public DbSet<Employee> Employees { get; set; }
    }

    class Program
    {
        static void Main()
        {
            CompanyDbContext db = new CompanyDbContext();
            decimal sum = db.Employees.Select(temp => temp.Salary).Sum();
            decimal avg = db.Employees.Select(temp => temp.Salary).Average();
        }
    }
}

```

```
    decimal min = db.Employees.Select(temp => temp.Salary).Min();
    decimal max = db.Employees.Select(temp => temp.Salary).Max();
    decimal count = db.Employees.Select(temp => temp.Salary).Count();
    Console.WriteLine("Sum: " + sum);
    Console.WriteLine("Avg: " + avg);
    Console.WriteLine("Min: " + min);
    Console.WriteLine("Max: " + max);
    Console.WriteLine("Count: " + count);
    Console.ReadKey();
}
}
```

## Compile the Project

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B". The project will be compiled.

## Run the Project

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".

### Output:

```
C:\mvc\AggregateOperatorsExample\AggregateOperatorsExample\bin\Debug\AggregateOperatorsExample.exe

Sum: 23700
Avg: 4740.000000
Min: 2500
Max: 7600
Count: 5
```

## Deferred Execution

- The LINQ query of EF executes when you call `ToList()`, `First()`, `FirstOrDefault()`, `Last()`, `LastOrDefault()` methods only; If you don't call any of these methods, the query will not execute. If you write the LINQ query in once place, call any of the above specified methods in other place, we call it as deferred execution. That means it executes differently, unlike regular statement, the LINQ query will be really processed / executed only you call any of the above methods.

- **Syntax:** `DbContextclass referencevariable = new DbContextclass();`  
`IQueryable<modelclassname> query = referencevariable.Dbsetname.Where(temp => temp.columnname);`  
`List<modelclassname> listname = query.ToList();`

## Deferred Execution - Example

## Creating Database

- Open SQL Server Management Studio. Select “Windows Authentication”. Click on “Connect”. Ignore this step if “departmentsandemployees” database already exists. Click on “New Query”. Type the following code:

```
create database departmentsandemployees
go
use departmentsandemployees
go
create table Departments(DeptNo int primary key, DeptName nvarchar(max), Loc nvarchar(max))
go
create table Employees(EmpID int primary key, EmpName varchar(max), Salary decimal, DeptNo int references
Departments(DeptNo))
go
insert into Departments values(10, 'Accounting', 'New York')
insert into Departments values(20, 'Operations', 'New Delhi')
insert into Departments values(30, 'Sales', 'New Jersey')
insert into Employees values(1, 'Scott', 2000, 10)
insert into Employees values(2, 'Allen', 6500, 10)
insert into Employees values(3, 'Jones', 4577, 20)
insert into Employees values(4, 'James', 2500, 20)
insert into Employees values(5, 'Smith', 3345, 30)
```

```
insert into Employees values(6, 'Harry', 3600, 30)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "Console App (.NET Framework)". Type the project name "DeferredExecutionExample". Type the location as "C:\mvc". Type the solution name as "DeferredExecutionExample". Click on OK.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
```

### Code for "Program.cs"

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

namespace DeferredExecutionExample
{
    public class Department
    {
        [Key]
        public int DeptNo { get; set; }
        public string DeptName { get; set; }
        public string Loc { get; set; }
    }
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
        public int DeptNo { get; set; }
    }
    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=departmentsandemployees")
        {
        }
        public DbSet<Department> Departments { get; set; }
        public DbSet<Employee> Employees { get; set; }
    }
    class Program
    {
        static void Main()
        {
            CompanyDbContext db = new CompanyDbContext();
            var depts = db.Departments.Where(temp => temp.DeptNo > 10);
            var emps = db.Employees.Where(temp => temp.Salary > 3000);
            List<Department> d = depts.ToList();
            List<Employee> e = emps.ToList();

            foreach (Department temp in d)
```

```
        {
            Console.Write(temp.DeptNo);
            Console.Write(",");
            Console.Write(temp.DeptName);
            Console.Write(",");
            Console.WriteLine(temp.Loc);
            Console.WriteLine();
        }
        Console.WriteLine();
    }

    foreach (Employee temp in e)
    {
        Console.Write(temp.EmpID);
        Console.Write(",");
        Console.Write(temp.EmpName);
        Console.Write(",");
        Console.WriteLine(temp.Salary);
        Console.Write(",");
        Console.WriteLine(temp.DeptNo);
        Console.WriteLine();
    }
    Console.ReadKey();
}
}
```

## Compile the Project

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B". The project will be compiled.

## Run the Project

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".

## Output:

```
C:\mvcc\DeferredExecutionExample\DeferredExecutionExample\bin\Debug\DeferredExecutionExample.exe  
20,Operations,New Delhi  
30,Sales,New Jersy  
  
2,Allen,6500,10  
3,Jones,4577,20  
5,Smith,3345,30  
6,Harry,3600,30
```

## NON-QUERY OPERATIONS IN EF

## Insertion

- We can insert new rows into the table using EF. The `Add()` method just adds the new model object to the existing virtual table. The `SaveChanges()` method automatically generates `INSERT` sql statement and executes the same at database.

- **Syntax:** DbContextclass referencevariable = new DbContextclass();  
Modelclass referencevariable2 = new Modelclass();  
referencevariable2.property1 = value;  
referencevariable2.property2 = value;  
...  
referencevariable.Dbsetname.Add(referencevariable2);  
referencevariable.SaveChanges()

## Insertion - Example

### **Creating Database**

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```
create database company
go
use company
go
create table Employees(EmpID int primary key, EmpName nvarchar(max), Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### **Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "Console App (.NET Framework)". Type the project name "InsertionExample". Type the location as "C:\mvc". Type the solution name as "InsertionExample". Click on OK.

### **Installing Packages**

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
```

### **Code for "Program.cs"**

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Data.Entity;
using System.Collections.Generic;
using System.Linq;

namespace InsertionExample
{
    public class Employee
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }
        public DbSet<Employee> Employees { get; set; }
    }

    class Program
    {
        static void Main()
        {
            CompanyDbContext db = new CompanyDbContext();
```

```
Console.WriteLine("Emp ID: ");
int a = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("Emp Name: ");
string b = Console.ReadLine();
Console.WriteLine("Salary: ");
decimal c = Convert.ToDecimal(Console.ReadLine());
Employee emp = new Employee();
emp.EmpID = a;
emp.EmpName = b;
emp.Salary = c;
db.Employees.Add(emp);
db.SaveChanges();
Console.WriteLine("Inserted");
Console.ReadKey();
}
```

## Compile the Project

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B". The project will be compiled.

## Run the Project

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".

### Output:

```
C:\mvc\InsertionExample\InsertionExample\bin\Debug\InsertionExample.exe
Emp ID:
20
Emp Name:
abc
Salary:
7000
Inserted
```

Updation

- We can update (modify) existing rows of the table using EF. The `FirstOrDefault()` method gets an existing row of the table. The `SaveChanges()` method automatically generates `UPDATE` sql statement and executes the same at database.

- **Syntax:**

```
DbContextclass referencevariable = new DbContextclass();
Modelclass referencevariable2 = referencevariable.Dbsetname.Where(temp => condition).FirstOrDefault();
referencevariable2.property1 = value;
referencevariable2.property2 = value;
...
referencevariable.SaveChanges();
```

## Updation - Example

## Creating Database

- Open SQL Server Management Studio. Select “Windows Authentication”. Click on “Connect”. Ignore this step if “company” database already exists. Click on “New Query”. Type the following code:

```
create database company
go
use company
go
create table Employees(EmpID int primary key, EmpName nvarchar(max), Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
```

```
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "Console App (.NET Framework)". Type the project name "UpdationExample". Type the location as "C:\mvc". Type the solution name as "UpdationExample". Click on OK.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened.
- Type the following command in "Package Manager Console" and press Enter.  
`install-package EntityFramework`

### Code for "Program.cs"

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Data.Entity;
using System.Collections.Generic;
using System.Linq;

namespace UpdationExample
{
    public class Employee
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }
        public DbSet<Employee> Employees { get; set; }
    }

    class Program
    {
        static void Main()
        {
            CompanyDbContext db = new CompanyDbContext();
            Console.WriteLine("Emp ID:");
            int a = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Emp Name:");
            string b = Console.ReadLine();
            Console.WriteLine("Salary:");
            decimal c = Convert.ToDecimal(Console.ReadLine());
            Employee emp = db.Employees.Where(temp => temp.EmpID == a).FirstOrDefault();
            if (emp != null)
            {
                emp.EmpName = b;
                emp.Salary = c;
                db.SaveChanges();
            }
        }
    }
}
```

```
        Console.WriteLine("Updated");
    }
    else
    {
        Console.WriteLine("Invalid EmpID");
    }
    Console.ReadKey();
}
}
```

## Compile the Project

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B". The project will be compiled.

### Run the Project

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".

### Output:

```
C:\mvc\UpdationExample\UpdationExample\bin\Debug\UpdationExample.exe

Emp ID:
20
Emp Name:
pqrs
Salary:
8888
Updated
-
```

## Deletion

- We can delete existing rows of the table using EF. The `FirstOrDefault()` method gets an existing row of the table. The `SaveChanges()` method automatically generates `DELETE` sql statement and executes the same at database.
  - **Syntax:**  
`DbContextclass referencevariable = new DbContextclass();`  
`Modelclass referencevariable2 = referencevariable.Dbsetname.Where(temp => condition).FirstOrDefault();`  
`referencevariable.SaveChanges();`

## Deletion - Example

## Creating Database

- Open SQL Server Management Studio. Select “Windows Authentication”. Click on “Connect”. Ignore this step if “company” database already exists. Click on “New Query”. Type the following code:

```
create database company
go
use company
go
create table Employees(
EmpID int primary key,
EmpName nvarchar(max),
Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "Console App (.NET Framework)". Type the project name "DeletionExample". Type the location as "C:\mvc". Type the solution name as "DeletionExample". Click on OK.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
```

### Code for "Program.cs"

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Data.Entity;
using System.Collections.Generic;
using System.Linq;

namespace DeletionExample
{
    public class Employee
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }

        public DbSet<Employee> Employees { get; set; }
    }

    class Program
    {
        static void Main()
        {
            CompanyDbContext db = new CompanyDbContext();
            Console.WriteLine("Emp ID:");
            int a = Convert.ToInt32(Console.ReadLine());
            Employee emp = db.Employees.Where(temp => temp.EmpID == a).FirstOrDefault();
            if (emp != null)
            {
                db.Employees.Remove(emp);
                db.SaveChanges();
                Console.WriteLine("Deleted");
            }
            else
            {
                Console.WriteLine("Invalid EmpID");
            }
            Console.ReadKey();
        }
    }
}
```

### Compile the Project

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B". The project will be compiled.

### Run the Project

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".

#### Output:

```
C:\mvc\DeleteExample\DeleteExample\bin\Debug\DeleteExample.exe
Emp ID:
20
Deleted
```

## Calling Stored Procedures using EF

- Stored Procedure is a collection of SQL statements, stored in the database.
- **Syntax:** `DbContextclass referencevariable = new DbContextclass();  
List<Modelclassname> referencevariable2 = referencevariable.Database.SqlQuery<Modelclassname>("stored  
procedure name").ToList();`

## Calling Stored Procedures using EF - Example

### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```
create database company
go
use company
go
create table Employees(
    EmpID int primary key,
    EmpName nvarchar(40),
    Salary decimal
)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 5000)
insert into Employees values(3, 'Jones', 6500)
insert into Employees values(4, 'James', 7000)
go
create procedure GetEmployees
as begin
    select * from Employees
end
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "Console App (.NET Framework)". Type the project name "StoredProcExample". Type the location as "C:\mvc". Type the solution name as "StoredProcExample". Click on OK.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
```

### Code for 'Program.cs'

```
using System;
```

```

using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
using System.Collections.Generic;
using System.Linq;

namespace StoredProcedureExample
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }

        public DbSet<Employee> Employees { get; set; }
    }

    class Program
    {
        static void Main()
        {
            CompanyDbContext db = new CompanyDbContext();
            List<Employee> emps = db.Database.SqlQuery<Employee>("GetEmployees").ToList();
            foreach (Employee emp in emps)
            {
                Console.Write(emp.EmpID);
                Console.Write(",");
                Console.Write(emp.EmpName);
                Console.Write(",");
                Console.Write(emp.Salary);
                Console.WriteLine();
            }
            Console.ReadKey();
        }
    }
}

```

**Compile the Project**

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B". The project will be compiled.

**Run the Project**

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".

**Output:**

```

C:\mvc\StoredProcedureExample\StoredProcedureExample\bin\Debug\StoredProcedureExample.exe
1,Scott,4000
2,Allen,5000
3,Jones,6500
4,James,7000

```

**Stored Procedures with Parameters**

- You can pass parameters to the stored procedures using EF.

- **Syntax:**

```
DbContextclass referencevariable = new DbContextclass();
List<Modelclassname> referencevariable2 = referencevariable.Database.SqlQuery<Modelclassname>("stored
procedure name @parameternamel, @parametername2,...", value1, value2, ...).ToList();
```

### Stored Procedures with Parameters - Example

#### **Creating Database**

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```
create database company
go
use company
go
create table Employees(
    EmpID int primary key,
    EmpName nvarchar(40),
    Salary decimal
)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 5000)
insert into Employees values(3, 'Jones', 6500)
insert into Employees values(4, 'James', 7000)
go
create procedure sp_searchemployees
(@str nvarchar(max))
as begin
    select * from Employees where EmpName like '%' + @str + '%'
end
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

#### **Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "Console App (.NET Framework)". Type the project name "StoredProcedureWithParams". Type the location as "C:\mvc". Type the solution name as "StoredProcedureWithParams". Click on OK.

#### **Installing Packages**

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
```

#### **Code for 'Program.cs'**

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
using System.Collections.Generic;
using System.Linq;
using System.Data.SqlClient;

namespace StoredProcedureWithParams
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }
}
```

```

public class CompanyDbContext : DbContext
{
    public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
    {
    }
    public DbSet<Employee> Employees { get; set; }
}
class Program
{
    static void Main()
    {
        Console.WriteLine("Enter Emp Name to Search:");
        string s = Console.ReadLine();
        SqlParameter p1 = new SqlParameter("@str", s);
        CompanyDbContext db = new CompanyDbContext();
        List<Employee> emps = db.Database.SqlQuery<Employee>("sp_searchemployees @str", p1).ToList();
        foreach (Employee emp in emps)
        {
            Console.Write(emp.EmpID);
            Console.Write(",");
            Console.Write(emp.EmpName);
            Console.Write(",");
            Console.Write(emp.Salary);
            Console.WriteLine();
        }
        Console.ReadKey();
    }
}

```

**Compile the Project**

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B". The project will be compiled.

**Run the Project**

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".

Output:

```
C:\mvcsStoredProcWithParams\StoredProcWithParams\bin\Debug\StoredProcWithParams.exe
Enter Emp Name to Search:
a
2,Allen,5000
4,James,7000
```

**Data Annotations**

- Data Annotations are the attributes present in "System.ComponentModel.DataAnnotations" namespace or "System.ComponentModel.DataAnnotations.Schema" namespace. These are used to specify settings of the model class or properties.

<b>Data Annotation Attribute</b>	<b>Description</b>
Key	Used to specifies primary key column.
Table	Used to specify table name, to which the model class is binded.
Column	Used to specify the column name, to which the property is binded.
NotMapped	Specifies that the property should not be stored in the database table.
DatabaseGenerated	Specifies whether the primary key column is auto-generated (identity) or normal column.

ForeignKey	Used specify foreign key relation with primary key.
------------	---

### Data Annotations - Example

#### **Creating Database**

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "departmentsandemployees" database already exists. Click on "New Query". Type the following code:

```

create database departmentsandemployees
go
use departmentsandemployees
go
create table Departments(DeptNo int primary key, DeptName nvarchar(max), Loc nvarchar(max))
go
create table Employees(EmpID int primary key, EmpName varchar(max), Salary decimal, DeptNo int references
Departments(DeptNo))
go
insert into Departments values(10, 'Accounting', 'New York')
insert into Departments values(20, 'Operations', 'New Delhi')
insert into Departments values(30, 'Sales', 'New Jersy')
insert into Employees values(1, 'Scott', 2000, 10)
insert into Employees values(2, 'Allen', 6500, 10)
insert into Employees values(3, 'Jones', 4577, 20)
insert into Employees values(4, 'James', 2500, 20)
insert into Employees values(5, 'Smith', 3345, 30)
insert into Employees values(6, 'Harry', 3600, 30)
go

```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

#### **Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "Console App (.NET Framework)". Type the project name "DataAnnotationsExample". Type the location as "C:\mvc". Type the solution name as "DataAnnotationsExample". Click on OK.

#### **Installing Packages**

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
```

#### **Code for "Program.cs"**

```

using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Data.Entity;
using System.Collections.Generic;
using System.Linq;

namespace DataAnnotationsExample
{
    [Table("Departments")]
    public class Dept
    {
        [Key]
        public int DeptNo { get; set; }
        public string DeptName { get; set; }
        [Column("Loc")]
        public string Location { get; set; }
    }
    public class Employee
    {

```

```

[Key]
public int EmpID { get; set; }
public string EmpName { get; set; }
public decimal Salary { get; set; }
public int DeptNo { get; set; }
[ForeignKey("DeptNo")]
public virtual Dept Department { get; set; }
[NotMapped]
public double Tax { get; set; } = 1000;
}
public class CompanyDbContext : DbContext
{
    public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=departmentsandemployees")
    {
    }
    public DbSet<Dept> Departments { get; set; }
    public DbSet<Employee> Employees { get; set; }
}
class Program
{
    static void Main()
    {
        CompanyDbContext db = new CompanyDbContext();
        List<Employee> emps = db.Employees.ToList();
        foreach (Employee emp in emps)
        {
            Console.Write(emp.EmpID);
            Console.Write(",");
            Console.Write(emp.EmpName);
            Console.Write(",");
            Console.Write(emp.Salary);
            Console.Write(",");
            Console.Write(emp.DeptNo);
            Console.Write(",");
            Console.Write(emp.Department.DeptName);
            Console.Write(",");
            Console.Write(emp.Department.Location);
            Console.Write(",");
            Console.Write(emp.Tax);
            Console.WriteLine();
        }
        Console.ReadKey();
    }
}
}

```

**Compile the Project**

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B". The project will be compiled.

**Run the Project**

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".

**Output:**

```

C:\mvcc\DataAnnotationsExample\bin\Debug\DataAnnotationsExample.exe
1,Scott,2000,10,Accounting>New York,1000
2,Allen,6500,10,Accounting>New York,1000
3,Jones,4577,20,Operations>New Delhi,1000
4,James,2500,20,Operations>New Delhi,1000
5,Smith,3345,30,Sales>New Jersy,1000
6,Harry,3600,30,Sales>New Jersy,1000

```

## Chapter 2 : ASP.NET MVC 5

---

### FUNDAMENTALS OF MVC

#### Introduction to ASP.NET MVC

- "ASP.NET MVC" is a "web application framework", which is used to develop server side programs for web applications using "MVC architecture". MVC full form is "Model - View - Controller". ASP.NET MVC supports "MVC architecture". ASP.NET MVC's first version was released in 2009. "ASP.NET MVC" is a part of "ASP.NET". "ASP.NET MVC" is an alternative to "ASP.NET Web Forms".
- **Language:** It is a set of rules and syntaxes to write programs to communicate with system. Ex: C, Java, C#.NET etc.
- **Design Pattern:** It is a known solution for known problem in design. Ex: Factory, Abstract, Singleton etc.
- **Architecture:** It is a pictorial representation of components (Layers), which defines structure of the application. Ex: Three-Tier Architecture, MVC architecture, MVVM architecture etc.
- **Framework:** It is a ready-made project skeleton, which is developed based on a specific language, specific design pattern and specific architecture, to create specific type of applications. The framework provides built-in code for common project tasks and let the developers to develop realtime applications easily and faster. Ex: ASP.NET MVC, Spring, Cake PHP etc.

#### Advantages of ASP.NET MVC

- **Faster Performance:**

- ASP.NET executes Page Life Cycle executes everytime when you send request to the application. It converts the asp.net tags in aspx file into c#.net code; compiles the c#.net code into MSIL language; creates objects for each control on the page; executes initialization stage, loading stage, rendering stage, unloading stage for each control on the page; executes PreInit, Init, PreLoad, Load, PreRender, Unload events for each control on the page; So the asp.net page life cycle is heavy and kills the performance of the web server; ASP.NET MVC doesn't support controls and page life cycle; so it is light-weight and faster.

- **Clean separation of concerns:**

- ASP.NET doesn't support to develop ".aspx" file and ".aspx.cs" files independently. ASP.NET MVC supports to develop model, controller and view independently and parallelly.

- **Unit Testing:**

- ASP.NET doesn't support unit testing on ".aspx.cs" file, without ".aspx" file. ASP.NET MVC supports unit testing on controllers, models and views.

#### MVC Architecture

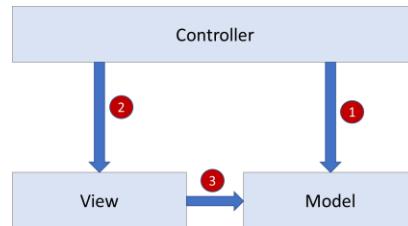
- The design pattern is a "solution for known problem in design". The architecture is a "pictorial representation of components of the application". MVC is a "design pattern cum architecture", which dictates you how to divide the project code as multiple individual units (layers) and manage them easily. So that multiple developers (or teams) can involve in the development parallel, without depending on each other.
- MVC architecture was introduced in 1979, by "Reenskaug".
- To create apps using "MVC", we require "MVC framework". An "MVC framework" provides necessary libraries to use "MVC architecture".
  1. **Java:** Spring
  2. **PHP:** Cake PHP
  3. **NodeJS:** Express
  4. **JavaScript:** AngularJS
  5. **iOS:** Objective C
  6. **.NET:** ASP.NET MVC

#### Model, View and Controller

- **Model:** Model is a class that stores the data.
- **View:** View is a web page that contains presentation logic to display model data.

## Asp.Net Mvc 5

- **Controller:** Controller is a class, which receives request from browser, creates model object, passes model object to the view, receives viewresult after execution of view and passes viewresult to the browser as response.



### Basic Principles of MVC

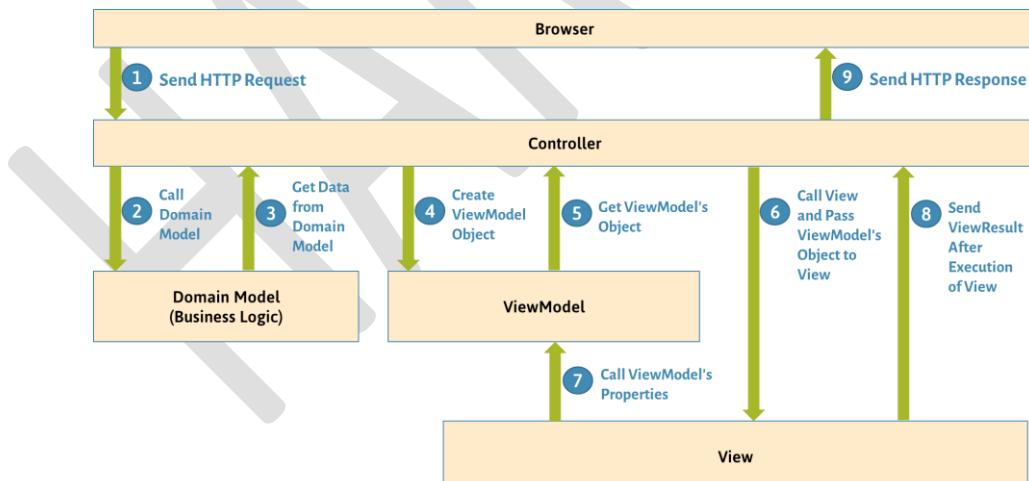
1. Controller calls Model (Controller creates object for the model class).
2. Controller calls View.
3. View calls Model (View gets data from model object).

### Summary

"Controller" passes "model" objects to "view".

#### MVC Execution Flow

- **User:** Enter the URL.
- **Browser:** Sends a HTTP request to the server.
- **Controller:** Receives the HTTP request from the browser.
- **Controller:** Creates an object for the model class & adds data to it.
- **Controller:** Calls View & passes model object to the view.
- **View:** Loads data from model class's object and renders data in respective place holders.
- **View:** Executes the server side code of the view and generates the "view result". The "view result" contains the html code that is generated after executing the view.
- **Controller:** Sends the "view result" to browser as response.
- **Browser:** Receives the "view result" (html).
- **Browser:** Executes the html code and displays the output to the user.



#### Versions of ASP.NET MVC

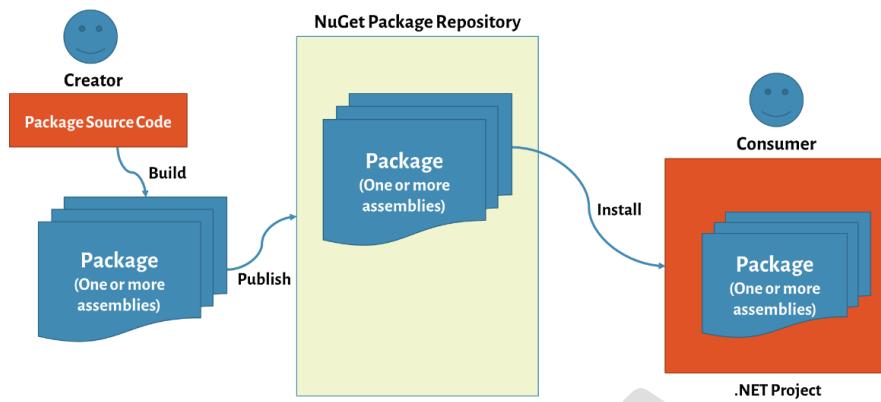
Sl. No	Version	Month & Year of release	Visual Studio	.NET Framework	Features
1	ASP.NET MVC 1	March 2009	Visual Studio 2008	.NET 3.5	<ul style="list-style-type: none"> <li>• MVC pattern with ASPX Engine</li> <li>• HTML Helpers</li> <li>• Routing</li> </ul>

## Asp.Net Mvc 5

					<ul style="list-style-type: none"> <li>• Unit Testing</li> </ul>
2	ASP.NET MVC 2	March 2010	Visual Studio 2008 (or) Visual Studio 2010	.NET 3.5 .NET 4.0	<ul style="list-style-type: none"> <li>• Strongly typed HTML Helpers</li> <li>• Support for data annotations</li> </ul>
3	ASP.NET MVC 3	January 2011	Visual Studio 2010 (or) Visual Studio 2012	.NET 4.0	<ul style="list-style-type: none"> <li>• Razor</li> <li>• EF Code First</li> <li>• Partial Page Output Caching</li> <li>• ViewBag</li> <li>• Global Action Filters</li> </ul>
4	ASP.NET MVC 4	August 2012	Visual Studio 2010 SP1 (or) Visual Studio 2012 (or) Visual Studio 2013	.NET 4.0 .NET 4.5	<ul style="list-style-type: none"> <li>• ASP.NET Web API</li> <li>• Bundling and Minification</li> <li>• Asynchronous Controllers</li> </ul>
5	ASP.NET MVC 5	October 2013	Visual Studio 2012 (or) Visual Studio 2013 (or) Visual Studio 2015 (or) Visual Studio 2017	.NET 4.5 .NET 4.5.1	<ul style="list-style-type: none"> <li>• ASP.NET Web API 2</li> <li>• ASP.NET Identity</li> <li>• Attribute Based Routing</li> <li>• Filter Overrides</li> </ul>
6	ASP.NET Core MVC 1.0 (Earlier it was ASP.NET MVC 6)	Aug 2016	Visual Studio 2015 (or) Visual Studio 2017	.NET 4.6 .NET 4.6.1 .NET 4.6.2 .NET 4.7	<ul style="list-style-type: none"> <li>• More light-weight</li> <li>• Request pipeline</li> <li>• Built based on ASP.NET Core</li> <li>• Cross-platform</li> <li>• Open source</li> <li>• Better support for cloud (Microsoft Azure)</li> </ul>
7	ASP.NET Core MVC 2.0	Aug 2017	Visual Studio 2015 (or) Visual Studio 2017	.NET 4.6 .NET 4.6.1 .NET 4.6.2 .NET 4.7	<ul style="list-style-type: none"> <li>• Microsoft.AspNetCore.All Package</li> <li>• Http.Sys package</li> </ul>

### Introduction to NuGet Packages

- The NuGet Package is a collection of assemblies (DLL Files).
- When the developer installs a NuGet Package in the Project, it automatically adds corresponding DLL files to the project; so we can use all of its API in the project.
- All NuGet Packages are stored at "nuget.org" server.
- One NuGet Package may depend on other NuGet Packages also. Those packages are called as "dependencies".
- The "packages.config" file is an XML file, which contains the list of packages that you want to download / install in the current project.
- The "NuGet Package Manager" is a command-line tool, which is used to install / un-install packages in the current project.



### Syntax of "packages.config"

```

<?xml version="1.0" encoding="utf-8"?>
<packages>
  <package id="Packagename" version="number" targetFramework="net47" />
  ...
</packages>
  
```

### NuGet Packages of ASP.NET MVC

- The following NuGet Packages are need to run ASP.NET MVC application:
  - Microsoft.AspNet.Mvc**: Provides the API for development of model, view and controller.
  - Microsoft.AspNet.WebPages**: Provides the API for development of views (web pages).
  - Microsoft.AspNet.Razor**: Provides the API to use Razor Syntaxes (C#.NET code) in the Views.
  - Microsoft.CodeDom.Providers.DotNetCompilerPlatform**: Provides compiler for MVC applications.
  - Microsoft.Net.Compilers**: Provides compiler for C#.NET.
  - Microsoft.Web.Infrastructure**: Provides API for URL Routing.

### Folder Structure of MVC Application

- Controllers**: Contains all the controls of the project.
- Views**: Contains all the views of the project.
- Models**: Contains all the models of the project.
- Packages.config**: Contains the list of NuGet packages that are installed in the current project.
- Web.config**: Contains configuration settings of the project.
- Global.asax**: Contains methods like Application\_Start(), Application\_End(), Session\_Start(), Session\_End(), Application\_Error() etc.
- App\_Start**
  - RouteConfig.cs**: Contains code for URL Routing.
  - BundleConfig.cs**: Contains code for Bundling and Minification.
  - FilterConfig.cs**: Contains code for Global Filters.

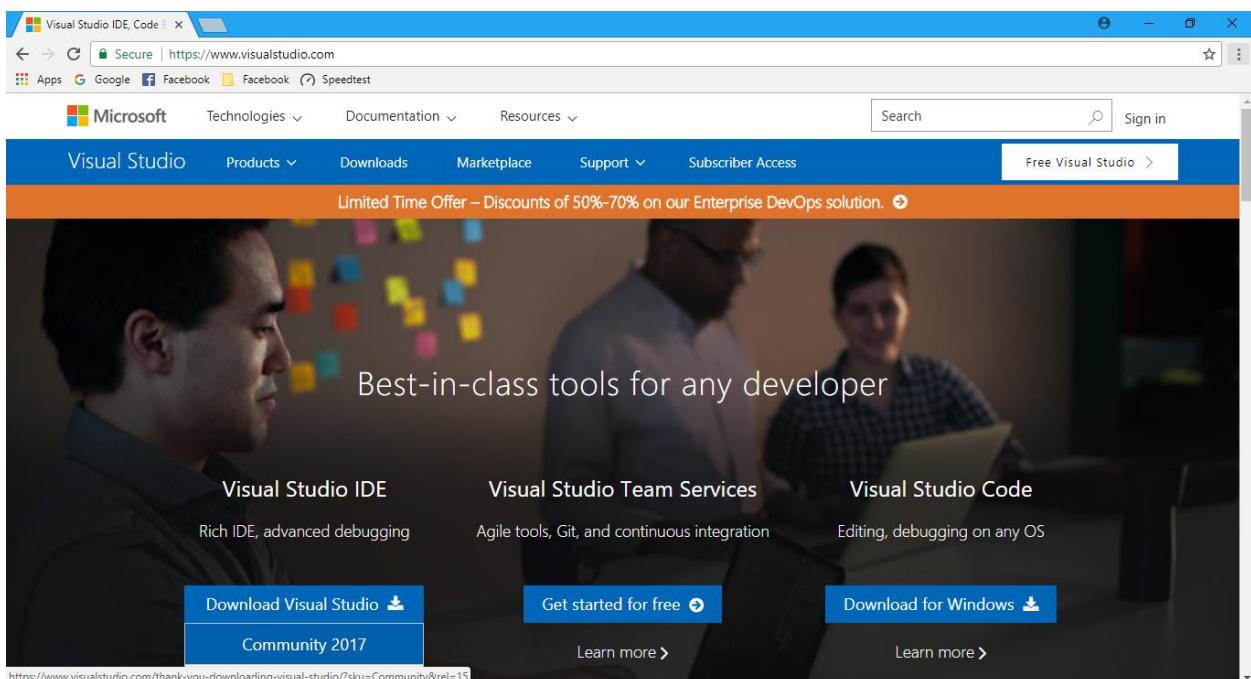
### Steps to Prepare First Example in MVC

1. Installing Visual Studio 2017
2. Creating MVC Project
3. Compiling MVC Project
4. Executing MVC Project

#### 1. Steps to install "Visual Studio 2017"

Go to <http://www.visualstudio.com>

Click on "Download Visual Studio" - "Community 2017".

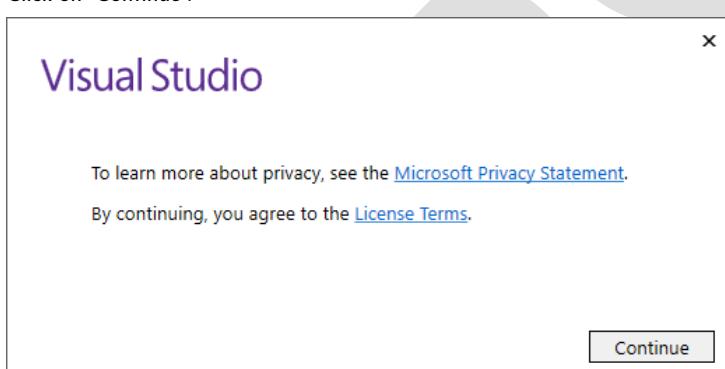


You will get "vs\_community.exe" file.

Run "vs\_community.exe" file.

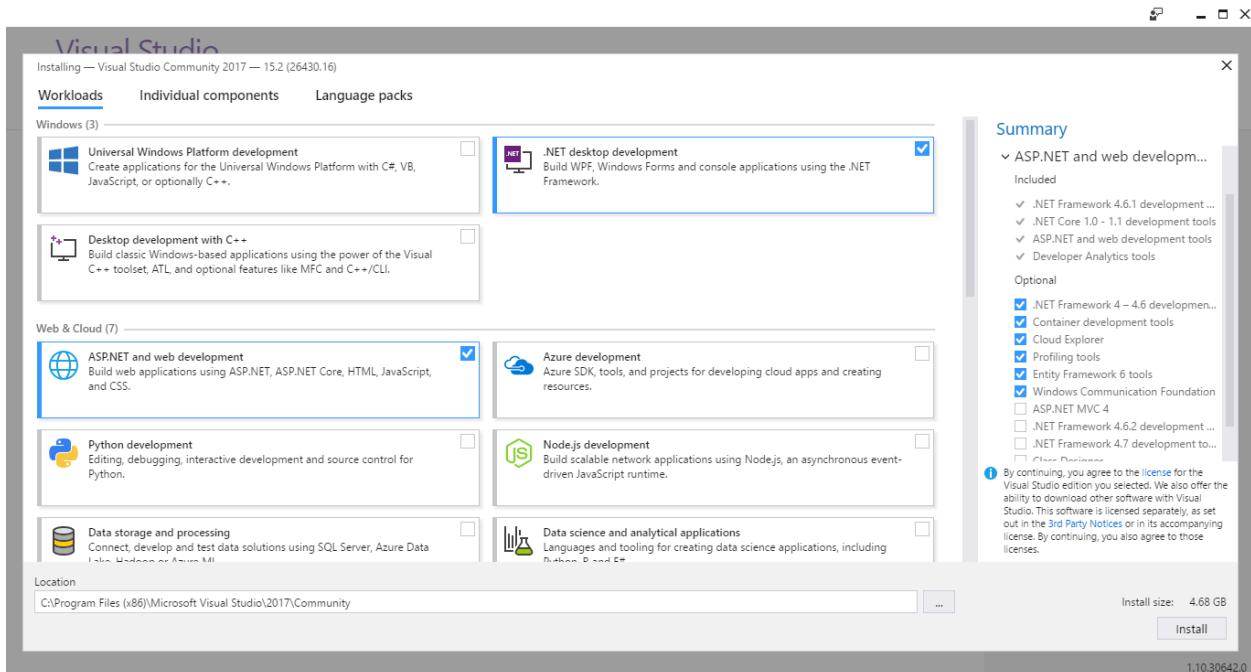
Click on "Run".

Click on "Continue".



Check the checkboxes ".NET desktop development" and "ASP.NET and web development".

## Asp.Net Mvc 5



Click on "Install".

The screenshot shows the Visual Studio 2017 Community edition setup window. The 'Products' tab is selected. In the 'Installed' section, the download progress for 'Visual Studio Community 2017' is shown, with two progress bars at 0% completion. In the 'Available' section, 'Visual Studio Enterprise 2017' and 'Visual Studio Professional 2017' are listed, each with an 'Install' button. The 'Welcome!' section features a message encouraging users to go online for additional tools. The 'Marketplace' section provides links to Microsoft Developer Community and Visual Studio Support. The total install size is 4.68 GB.

Wait for installation to be completed.

## Visual Studio

### Products

#### Installed

##### Visual Studio Community 2017

15.2 (26430.16)

A restart is required. If needed, any remaining setup will resume automatically after the restart.

[Release notes](#)[Restart](#)

Reboot required

Success! One more step to go. Please restart your computer before you start Visual Studio Community 2017.

[Get troubleshooting tips](#)[Restart](#)[Not now](#)

#### Available

##### Visual Studio Enterprise 2017

Microsoft DevOps solution for productivity and coordination across teams of any size

[License terms](#) | [Release notes](#)

15.2 (26430.16)

[Install](#)

### Welcome!

We invite you to go online to hone your skills and find additional tools to support your development workflow.

#### Learn

Whether you're new to development or an experienced developer, we have you covered with our tutorials, videos, and sample code.

#### Marketplace

Use Visual Studio extensions to add support for new technologies, integrate with other products and services, and fine-tune your experience.

#### Need some help?

Check out the [Microsoft Developer Community](#) where developers provide feedback and answers to many common problems.

Get help from Microsoft at [Visual Studio Support](#)

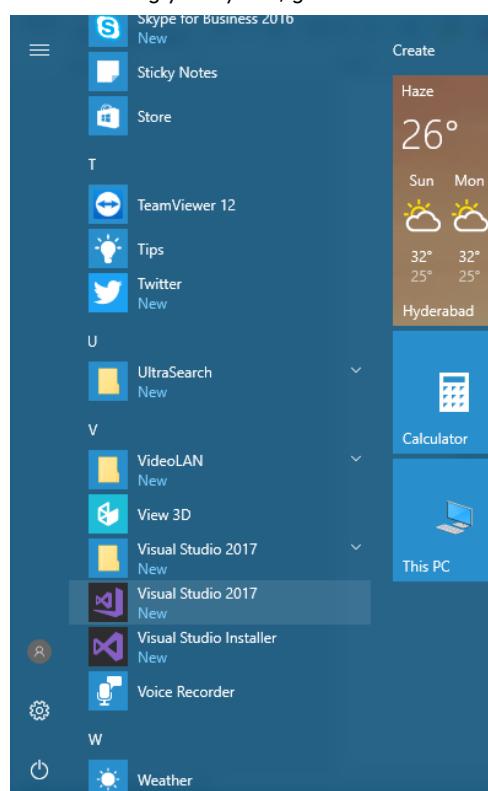
##### Visual Studio Professional 2017

Professional developer tools and services for small teams

1.10.30642.0

Click on "Restart".

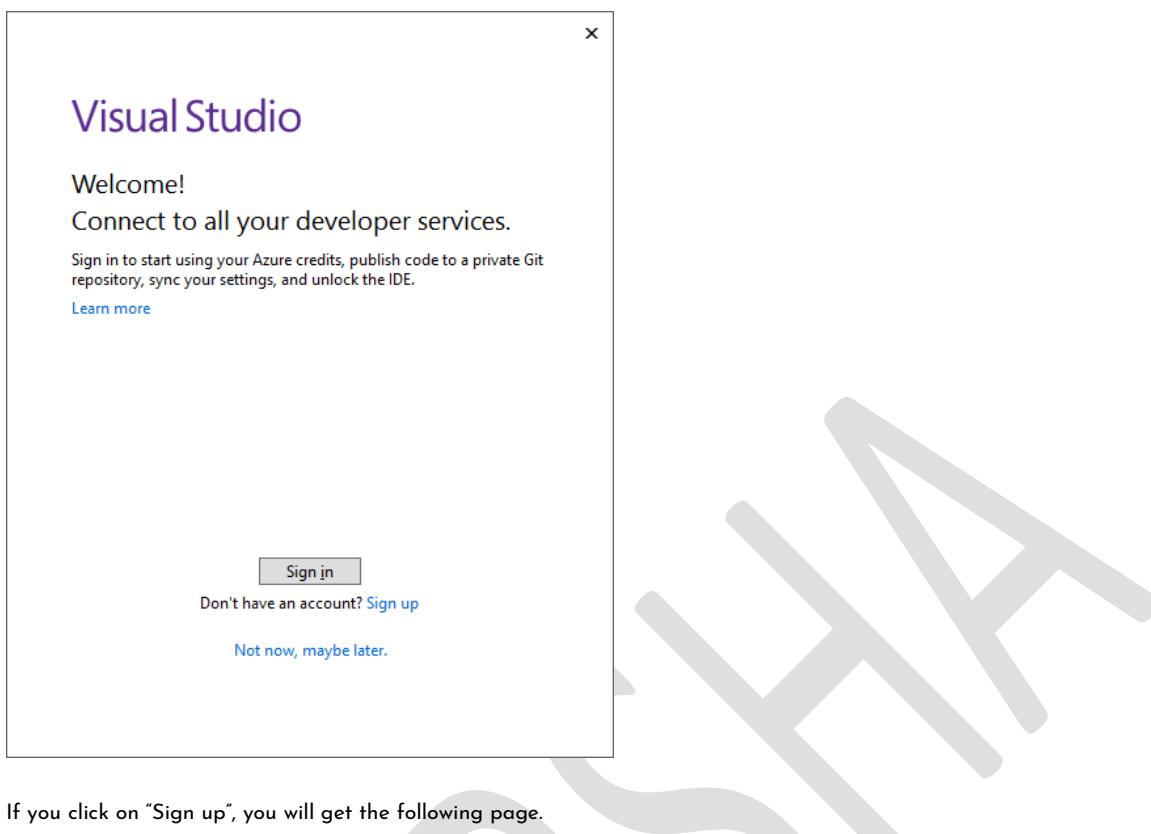
After restarting your system, go to "Start" > "Visual Studio 2017".



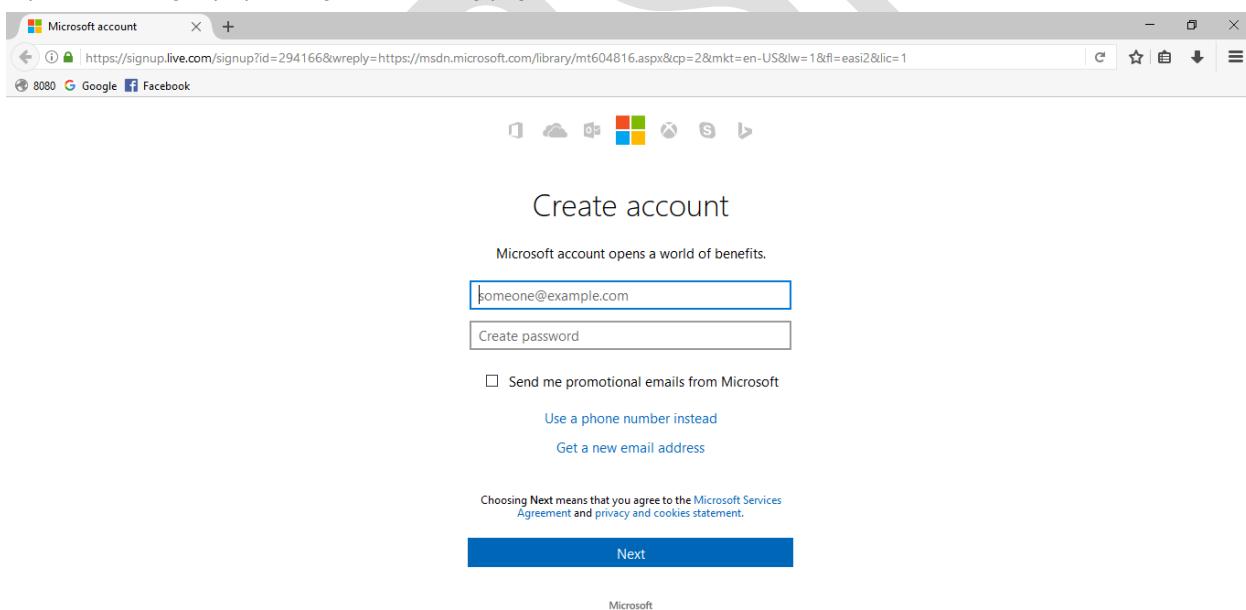
If you have Microsoft account already and click on "Sign in" and complete the login process.

If you don't have Microsoft account, click on "Sign up" and complete the registration process.

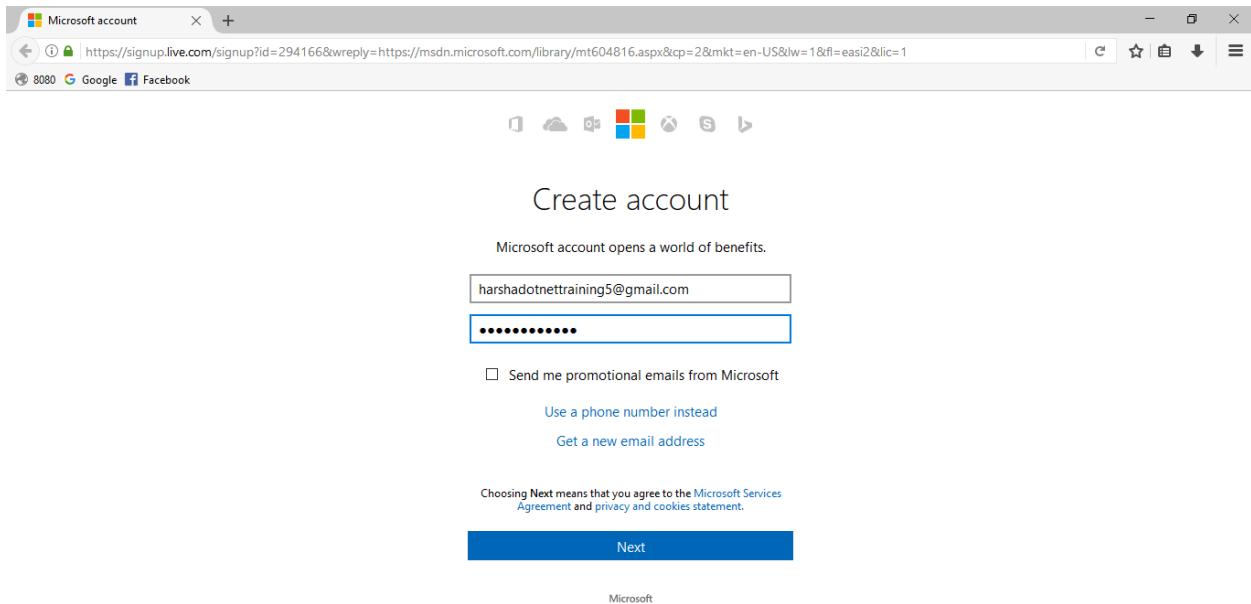
Note: If you don't want to login, click on "Not now, maybe later"; but then Visual Studio expires in 30 days.



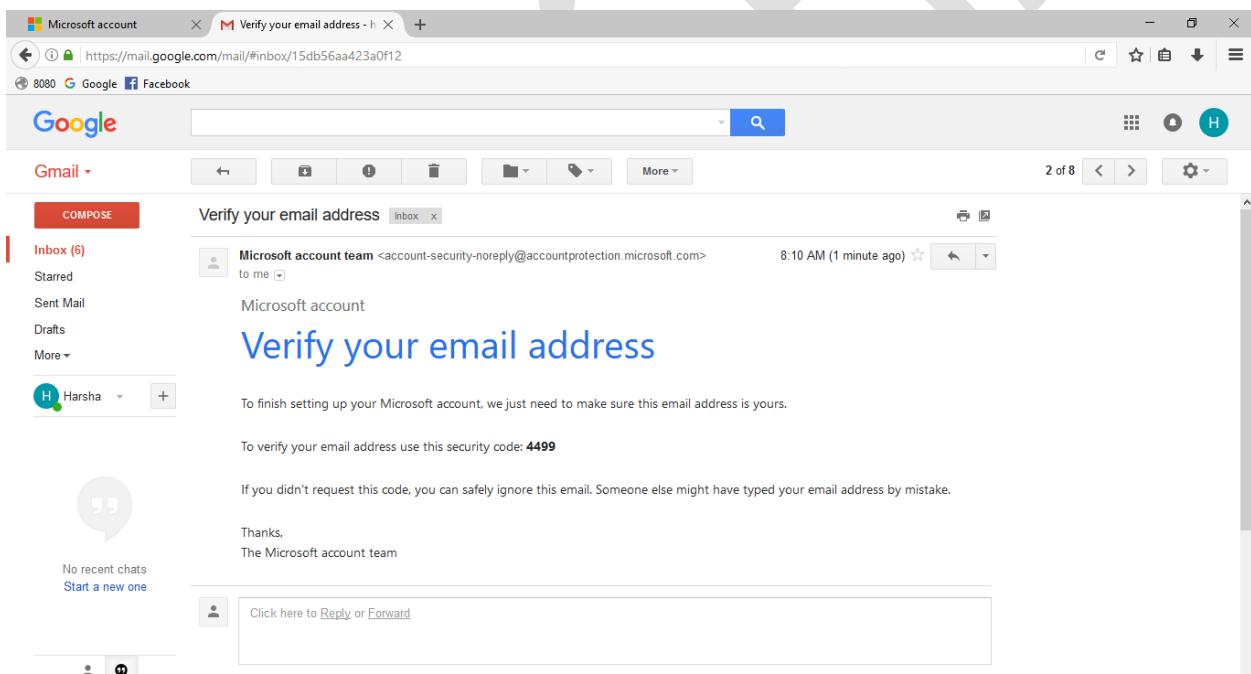
If you click on "Sign up", you will get the following page.



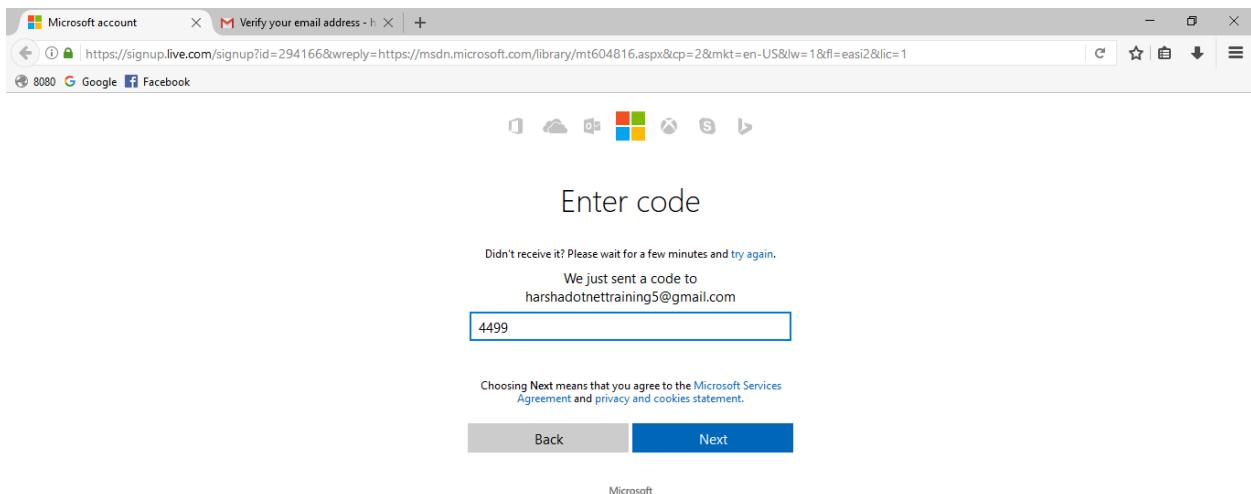
Enter your email id and enter new password.



Click on "Next".



You will get a security code to your email. Login into your gmail, check the code that you have received and enter it in this page.

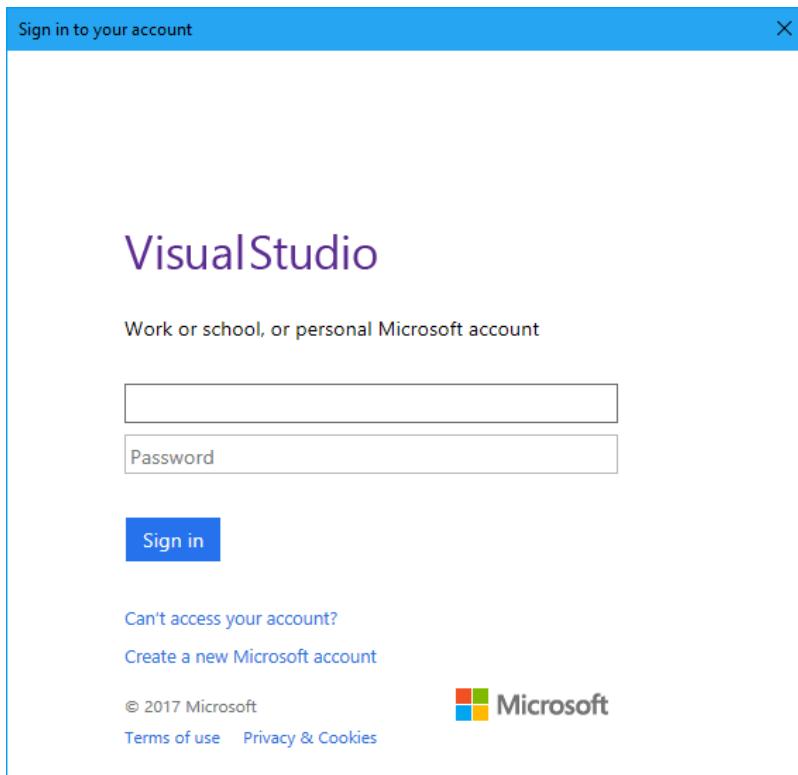


After entering the security code, click on "Next".

A screenshot of the MSDN Library developer network page. The URL is https://msdn.microsoft.com/library/mt604816.aspx. The main content area says 'Return to Visual Studio to sign in to the IDE with your new Microsoft account!'. It includes a note about successfully creating a Microsoft Account and instructions to sign in from Visual Studio. A screenshot of the Visual Studio welcome screen is shown, with the 'Sign in' button circled in red.

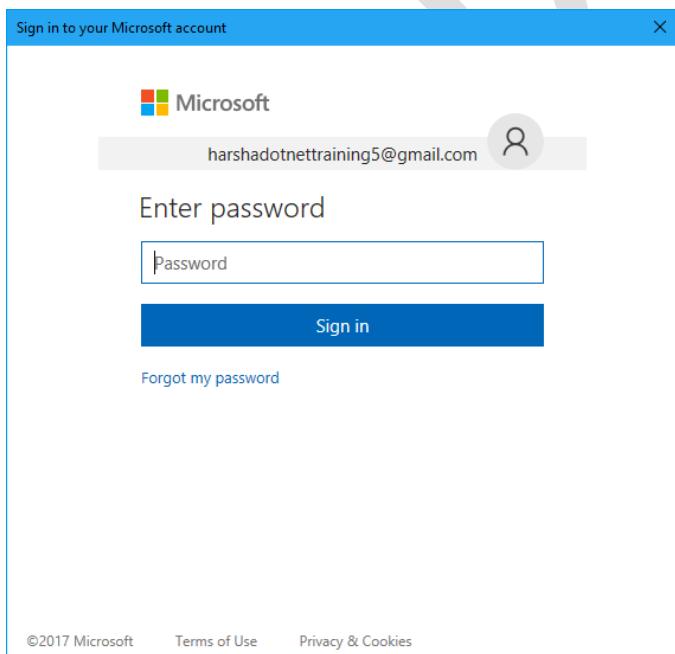
If you get the above page, you have successfully registered.

Now click on "Sign in" in top right corner in Visual Studio.

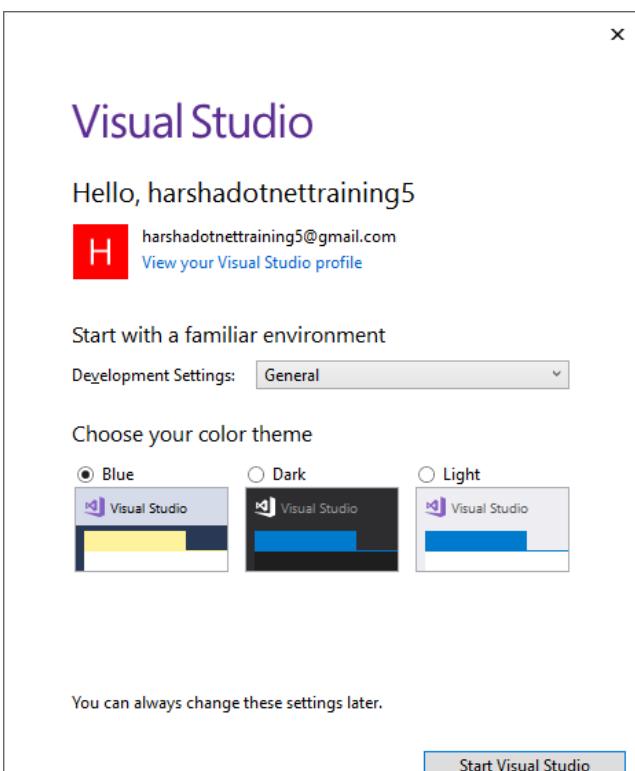


Enter the Email and press Enter key.

Click on "Sign in".



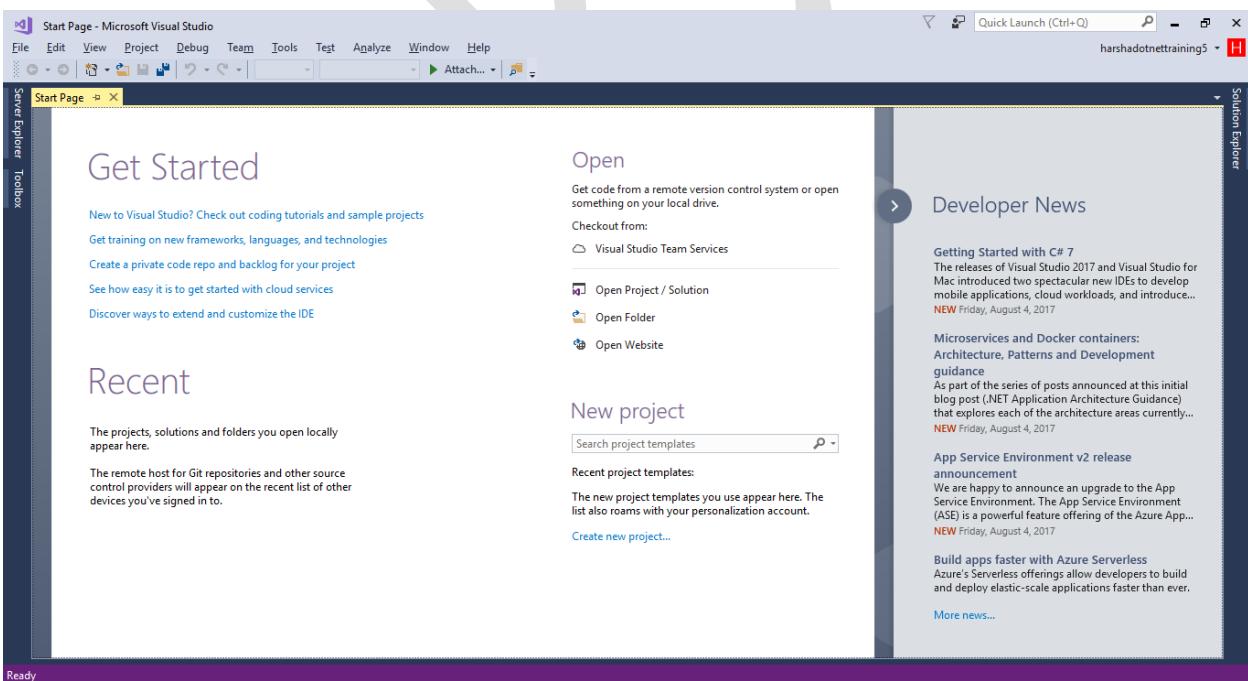
Now enter the password and click on "Sign in".



Select "Development Settings" as "General".

Select "color theme" as "Blue".

Click on "Start Visual Studio".

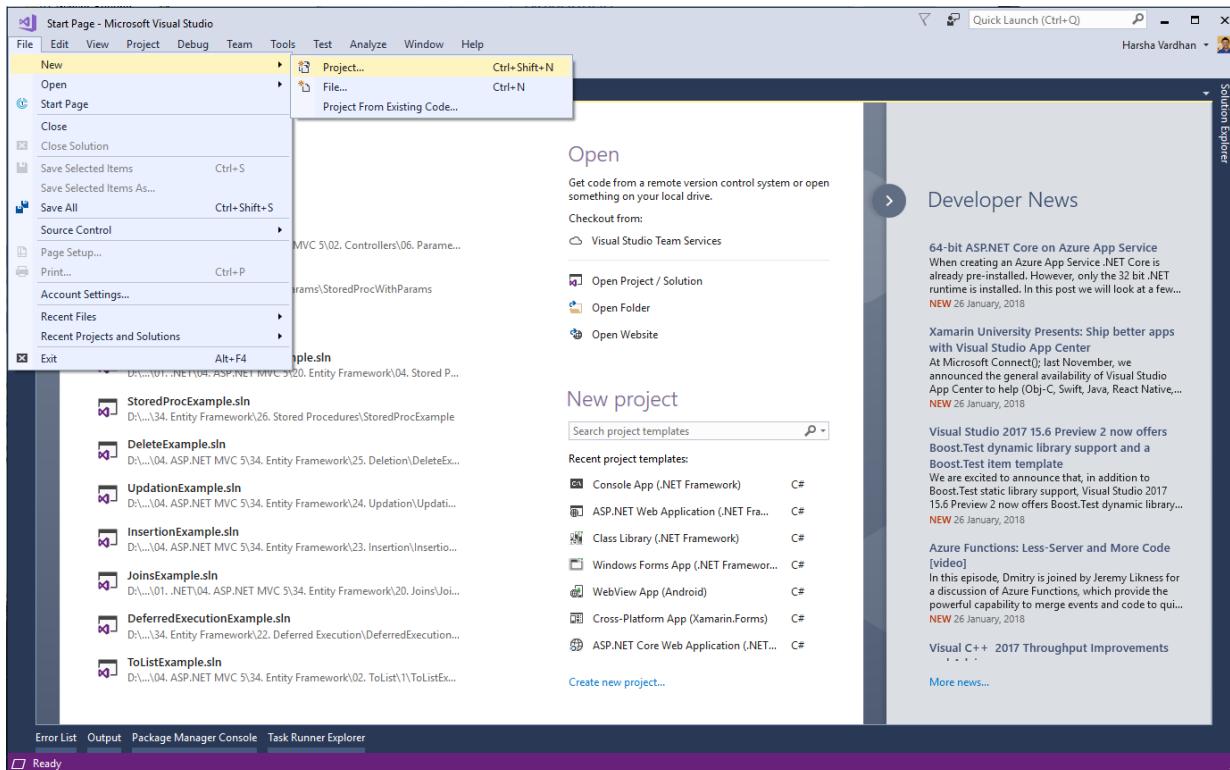


Visual Studio Installation has been completed successfully.

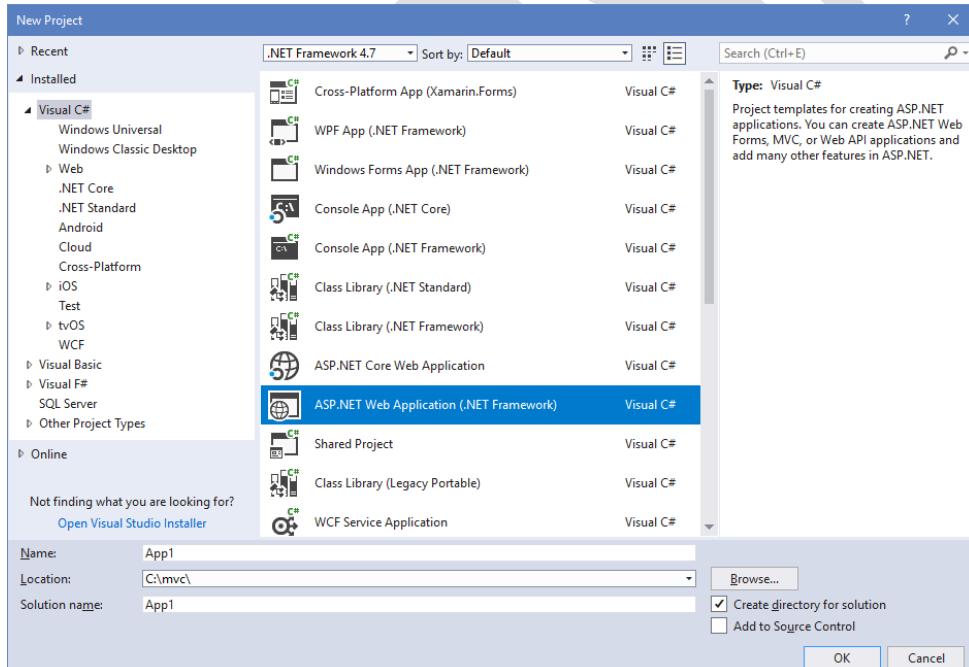
## 2. Creating MVC Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project".

## Asp.Net Mvc 5

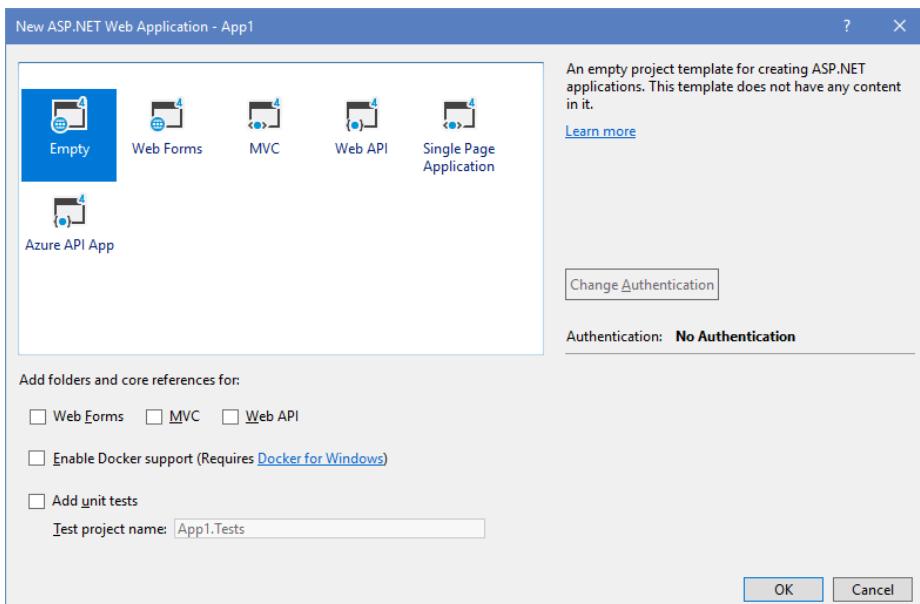


- "New Project" dialogbox appears.

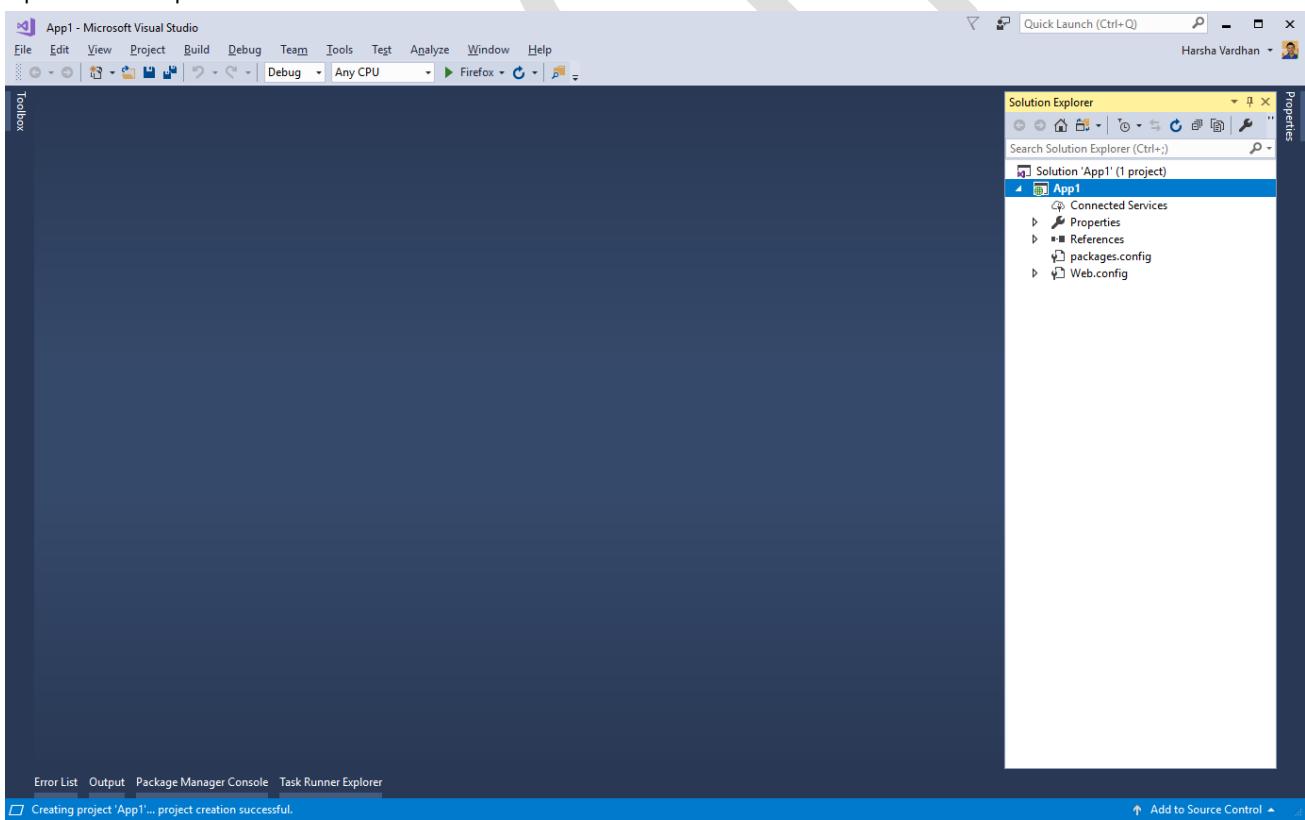


- Select ".NET Framework 4.7".
- Select "Visual C#".
- Select "ASP.NET Web Application (.NET Framework)".
- Type the project name "App1".
- Type the location as "C:\mvc".
- Type the solution name as "App1".
- Click on OK.

## Asp.Net Mvc 5

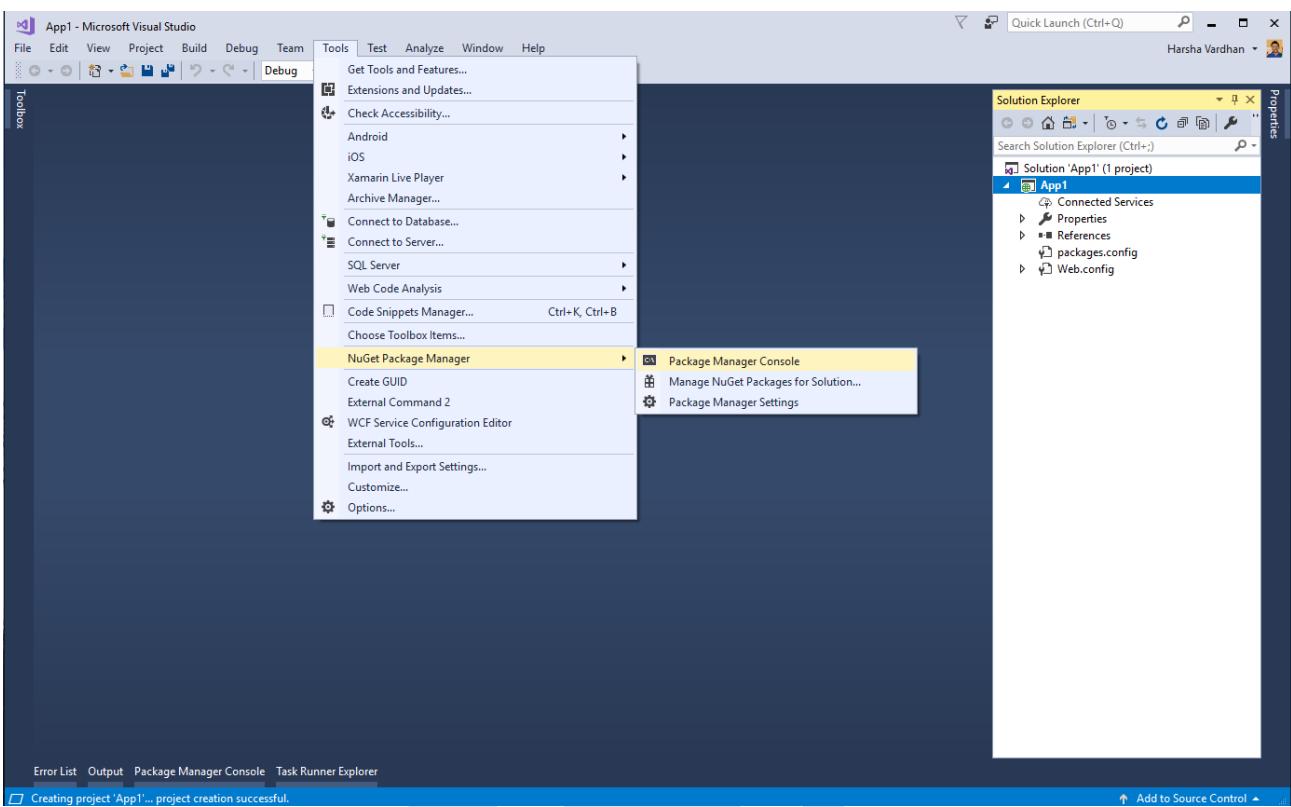


- Select "Empty".
- Uncheck all the checkboxes (in case of any selected).
- Click on OK.
- Open Solution Explorer.

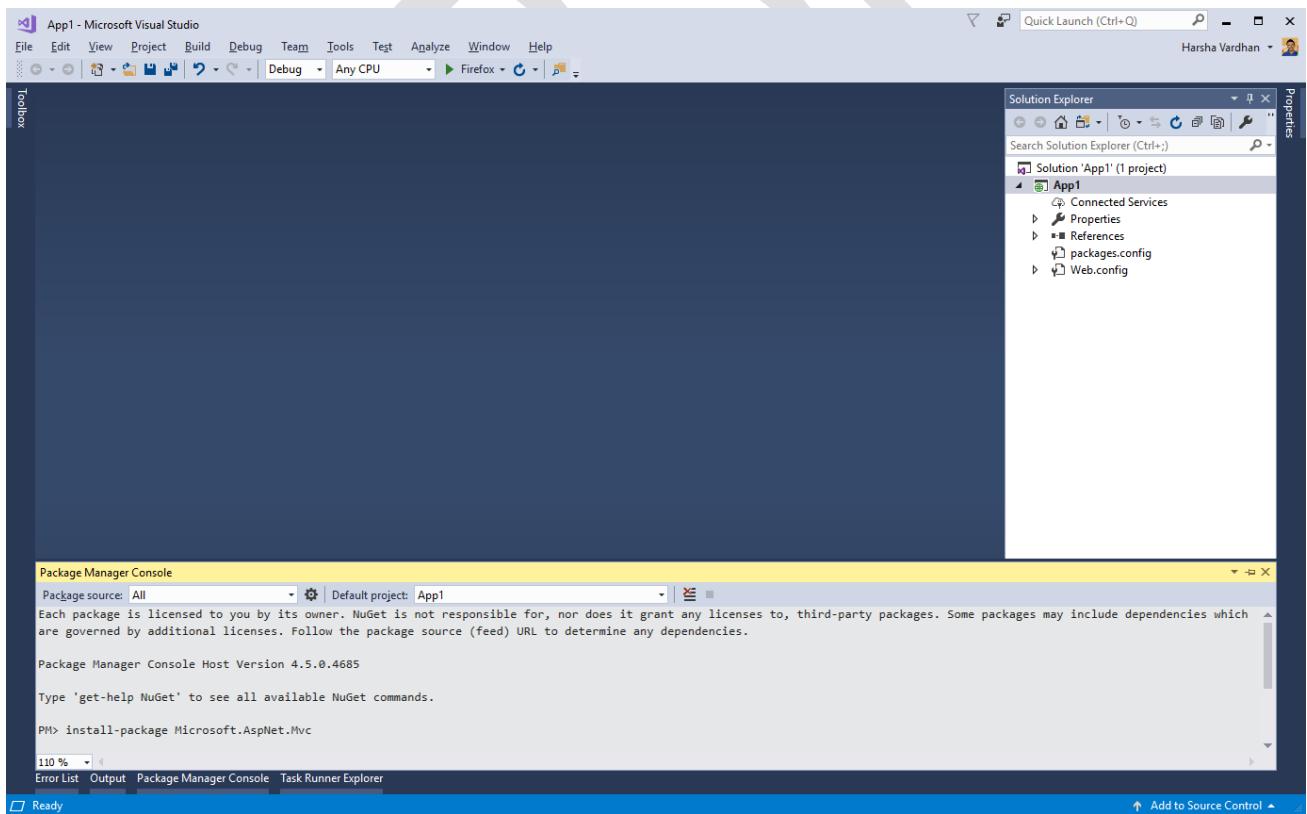


- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".

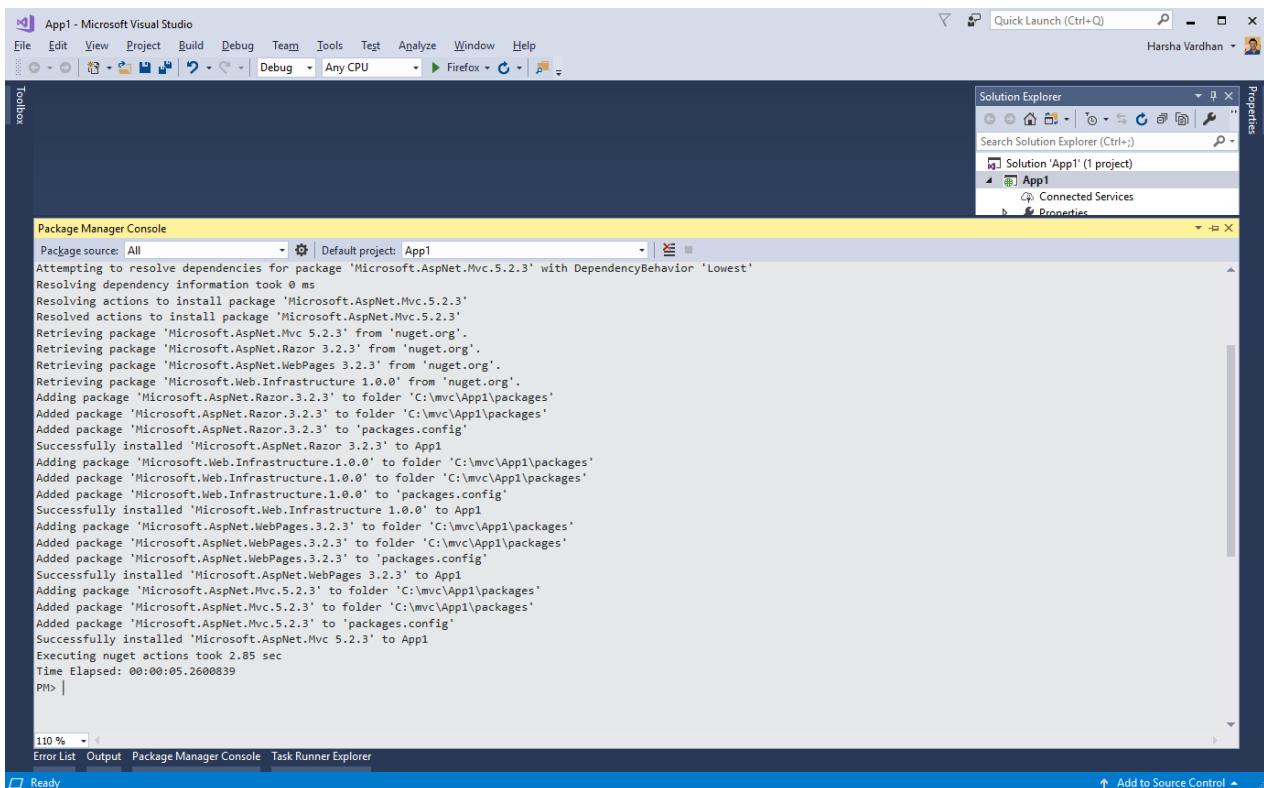
## Asp.Net Mvc 5



- "Package Manager Console" window will be opened.



- Type the following command in "Package Manager Console" and press Enter.  
**install-package Microsoft.AspNet.Mvc**



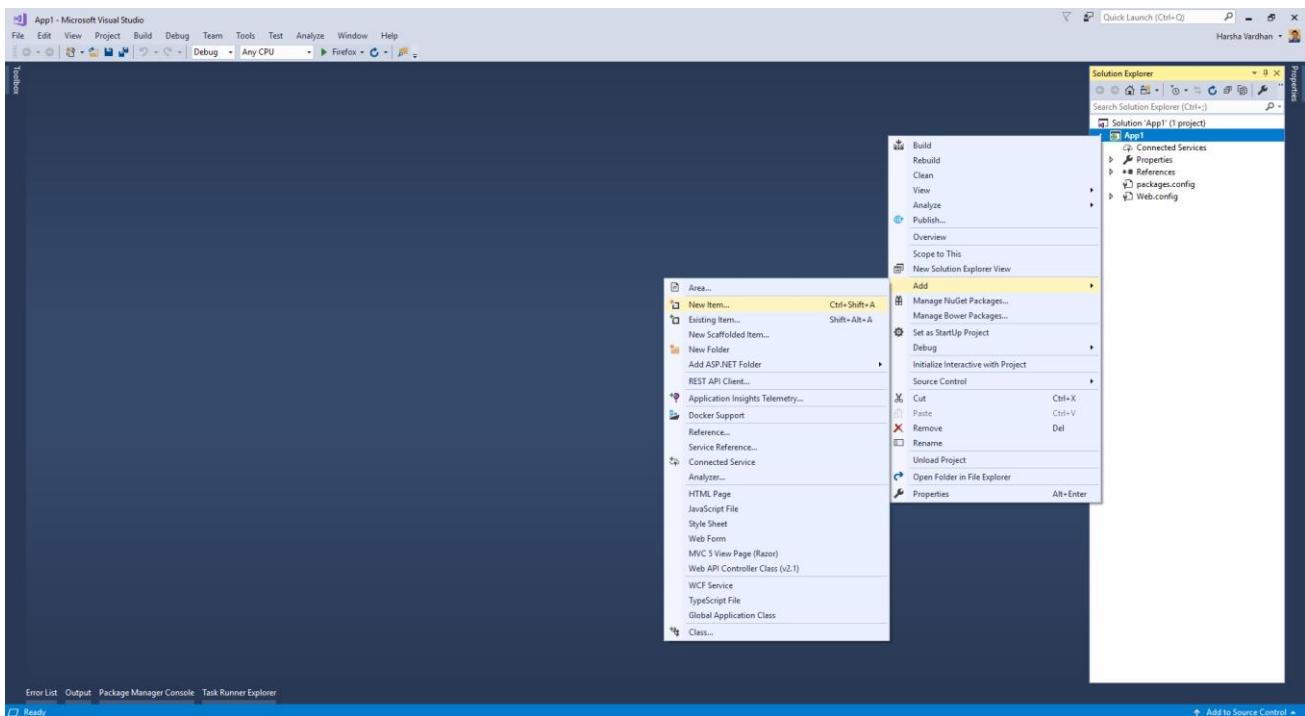
- The following code will be automatically generated for "packages.config".

### Code for "packages.config" (Automatically generated)

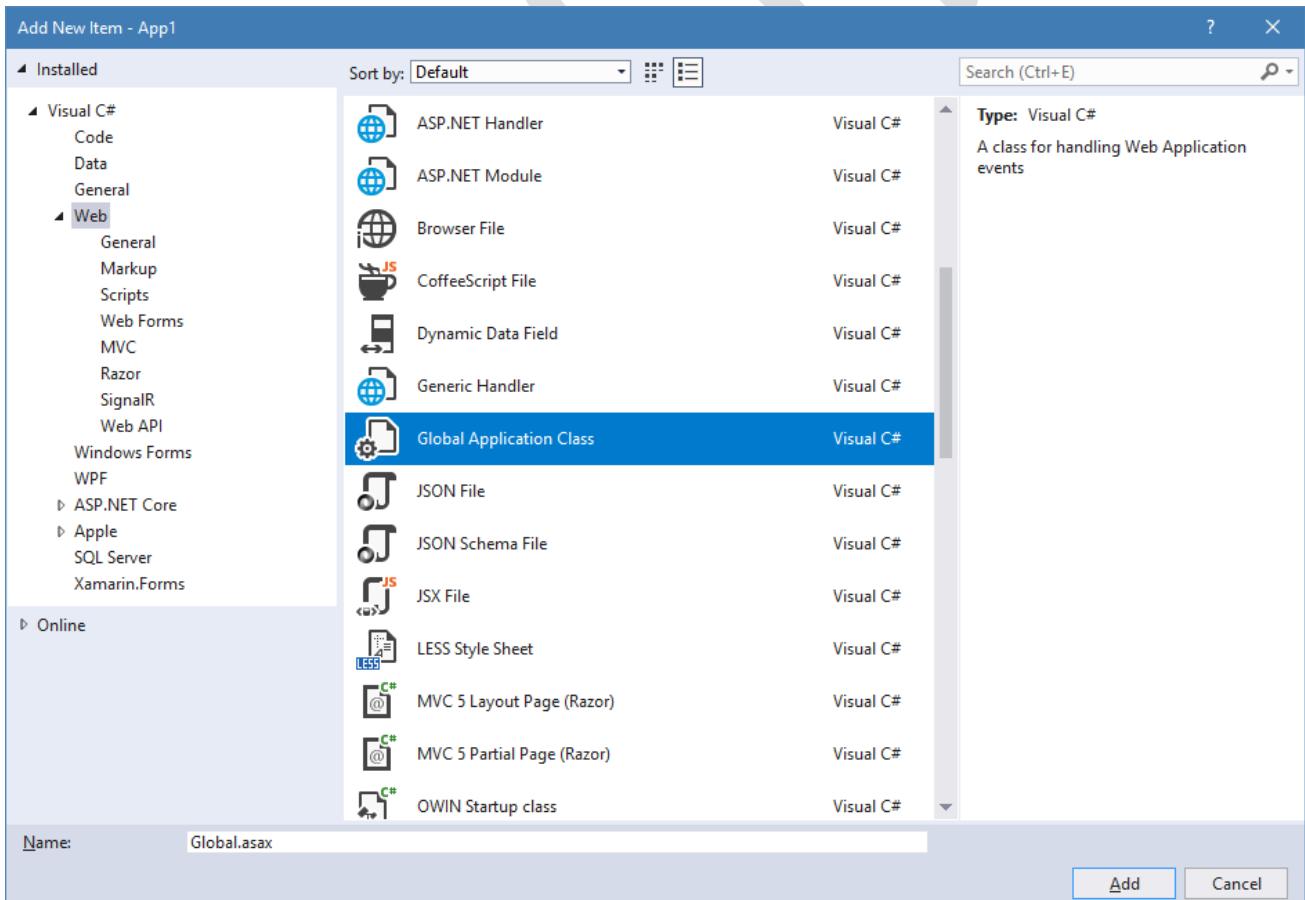
```
<?xml version="1.0" encoding="utf-8"?>
<packages>
  <package id="Microsoft.AspNet.Mvc" version="5.2.3" targetFramework="net47" />
  <package id="Microsoft.AspNet.Razor" version="3.2.3" targetFramework="net47" />
  <package id="Microsoft.AspNet.WebPages" version="3.2.3" targetFramework="net47" />
  <package id="Microsoft.CodeDom.Providers.DotNetCompilerPlatform" version="1.0.7" targetFramework="net47" />
  <package id="Microsoft.Net.Compilers" version="2.1.0" targetFramework="net47" developmentDependency="true" />
  <package id="Microsoft.Web.Infrastructure" version="1.0.0.0" targetFramework="net47" />
</packages>
```

- Right click on the project (App1) and click on "Add" - "New Item".

## Asp.Net Mvc 5



- Click on "Web" - "Global Application Class". Filename is "Global.asax" (automatically displayed). Click on "Add".



- It generates several methods like "Application\_Start()", "Session\_Start()", "Application\_Error()" etc.
- Replace the code for "Global.asax" file as follows:

### Code for "Global.asax"

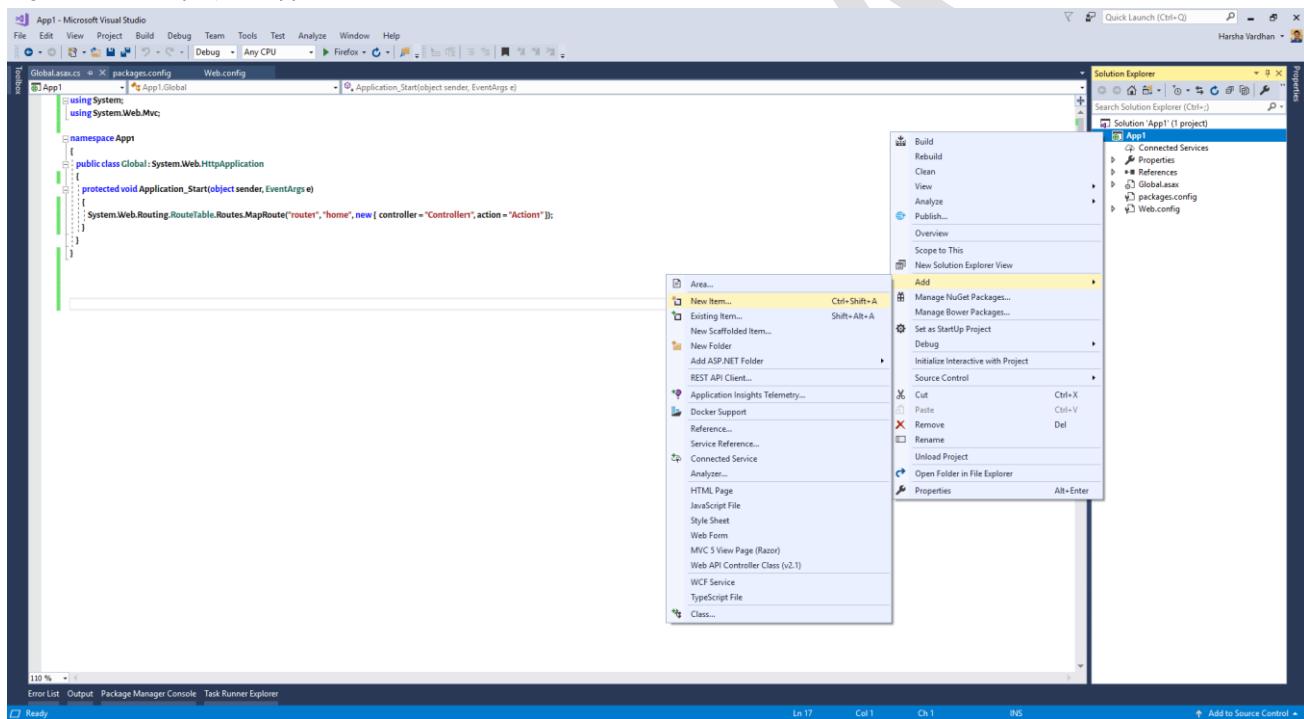
```

using System;
using System.Web.Mvc;

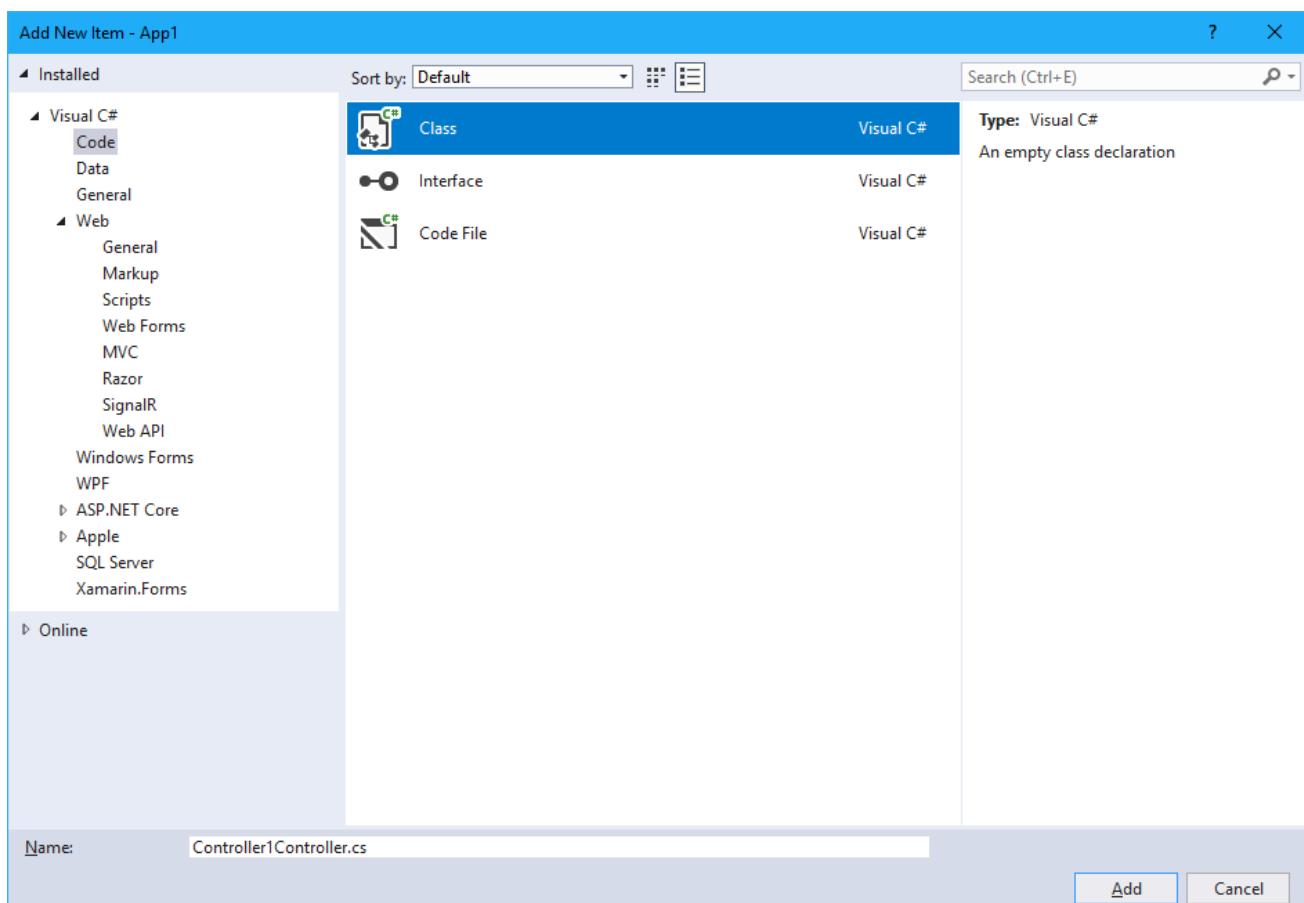
namespace App1
{
    public class Global : System.Web.HttpApplication
    {
        protected void Application_Start(object sender, EventArgs e)
        {
            System.Web.Routing.RouteTable.Routes.MapRoute("route1", "home", new { controller = "Controller1", action = "Action1" });
        }
    }
}

```

- Right click on the project (App1) and click on "Add" - "New Item".



- Click on "Code" - "Class". Filename is "Controller1Controller.cs". Click on "Add".

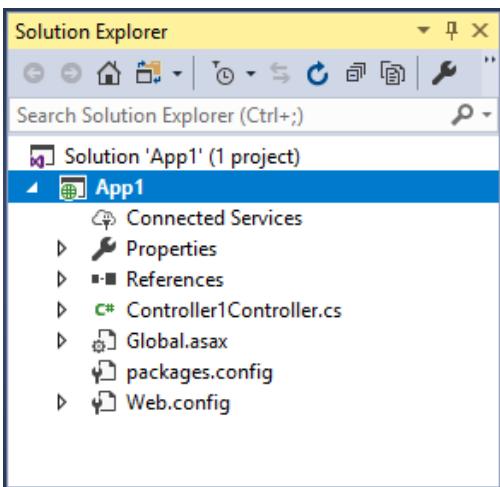


- Replace the code for "Controller1Controller.cs" file as follows:

### Code for "Controller1Controller.cs"

```
using System;
using System.Web.Mvc;

namespace App1
{
    public class Controller1Controller : Controller
    {
        public string Action1()
        {
            return "Hello World";
        }
    }
}
```



### 3. Compile the MVC Project

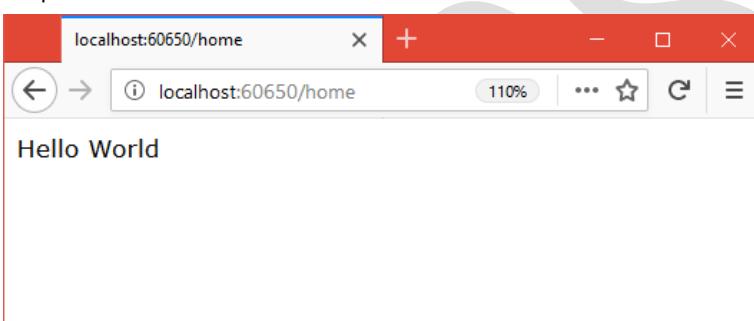
- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B". The project will be compiled.

### 4. Run the MVC Project

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".
- Type "<http://localhost:portnumber/home>".

Here "60650" is an example of port no. The port no will be automatically generated by visual studio. It may vary.

#### Output:



## CONTROLLERS

### What is Controller

- Controller is a class, which receives HTTP request from the browser and sends HTTP response to the browser. Execution flow starts and ends with controller only. Controller is a collection of action methods. Action method performs a specific operation. When the browser sends a HTTP request, "ASP.NET MVC" automatically creates an object for the controller class and calls appropriate action method.
- Controller mediates between model and view. Controller calls model. That means controller creates object for model class & adds data into it. Controller calls view. Controller supplies data to the view. A project can have multiple controllers. Controllers are present in "Controllers" folder. By default, the controller class exists in "Projectname.Controllers" namespace. The controller class's name must be suffix with a word "Controller". Ex: HomeController. The controller class should be a child class of a pre-defined class called "System.Web.Mvc.Controller". The "System.Web.Mvc.Controller" class provides a set of properties and methods that are

### Syntax of Controller

```
using System.Web.Mvc;
namespace Projectname
{
    public class ControllernameController : Controller
    {
        Action methods here
    }
}
```

## Action Methods

- A controller is a collection of "action methods". All the methods present within the controller class are treated as action methods. An "action method" represents an "operation" that can be performed based on the controller.

### Syntax of Action method

```
public returnType Actionname ()  
{  
    return some value;  
}
```

## URL ROUTING

### What is URL Routing

- The process of mapping between "url" and "controller" is called as "Routing". Routing is done with "MapRoute" method.

### Syntax of URL Routing

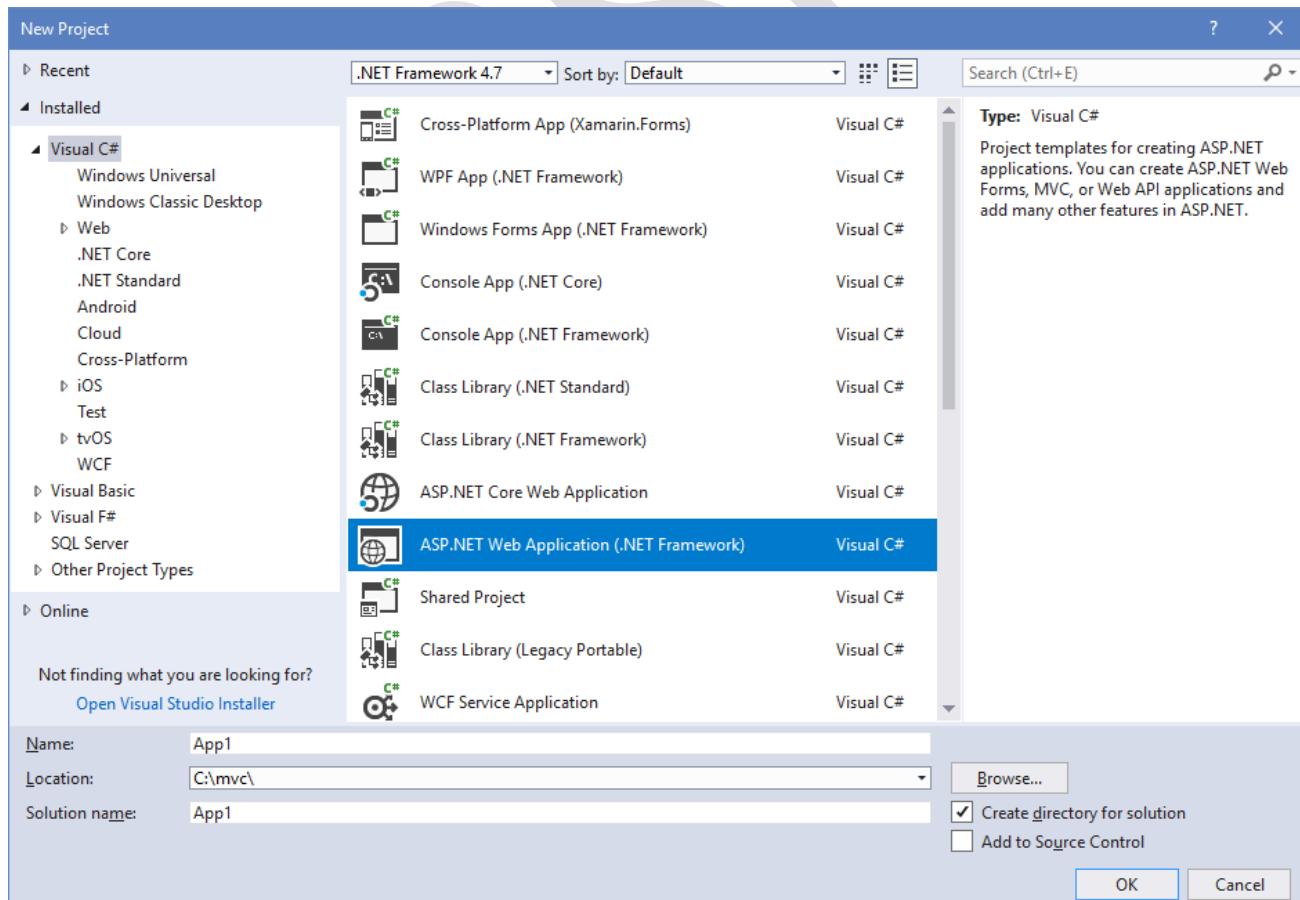
**Syntax:** System.Web.Routing.RouteTable.Routes.MapRoute("route name", "url", new { controller = "controller name", action = "action name" });

**Example:** System.Web.Routing.RouteTable.Routes.MapRoute("route1", "home", new { controller = "controller1", action = "action1" });

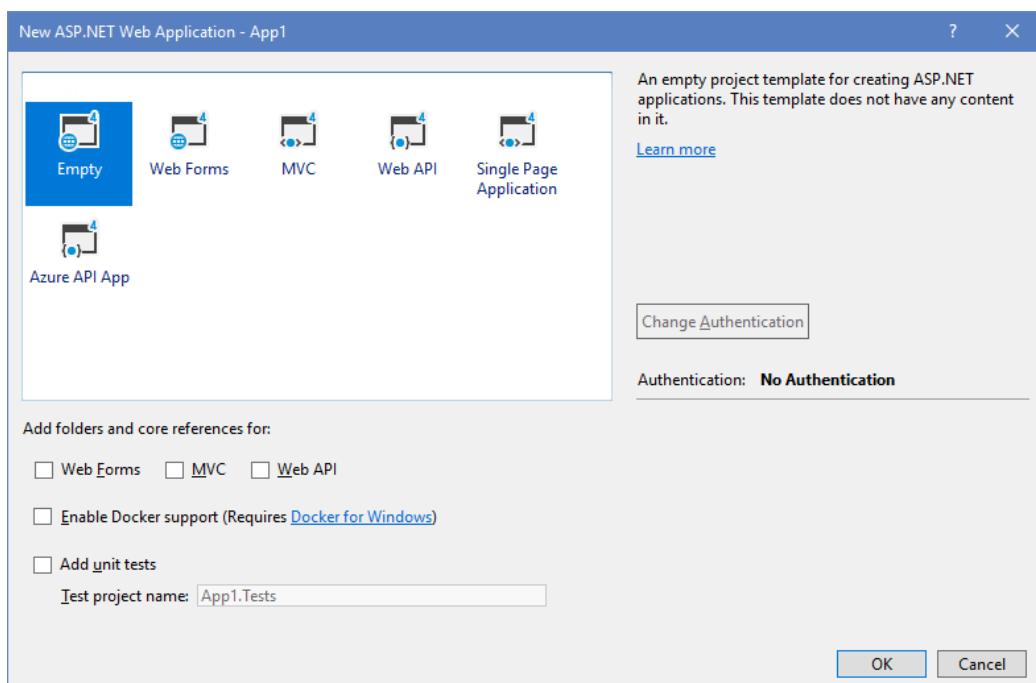
## Multiple Routes - Example

### Creating MVC Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears.



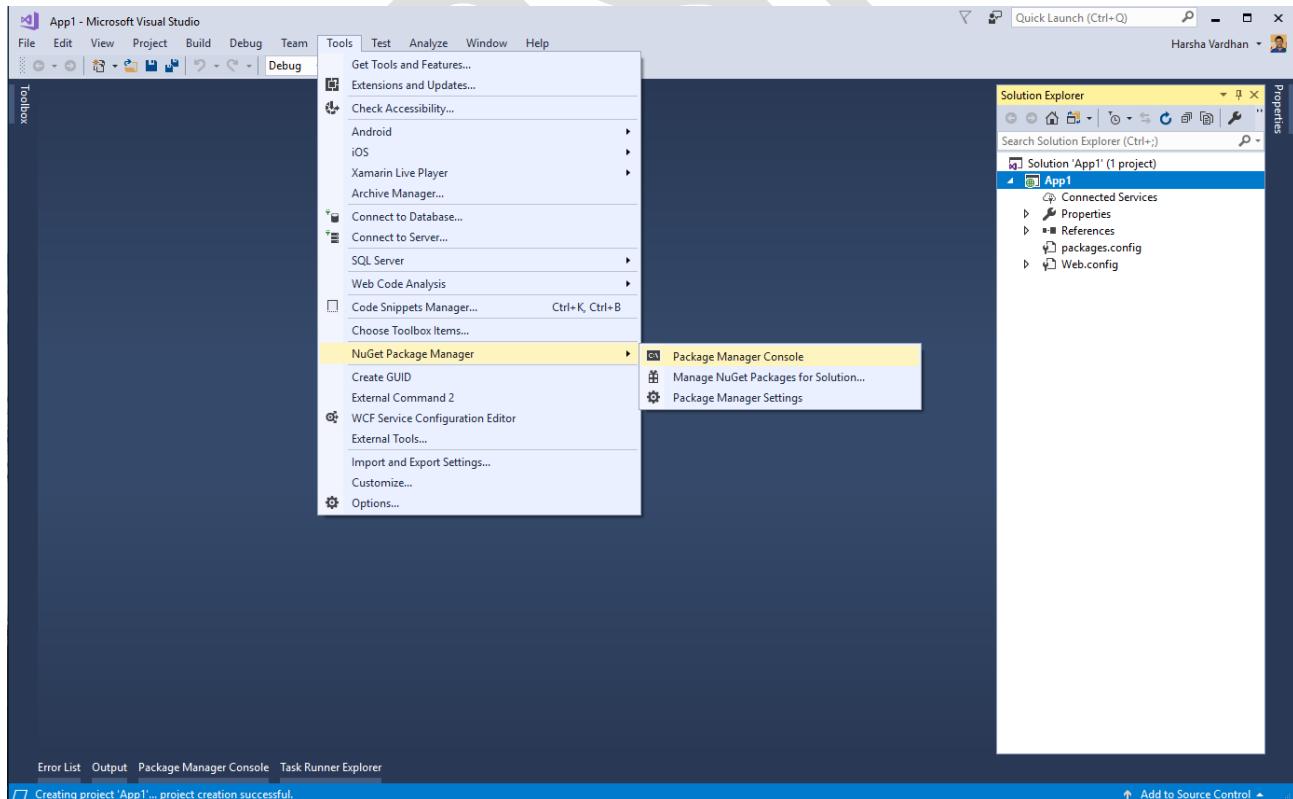
- Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "App1". Type the location as "C:\mvc". Type the solution name as "App1". Click on OK.



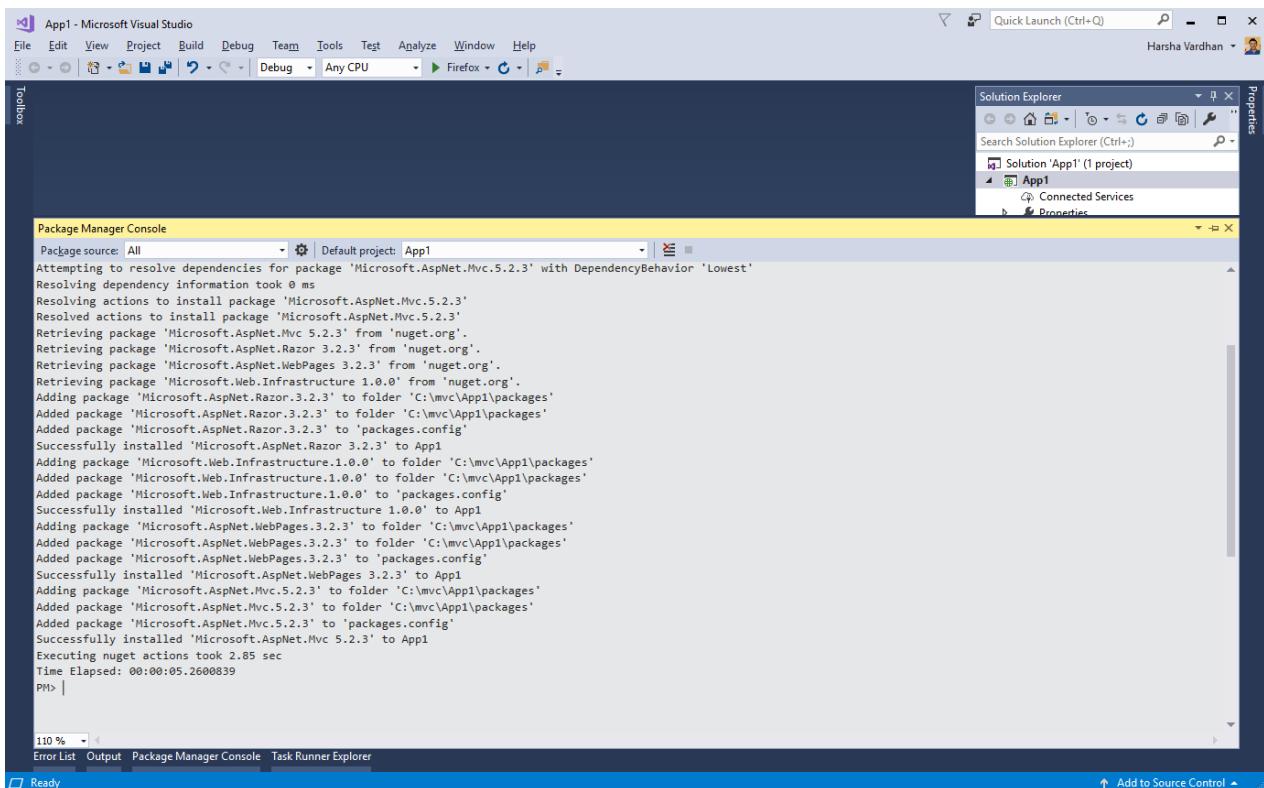
- Select "Empty". Uncheck all the checkboxes (in case of any selected). Click on OK.

### **Installing Packages**

- Open Solution Explorer. Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".



- "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.  
`install-package Microsoft.AspNet.Mvc`



- The following code will be automatically generated for "packages.config".

### Code for "packages.config" (Automatically generated)

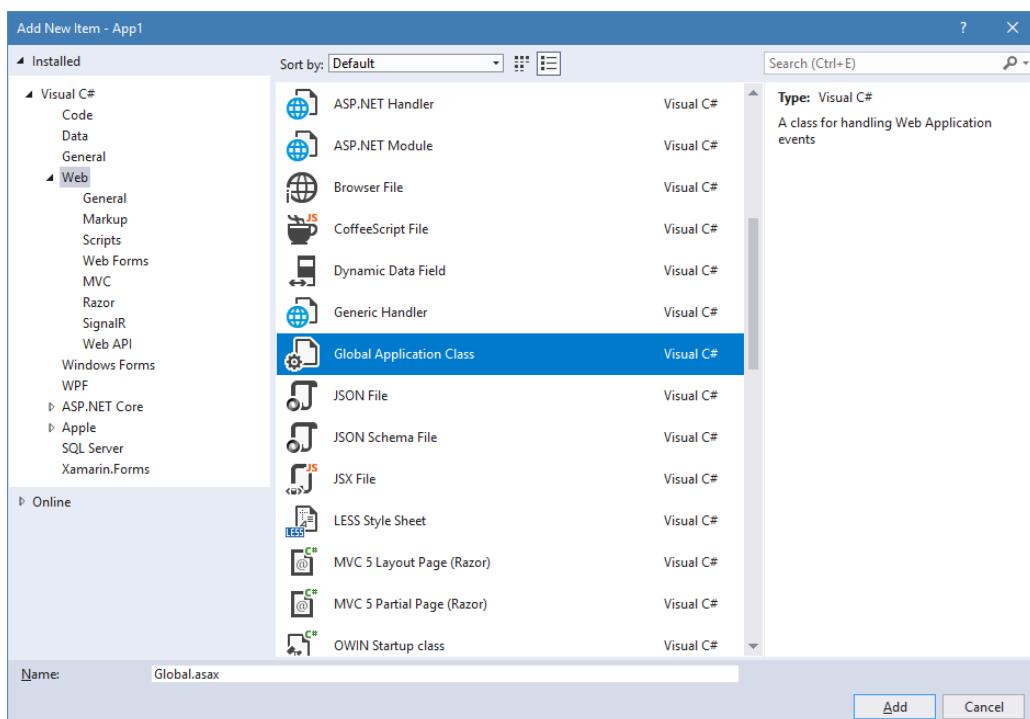
```

<?xml version="1.0" encoding="utf-8"?>
<packages>
  <package id="Microsoft.AspNet.Mvc" version="5.2.3" targetFramework="net47" />
  <package id="Microsoft.AspNet.Razor" version="3.2.3" targetFramework="net47" />
  <package id="Microsoft.AspNet.WebPages" version="3.2.3" targetFramework="net47" />
  <package id="Microsoft.CodeDom.Providers.DotNetCompilerPlatform" version="1.0.7" targetFramework="net47" />
  <package id="Microsoft.Net.Compilers" version="2.1.0" targetFramework="net47" developmentDependency="true" />
  <package id="Microsoft.Web.Infrastructure" version="1.0.0.0" targetFramework="net47" />
</packages>

```

### Creating Global.asax

- Right click on the project (App1) and click on "Add" - "New Item". Click on "Web" - "Global Application Class". Filename is "Global.asax" (automatically displayed). Click on "Add".



- It generates several methods like "Application\_Start()", "Session\_Start()", "Application\_Error()" etc. Replace the code for "Global.asax" file as follows:

### **Code for "Global.asax"**

```

using System;
using System.Web.Mvc;

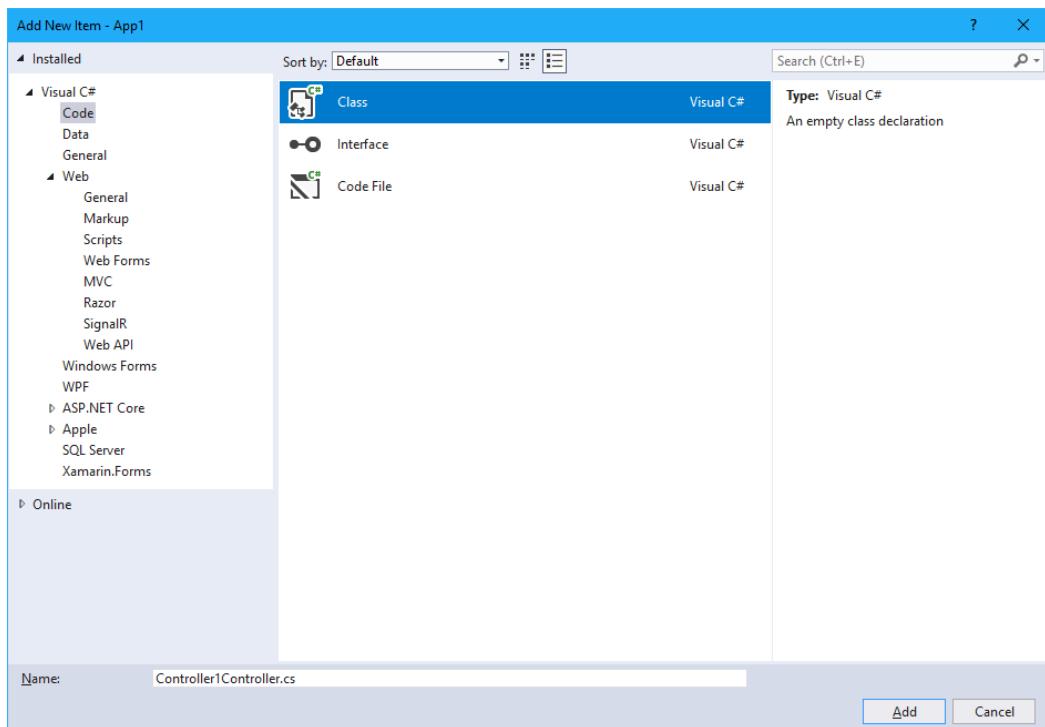
namespace App1
{
    public class Global : System.Web.HttpApplication
    {

        protected void Application_Start(object sender, EventArgs e)
        {
            System.Web.Routing.RouteTable.Routes.MapRoute("route1", "home", new { controller = "Controller1", action = "Action1" });
            System.Web.Routing.RouteTable.Routes.MapRoute("route2", "about", new { controller = "Controller1", action = "Action2" });
            System.Web.Routing.RouteTable.Routes.MapRoute("route3", "contact", new { controller = "Controller1", action = "Action3" });
            System.Web.Routing.RouteTable.Routes.MapRoute("route4", "products", new { controller = "Controller2", action = "Action4" });
            System.Web.Routing.RouteTable.Routes.MapRoute("route5", "services", new { controller = "Controller2", action = "Action5" });
        }
    }
}

```

### **Creating Controller1Controller.cs**

- Right click on the project (App1) and click on "Add" - "New Item". Click on "Code" - "Class". Filename is "Controller1Controller.cs". Click on "Add".



- Replace the code for "Controller1Controller.cs" file as follows:

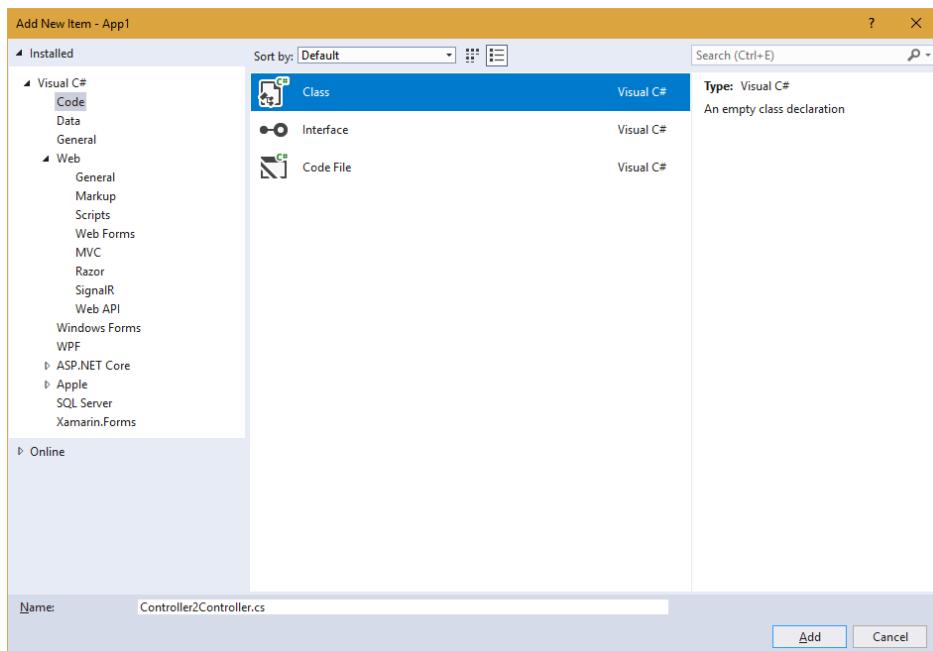
### **Code for "Controller1Controller.cs"**

```
using System;
using System.Web.Mvc;

namespace App1
{
    public class Controller1Controller : Controller
    {
        public string Action1()
        {
            return "Home Page Here";
        }
        public string Action2()
        {
            return "About Page Here";
        }
        public string Action3()
        {
            return "Contact Page Here";
        }
    }
}
```

### **Creating Controller2Controller.cs**

- Right click on the project (App1) and click on "Add" - "New Item". Click on "Code" - "Class". Filename is "Controller2Controller.cs". Click on "Add".

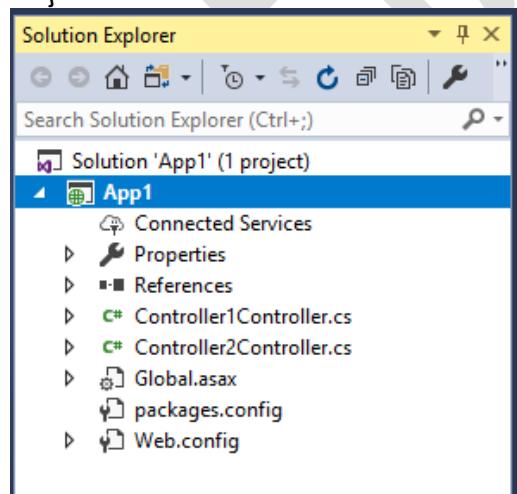


- Replace the code for "Controller2Controller.cs" file as follows:

### Code for "Controller2Controller.cs"

```
using System;
using System.Web.Mvc;

namespace App1
{
    public class Controller2Controller : Controller
    {
        public string Action4()
        {
            return "Products Page Here";
        }
        public string Action5()
        {
            return "Services Page Here";
        }
    }
}
```



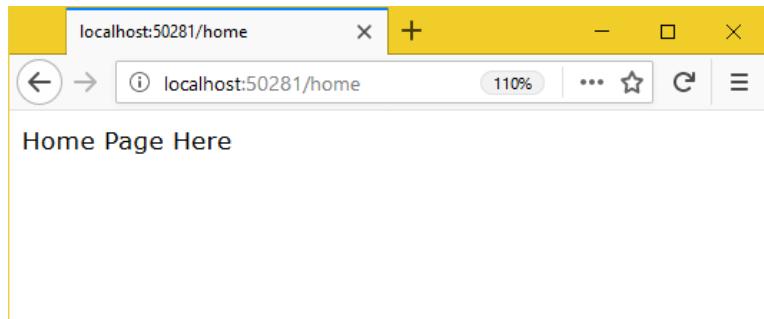
### Compile the MVC Project

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B". The project will be compiled.

### Run the MVC Project

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".
- Type "<http://localhost:50281/home>".
- **Note:** The port number may vary.

### Output:



Also try:

<http://localhost:50281/about>  
<http://localhost:50281/contact>  
<http://localhost:50281/products>  
<http://localhost:50281/services>

## Generic URL Routing

- It is used to accept controller name and action name in the url at the browser.

### Syntax of Generic URL Routing

**Syntax:** System.Web.Routing.RouteTable.Routes.MapRoute( "route name", "{controller}/{action}" );

**Example:** System.Web.Routing.RouteTable.Routes.MapRoute( "route1", "{controller}/{action}" );

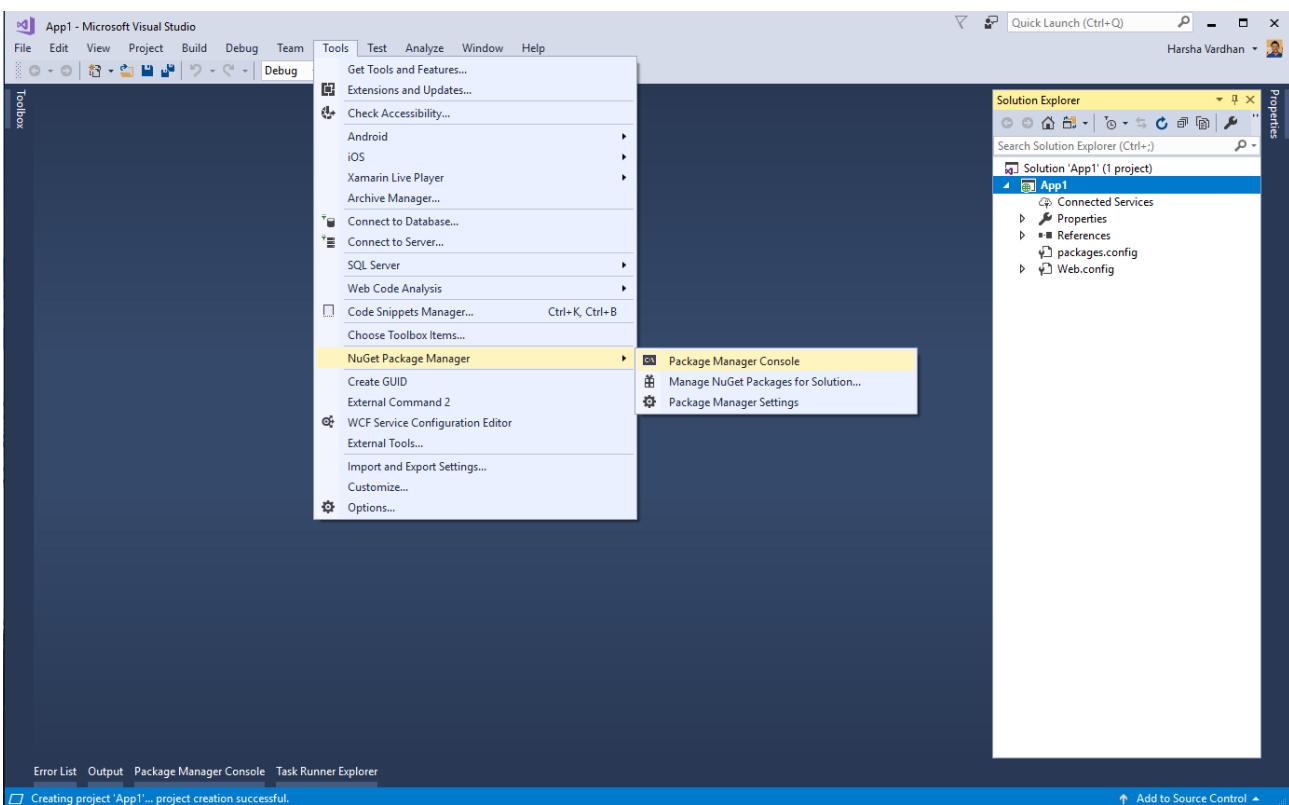
## Generic Routing - Example

### Creating MVC Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "App1". Type the location as "C:\mvc". Type the solution name as "App1". Click on OK. Select "Empty". Uncheck all the checkboxes (in case of any selected). Click on OK.

### Installing Packages

- Open Solution Explorer. Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".



- "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.  
**install-package Microsoft.AspNet.Mvc**

```

Package Manager Console
Default project: App1
Attempting to resolve dependencies for package 'Microsoft.AspNet.Mvc.5.2.3' with DependencyBehavior 'Lowest'
Resolving dependency information took 0 ms
Resolving actions to install package 'Microsoft.AspNet.Mvc.5.2.3'
Resolved actions to install package 'Microsoft.AspNet.Mvc.5.2.3'
Retrieving package 'Microsoft.AspNet.Mvc.5.2.3' from 'nuget.org'.
Retrieving package 'Microsoft.AspNet.Razor.3.2.3' from 'nuget.org'.
Retrieving package 'Microsoft.AspNet.WebPages.3.2.3' from 'nuget.org'.
Retrieving package 'Microsoft.Web.Infrastructure.1.0.0' from 'nuget.org'.
Adding package 'Microsoft.AspNet.Razor.3.2.3' to folder 'C:\mvc\app1\packages'
Added package 'Microsoft.AspNet.Razor.3.2.3' to folder 'C:\mvc\app1\packages'
Added package 'Microsoft.AspNet.Razor.3.2.3' to 'packages.config'
Successfully installed 'Microsoft.AspNet.Razor.3.2.3' to App1
Adding package 'Microsoft.Web.Infrastructure.1.0.0' to folder 'C:\mvc\app1\packages'
Added package 'Microsoft.Web.Infrastructure.1.0.0' to 'packages.config'
Added package 'Microsoft.Web.Infrastructure.1.0.0' to 'packages.config'
Successfully installed 'Microsoft.Web.Infrastructure.1.0.0' to App1
Adding package 'Microsoft.AspNet.WebPages.3.2.3' to folder 'C:\mvc\app1\packages'
Added package 'Microsoft.AspNet.WebPages.3.2.3' to 'packages.config'
Added package 'Microsoft.AspNet.WebPages.3.2.3' to 'packages.config'
Successfully installed 'Microsoft.AspNet.WebPages.3.2.3' to App1
Adding package 'Microsoft.AspNet.Mvc.5.2.3' to folder 'C:\mvc\app1\packages'
Added package 'Microsoft.AspNet.Mvc.5.2.3' to folder 'C:\mvc\app1\packages'
Added package 'Microsoft.AspNet.Mvc.5.2.3' to 'packages.config'
Successfully installed 'Microsoft.AspNet.Mvc.5.2.3' to App1
Executing nuget actions took 2.85 sec
Time Elapsed: 00:00:05.2600839
PM> |

```

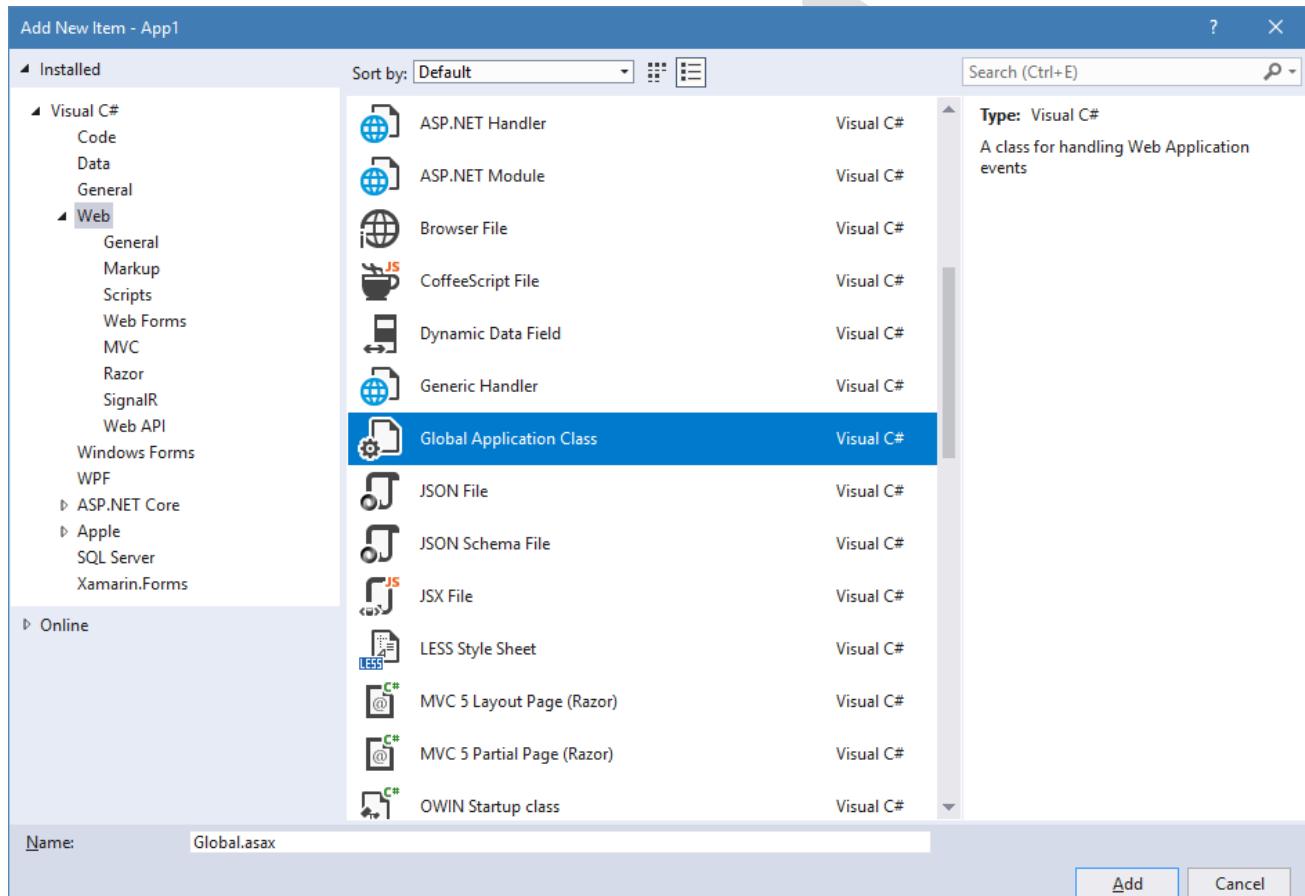
- The following code will be automatically generated for "packages.config".

### Code for "packages.config" (Automatically generated)

```
<?xml version="1.0" encoding="utf-8?>
<packages>
<package id="Microsoft.AspNet.Mvc" version="5.2.3" targetFramework="net47" />
<package id="Microsoft.AspNet.Razor" version="3.2.3" targetFramework="net47" />
<package id="Microsoft.AspNet.WebPages" version="3.2.3" targetFramework="net47" />
<package id="Microsoft.CodeDom.Providers.DotNetCompilerPlatform" version="1.0.7" targetFramework="net47" />
<package id="Microsoft.Net.Compilers" version="2.1.0" targetFramework="net47" developmentDependency="true" />
<package id="Microsoft.Web.Infrastructure" version="1.0.0.0" targetFramework="net47" />
</packages>
```

### Creating Global.asax

- Right click on the project (App1) and click on "Add" - "New Item". Click on "Web" - "Global Application Class". Filename is "Global.asax" (automatically displayed). Click on "Add".



- It generates several methods like "Application\_Start()", "Session\_Start()", "Application\_Error()" etc. Replace the code for "Global.asax" file as follows:

### Code for "Global.asax"

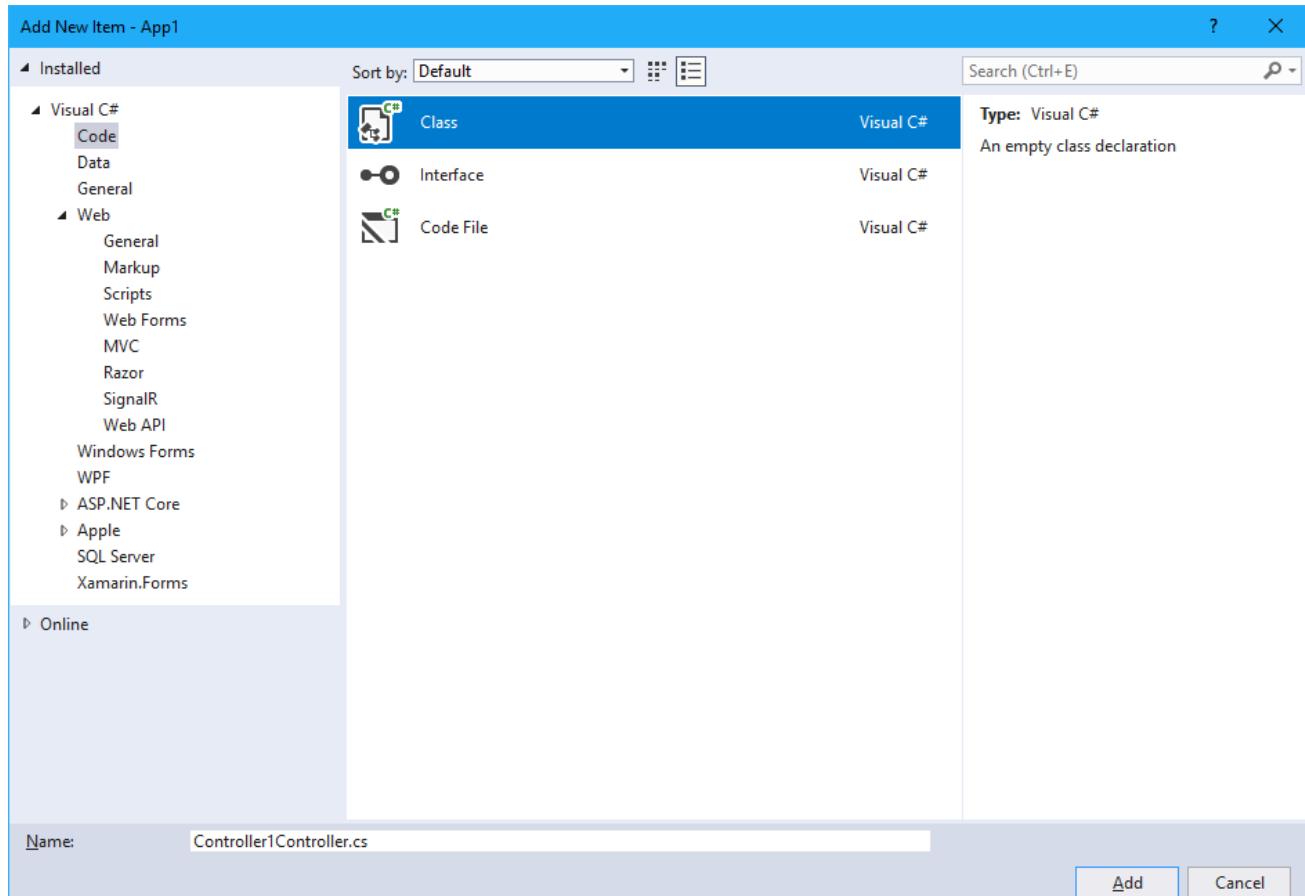
```
using System;
using System.Web.Mvc;

namespace App1
{
    public class Global : System.Web.HttpApplication
    {
        protected void Application_Start(object sender, EventArgs e)
        {
            System.Web.Routing.RouteTable.Routes.MapRoute("route1", "{controller}/{action}");
        }
    }
}
```

}

### **Creating Controller1Controller.cs**

- Right click on the project (App1) and click on "Add" - "New Folder". Type the folder name as "Controllers" and press Enter. Right click on the "Controllers" folder and click on "Add" - "New Item". Click on "Code" - "Class". Filename is "Controller1Controller.cs". Click on "Add".



- Replace the code for "Controller1Controller.cs" file as follows:

### **Code for "Controller1Controller.cs"**

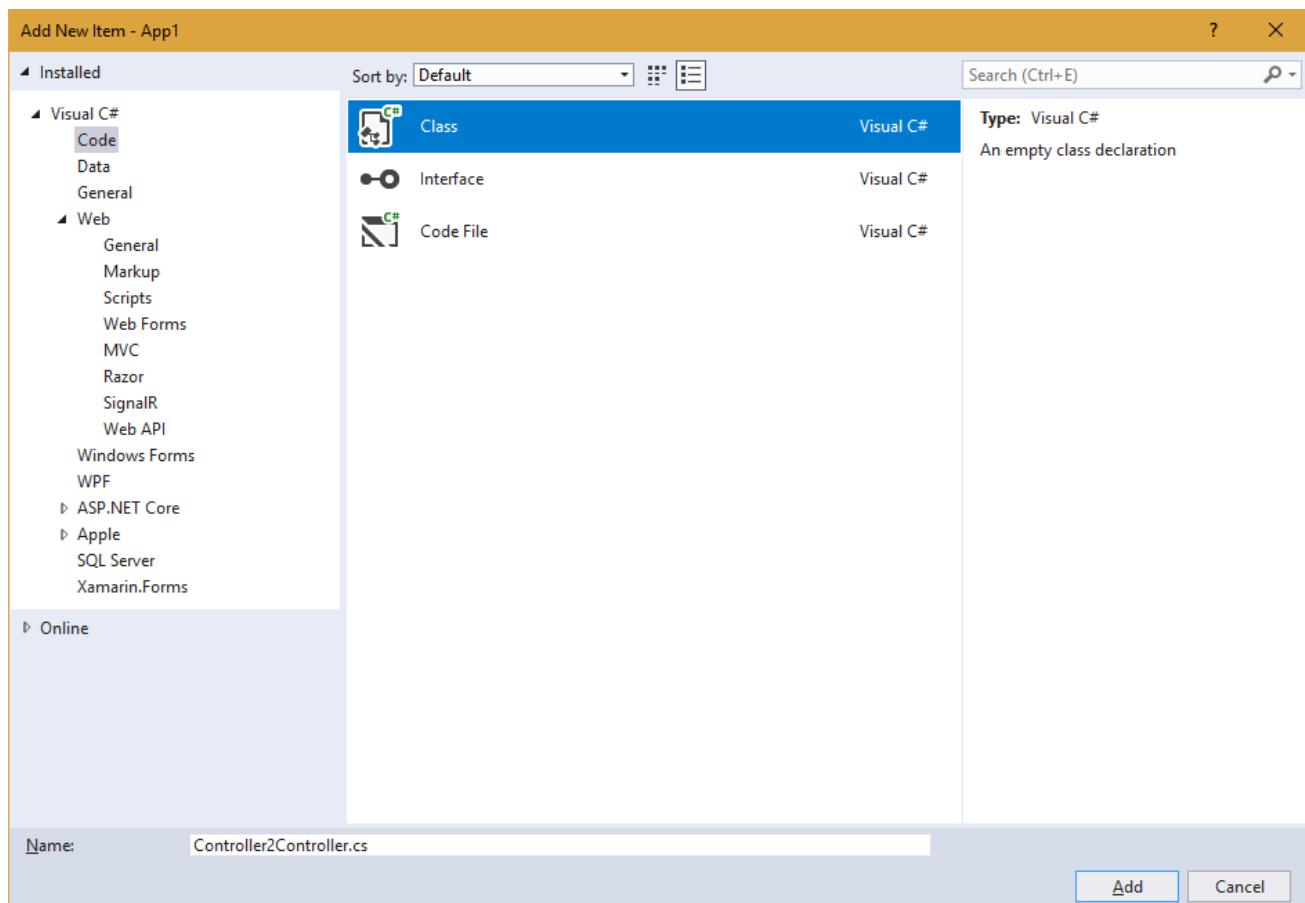
```
using System;
using System.Web.Mvc;

namespace App1.Controllers
{
    public class Controller1Controller : Controller
    {
        public string Action1()
        {
            return "Home Page Here";
        }
        public string Action2()
        {
            return "About Page Here";
        }
        public string Action3()
        {
            return "Contact Page Here";
        }
    }
}
```

}

### **Creating Controller2Controller.cs**

- Right click on the "Controllers" folder and click on "Add" - "New Item". Click on "Code" - "Class". Filename is "Controller2Controller.cs". Click on "Add".



- Replace the code for "Controller2Controller.cs" file as follows:

### **Code for "Controller2Controller.cs"**

```
using System;
using System.Web.Mvc;

namespace App1.Controllers
{
    public class Controller2Controller : Controller
    {
        public string Action4()
        {
            return "Products Page Here";
        }
        public string Action5()
        {
            return "Services Page Here";
        }
    }
}
```

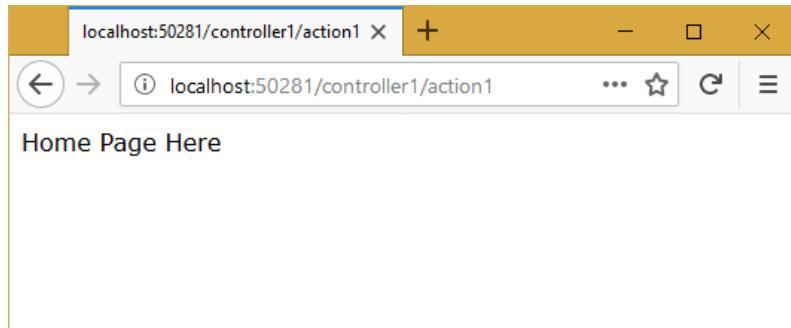
### **Compile the MVC Project**

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B". The project will be compiled.

### Run the MVC Project

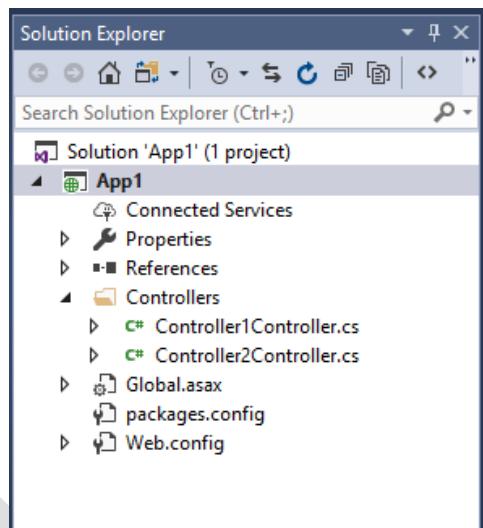
- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5". Type "<http://localhost:50281/controller1/action1>".
- Note:** The port number may vary.

#### Output:



Also try:

<http://localhost:50281/controller1/action1>  
<http://localhost:50281/controller1/action2>  
<http://localhost:50281/controller1/action3>  
<http://localhost:50281/controller2/action4>  
<http://localhost:50281/controller2/action5>



## Route Parameters

- It is used to accept values in the url at the browser.

### Syntax of Route Parameters

**Syntax:** System.Web.Routing.RouteTable.Routes.MapRoute( "route name", "{parameter1}/{parameter2}..." );

**Example:** System.Web.Routing.RouteTable.Routes.MapRoute( "route1", "{controller}/{action}/{id}" );

## Route Parameters - Example

### Creating MVC Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "App1". Type the location as "C:\mvc". Type the solution name as "App1". Click on OK. Select "Empty". Uncheck all the checkboxes (in case of any selected). Click on OK.

### Installing Packages

- Open Solution Explorer. Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.  
**install-package Microsoft.AspNet.Mvc**
- The following code will be automatically generated for "packages.config".

### Code for "packages.config" (Automatically generated)

```
<?xml version="1.0" encoding="utf-8"?>
<packages>
  <package id="Microsoft.AspNet.Mvc" version="5.2.3" targetFramework="net47" />
  <package id="Microsoft.AspNet.Razor" version="3.2.3" targetFramework="net47" />
  <package id="Microsoft.AspNet.WebPages" version="3.2.3" targetFramework="net47" />
  <package id="Microsoft.CodeDom.Providers.DotNetCompilerPlatform" version="1.0.7" targetFramework="net47" />
  <package id="Microsoft.Net.Compilers" version="2.1.0" targetFramework="net47" developmentDependency="true" />
  <package id="Microsoft.Web.Infrastructure" version="1.0.0.0" targetFramework="net47" />
</packages>
```

### **Creating Global.asax**

- Right click on the project (App1) and click on "Add" - "New Item". Click on "Web" - "Global Application Class". Filename is "Global.asax" (automatically displayed). Click on "Add". It generates several methods like "Application\_Start()", "Session\_Start()", "Application\_Error()" etc. Replace the code for "Global.asax" file as follows:

### **Code for "Global.asax"**

```

using System;
using System.Web.Mvc;

namespace App1
{
    public class Global : System.Web.HttpApplication
    {
        protected void Application_Start(object sender, EventArgs e)
        {
            System.Web.Routing.RouteTable.Routes.MapRoute("route1", "{controller}/{action}/{category}");
        }
    }
}

```

### **Creating ProductsController.cs**

- Right click on the project (App1) and click on "Add" - "New Folder". Type the folder name as "Controllers" and press Enter. Right click on the "Controllers" folder and click on "Add" - "New Item". Click on "Code" - "Class". Filename is "ProductsController.cs". Click on "Add". Replace the code for "ProductsController.cs" file as follows:

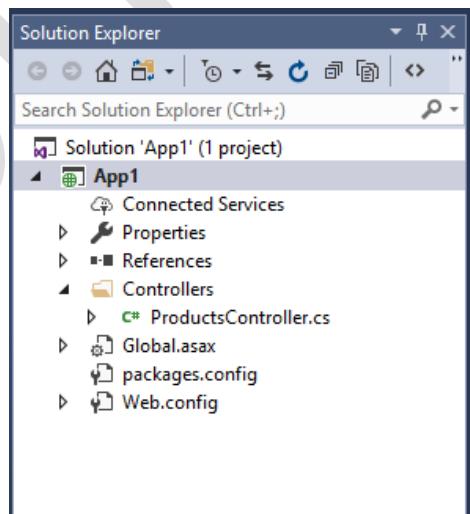
### **Code for "ProductsController.cs"**

```

using System;
using System.Web.Mvc;

namespace App1.Controllers
{
    public class ProductsController : Controller
    {
        public string Show(string Category)
        {
            return "Category: " + Category;
        }
    }
}

```



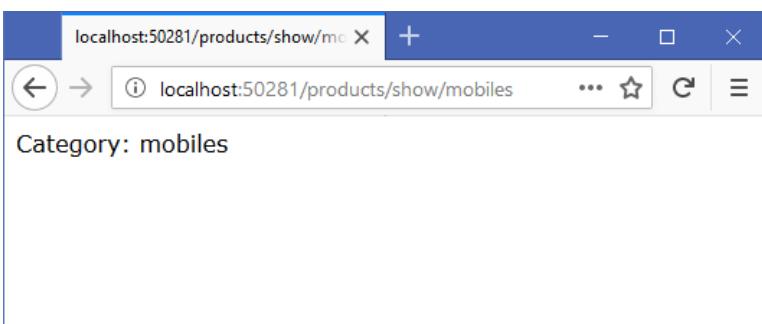
### **Compile the MVC Project**

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B". The project will be compiled.

### **Run the MVC Project**

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".
- Type "<http://localhost:50281/products/show/mobiles>".
- Note:** The port number may vary.

Output:



### Optional Parameters

- This concept is used to make the parameter as optional. That means the user may / may not pass value for the parameter. If the user has not passed the value, it takes "null" value if the parameter data type is "string", "0" (zero) value if the parameter data type is numerical data type.

#### Syntax of Optional Parameters

**Syntax:** System.Web.Routing.RouteTable.Routes.MapRoute( "route\_name", "{parameter1}/{parameter2}/...", new { parameter1 = UrlParameter.Optional, parameter2 = UrlParameter.Optional } );

**Example:** System.Web.Routing.RouteTable.Routes.MapRoute( "route1", "{controller}/{action}/{id}", new { id = UrlParameter.Optional } );

### Optional Parameters - Example

#### Creating MVC Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialog box appears. Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "App1". Type the location as "C:\mvc". Type the solution name as "App1". Click on OK. Select "Empty". Uncheck all the checkboxes (in case of any selected). Click on OK.

#### Installing Packages

- Open Solution Explorer. Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package Microsoft.AspNet.Mvc
```

- The following code will be automatically generated for "packages.config".

#### Code for "packages.config" (Automatically generated)

```
<?xml version="1.0" encoding="utf-8"?>
<packages>
  <package id="Microsoft.AspNet.Mvc" version="5.2.3" targetFramework="net47" />
  <package id="Microsoft.AspNet.Razor" version="3.2.3" targetFramework="net47" />
  <package id="Microsoft.AspNet.WebPages" version="3.2.3" targetFramework="net47" />
  <package id="Microsoft.CodeDom.Providers.DotNetCompilerPlatform" version="1.0.7" targetFramework="net47" />
  <package id="Microsoft.Net.Compilers" version="2.1.0" targetFramework="net47" developmentDependency="true" />
  <package id="Microsoft.Web.Infrastructure" version="1.0.0.0" targetFramework="net47" />
</packages>
```

#### Creating Global.asax

- Right click on the project (App1) and click on "Add" - "New Item". Click on "Web" - "Global Application Class". Filename is "Global.asax" (automatically displayed). Click on "Add". It generates several methods like "Application\_Start()", "Session\_Start()", "Application\_Error()" etc. Replace the code for "Global.asax" file as follows:

#### Code for "Global.asax"

```
using System;
using System.Web.Mvc;

namespace App1
{
    public class Global : System.Web.HttpApplication
    {
```

```
protected void Application_Start(object sender, EventArgs e)
{
    System.Web.Routing.RouteTable.Routes.MapRoute("route1", "{controller}/{action}/{category}", new { category =
UrlParameter.Optional });
}
```

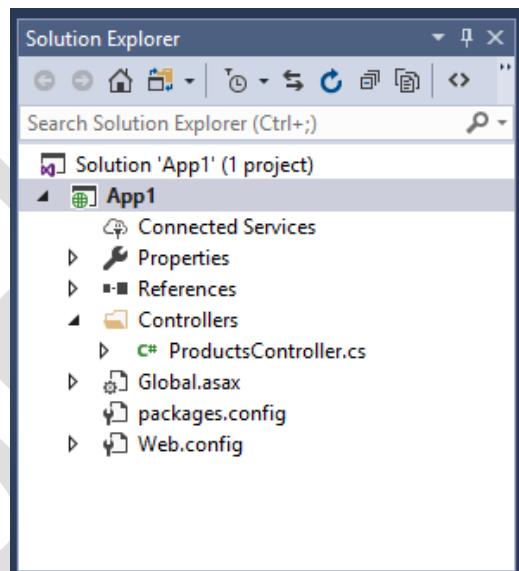
### Creating ProductsController.cs

- Right click on the project (App1) and click on "Add" - "New Folder". Type the folder name as "Controllers" and press Enter. Right click on the "Controllers" folder and click on "Add" - "New Item". Click on "Code" - "Class". Filename is "ProductsController.cs". Click on "Add". Replace the code for "ProductsController.cs" file as follows:

### Code for "ProductsController.cs"

```
using System;
using System.Web.Mvc;

namespace App1.Controllers
{
    public class ProductsController : Controller
    {
        public string Show(string Category)
        {
            return "Category: " + Category;
        }
    }
}
```



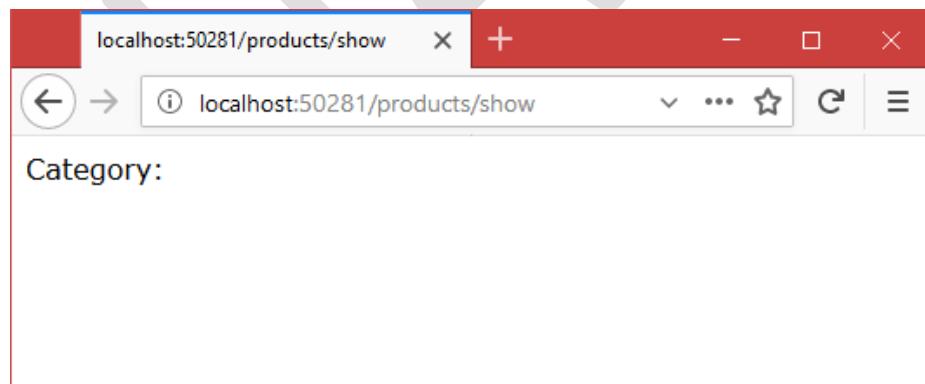
### Compile the MVC Project

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B". The project will be compiled.

### Run the MVC Project

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".
- Type "<http://localhost:50281/products/show>".
- Note:** The port number may vary.

### Output:



## Default Parameters

- This concept is used to set a default for the parameter. If the user has not passed the value, it takes the "default value"; if the user passes the value, it takes given value into the parameter.

### Syntax of Default Parameters

**Syntax:** System.Web.Routing.RouteTable.Routes.MapRoute( "route\_name", "{parameter1}/{parameter2}/...", new { parameter1 = defaultvalue, parameter2 = defaultvalue } );

**Example:** System.Web.Routing.RouteTable.Routes.MapRoute( "route1", "{controller}/{action}/{id}", new { id = 100 } );

## Default Parameters - Example

### Creating MVC Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "App1". Type the location as "C:\mvc". Type the solution name as "App1". Click on OK. Select "Empty". Uncheck all the checkboxes (in case of any selected). Click on OK.

### Installing Packages

- Open Solution Explorer. Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.  

```
install-package Microsoft.AspNet.Mvc
```
- The following code will be automatically generated for "packages.config".

### Code for "packages.config" (Automatically generated)

```
<?xml version="1.0" encoding="utf-8"?>
<packages>
  <package id="Microsoft.AspNet.Mvc" version="5.2.3" targetFramework="net47" />
  <package id="Microsoft.AspNet.Razor" version="3.2.3" targetFramework="net47" />
  <package id="Microsoft.AspNet.WebPages" version="3.2.3" targetFramework="net47" />
  <package id="Microsoft.CodeDom.Providers.DotNetCompilerPlatform" version="1.0.7" targetFramework="net47" />
  <package id="Microsoft.Net.Compilers" version="2.1.0" targetFramework="net47" developmentDependency="true" />
  <package id="Microsoft.Web.Infrastructure" version="1.0.0.0" targetFramework="net47" />
</packages>
```

### Creating Global.asax

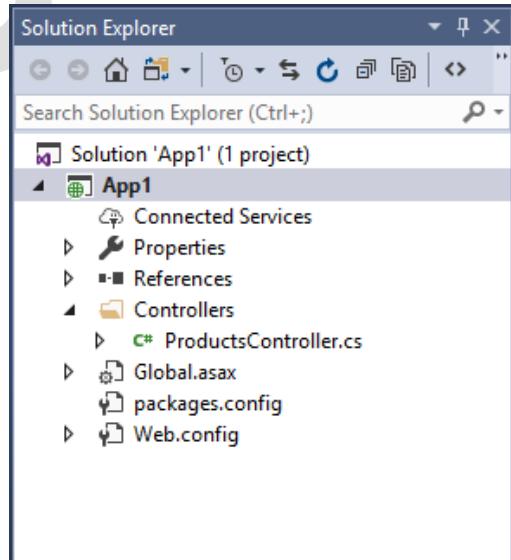
- Right click on the project (App1) and click on "Add" - "New Item". Click on "Web" - "Global Application Class". Filename is "Global.asax" (automatically displayed). Click on "Add". It generates several methods like "Application\_Start()", "Session\_Start()", "Application\_Error()" etc. Replace the code for "Global.asax" file as follows:

### Code for "Global.asax"

```
using System;
using System.Web.Mvc;

namespace App1
{
    public class Global : System.Web.HttpApplication
    {

        protected void Application_Start(object sender, EventArgs e)
        {
            System.Web.Routing.RouteTable.Routes.MapRoute("route1",
                "{controller}/{action}/{category}", new { category = "Electronics" });
        }
    }
}
```



### Creating ProductsController.cs

- Right click on the project (App1) and click on "Add" - "New Folder". Type the folder name as "Controllers" and press Enter. Right click on the "Controllers" folder and click on "Add" - "New Item". Click on "Code" - "Class". Filename is "ProductsController.cs". Click on "Add". Replace the code for "ProductsController.cs" file as follows:

### Code for "ProductsController.cs"

```
using System;
using System.Web.Mvc;
```

```
namespace App1.Controllers
{
    public class ProductsController : Controller
    {
        public string Show(string Category)
        {
            return "Category: " + Category;
        }
    }
}
```

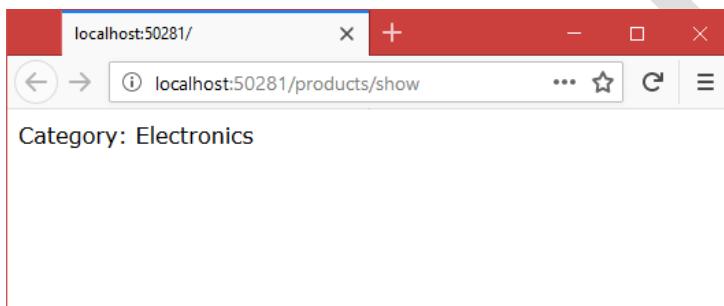
### Compile the MVC Project

- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B".
- The project will be compiled.

### Run the MVC Project

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5". Type "<http://localhost:50281/products/show>".
- **Note:** The port number may vary.

#### Output:



## Constraints

- This concept is used to set pattern of the parameter value. If the user passes a value that matches with the specified pattern, the route will be accepted and the value will be stored in the parameter; if the user passes a value that doesn't match with the specified pattern, the route will not be accepted.

### Syntax of Constants

**Syntax:** System.Web.Routing.RouteTable.Routes.MapRoute( "route\_name", "{parameter1}/{parameter2}...", new { parameter1 = defaultValue, parameter2 = defaultValue, ... }, new { parameter1 = @"constraint1", parameter2 = @"constraint2" } );

**Example:** System.Web.Routing.RouteTable.Routes.MapRoute( "route1", "[controller]/[action]/[id]", new { id = 100 }, new { id = @"^[[0-9]\*\$" } );

## Constraints - Example

### Creating MVC Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". "New Project" dialogbox appears. Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "App1". Type the location as "C:\mvcc". Type the solution name as "App1". Click on OK. Select "Empty". Uncheck all the checkboxes (in case of any selected). Click on OK.

### Installing Packages

- Open Solution Explorer. Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.  
`install-package Microsoft.AspNet.Mvc`
- The following code will be automatically generated for "packages.config".

### Code for "packages.config" (Automatically generated)

```
<?xml version="1.0" encoding="utf-8"?>
<packages>
```

```
<package id="Microsoft.AspNet.Mvc" version="5.2.3" targetFramework="net47" />
<package id="Microsoft.AspNet.Razor" version="3.2.3" targetFramework="net47" />
<package id="Microsoft.AspNet.WebPages" version="3.2.3" targetFramework="net47" />
<package id="Microsoft.CodeDom.Providers.DotNetCompilerPlatform" version="1.0.7" targetFramework="net47" />
<package id="Microsoft.Net.Compilers" version="2.1.0" targetFramework="net47" developmentDependency="true" />
<package id="Microsoft.Web.Infrastructure" version="1.0.0.0" targetFramework="net47" />
</packages>
```

### Creating Global.asax

- Right click on the project (App1) and click on "Add" - "New Item". Click on "Web" - "Global Application Class". Filename is "Global.asax" (automatically displayed). Click on "Add". It generates several methods like "Application\_Start()", "Session\_Start()", "Application\_Error()" etc. Replace the code for "Global.asax" file as follows:

#### Code for "Global.asax"

```
using System;
using System.Web.Mvc;

namespace App1
{
    public class Global : System.Web.HttpApplication
    {

        protected void Application_Start(object sender, EventArgs e)
        {
            System.Web.Routing.RouteTable.Routes.MapRoute("route1", "{controller}/{action}/{category}", new {}, new { category = @"\w+[^a-zA-Z]*$" });
            System.Web.Routing.RouteTable.Routes.MapRoute("route2", "{controller}/{action}/{empid}", new {}, new { empid = @"\w+[0-9]*$" });
        }
    }
}
```

### Creating ProductsController.cs

- Right click on the project (App1) and click on "Add" - "New Folder". Type the folder name as "Controllers" and press Enter. Right click on the "Controllers" folder and click on "Add" - "New Item". Click on "Code" - "Class". Filename is "ProductsController.cs". Click on "Add". Replace the code for "ProductsController.cs" file as follows:

#### Code for "ProductsController.cs"

```
using System;
using System.Web.Mvc;

namespace App1.Controllers
{
    public class ProductsController : Controller
    {
        public string Show(string Category)
        {
            return "Category: " + Category;
        }
    }
}
```

### Creating EmployeesController.cs

- Right click on the project (App1) and click on "Add" - "New Folder". Type the folder name as "Controllers" and press Enter. Right click on the "Controllers" folder and click on "Add" - "New Item". Click on "Code" - "Class". Filename is "EmployeesController.cs". Click on "Add". Replace the code for "EmployeesController.cs" file as follows:

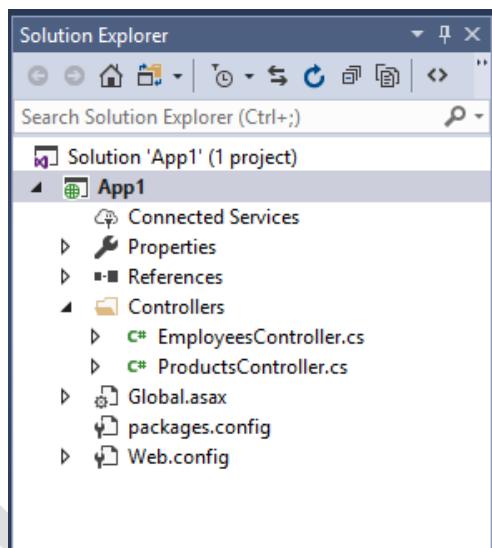
**Code for "EmployeesController.cs"**

```
using System;
using System.Web.Mvc;

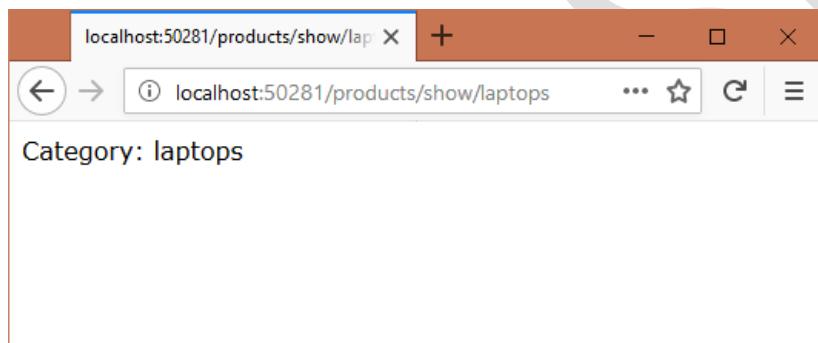
namespace App1.Controllers
{
    public class EmployeesController : Controller
    {
        public string Display(int EmpID)
        {
            return "Emp ID: " + EmpID;
        }
    }
}
```

**Compile the MVC Project**

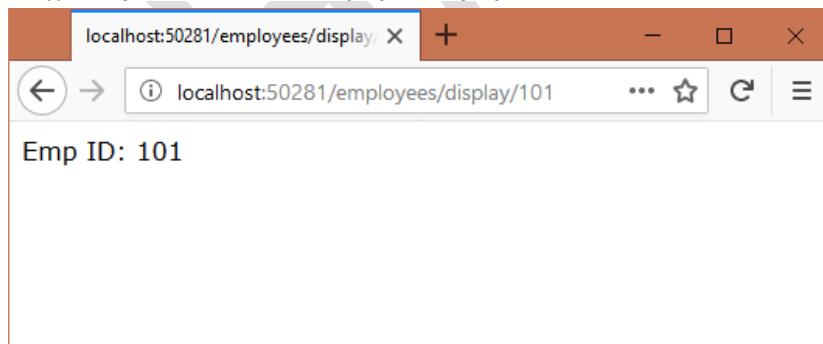
- Go to "Build" menu - click on "Build Solution" (or) Press "Ctrl+Shift+B".
- The project will be compiled.

**Run the MVC Project**

- Go to "Debug" menu - click on "Start Debugging" (or) Press "F5".
- Type "<http://localhost:50281/products/show/laptops>".
- **Note:** The port number may vary.

**Output:**

- Type "<http://localhost:50281/employees/display/101>".



### HTTP

#### What is HTTP

- HTTP stands for "Hypertext Transfer Protocol". HTTP provides a set of rules to transfer request from browser to server and response from server to browser. ASP.NET MVC supports HTTP.

#### MIME Types

- Represents type of data (content).
  - text/plain
  - text/html
  - text/css
  - text/javascript
  - application/json
  - text/xml
  - image/png
  - image/jpeg
  - audio/mp3
  - video/mp4

#### Character Encoding Formats

- The characters (alphabets, digits or symbols) are converted into equivalent numbers based on "character encoding formats".
  - ASCII (American Standard Code for Information Interchange): It contains standard characters only.
    - A-Z: 65 to 90
    - a-z: 97 to 122
    - 0-9: 48 to 57
    - Space: 32
    - Backspace: 8
  - Unicode: It is the extension of ASCII. Unicode includes all language characters worldwide.

#### HTTP Status Codes

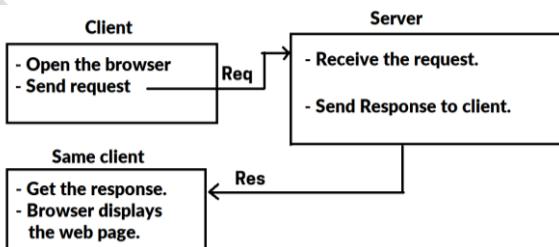
- Represents type of data (content).
  - 100: Continue
  - 200: OK
  - 302: Redirection
  - 304: Not Modified
  - 400: Bad Request
  - 401: Unauthorized
  - 404: Page not found
  - 500: Internal Server Error

#### Http Request

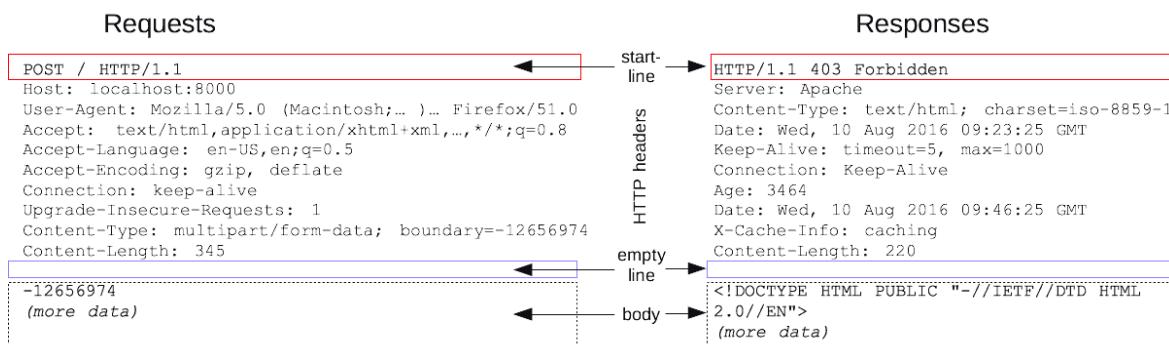
- It is a HTTP message sent from client to server.

#### Http Response

- It is a HTTP message sent from server to client.



## HTTP Request Format and Http Response Format



### Request Start Line

- It contains "HTTP Method", "URL" and "HTTP Version". HTTP Method specifies how you want to submit data to server. URL specifies to which address the data has to be submitted. HTTP version is fixed "HTTP/1.1".

### Request Headers

- It contains additional "key-value pairs" that are need to be sent to server, which tells client details to server and how the data has to be processed.

### Request Body

- It is the actual data to be sent to server. Ex: In the login form, the username and password will be submitted as request body.

### Response Start Line

- It contains "HTTP version", "HTTP Status Code" and "HTTP Status Name".

### Response Headers

- It contains additional "key-value pairs" that are need to be sent to client, which tells server details and how the data has to be processed.

### Response Body

- It is the actual data to be sent to server. Ex: In the login form, the message success / failure will be sent to the client.

## Syntax of URL Routing

**Syntax:** System.Web.Routing.RouteTable.Routes.MapRoute("route name", "url", new { controller = "controller name", action = "action name" });

**Example:** System.Web.Routing.RouteTable.Routes.MapRoute("route1", "home", new { controller = "controller1", action = "action1" });

## The "Request" object

### Request Object

- Browser sends a "HTTP Request" to server and gets "HTTP Response" from server. "Request" is a pre-defined object in "ASP.NET MVC", which represents the data that is sent from "browser to server". "Request" is an object of "System.Web.HttpRequestBase" class.

### Members of "Request" object

#### 1. Request.Url

- Represents the current url (address).
- Syntax:** Request.Url

#### 2. Request.PhysicalApplicationPath

- Represents current folder path.
- Syntax: Request.PhysicalApplicationPath

### **3. Request.Path**

- Represents current virtual path.
- Syntax: Request.Path

### **4. Request.Browser.Type**

- Represents the current browser name.
- Syntax: Request.Browser.Type

### **5. Request.QueryString**

- Represents the current query string.
- Query string is a set of parameters that the browser passes to the server.
- Syntax: Request.QueryString

### **6. Request.Headers**

- Represents the request headers (the headers, that are submitted from the browser to server).
- Syntax: Request.Headers["key"]

### **7. Request.HttpMethod**

- Represents the http method (GET | POST) of the current request.
- Syntax: Request.HttpMethod

## Request - Example

### **Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "RequestExample". Type the location as "C:\Mvc". Type the solution name as "RequestExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### **Creating HomeController.cs**

- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

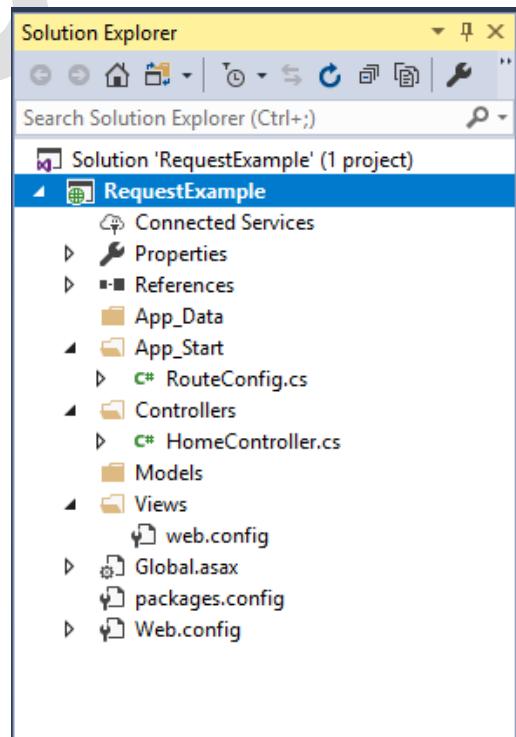
### **Code for "HomeController.cs"**

```

using System;
using System.Web.Mvc;

namespace RequestExample.Controllers
{
    public class HomeController : Controller
    {
        public void Index()
        {
            Response.Write("<p>URL: " + Request.Url.ToString() + "</p>");
            Response.Write("<p>Path: " + Request.Path + "</p>");
            Response.Write("<p>Query String: " + Request.QueryString + "</p>");
            Response.Write("<p>Method: " + Request.HttpMethod + "</p>");
            Response.Write("<p>Request headers: " + Request.Headers["User-Agent"] + "</p>");
            Response.Write("<p>Browser: " + Request.Browser.Type + "</p>");
            Response.Write("<p>Physical App Path: " + Request.PhysicalApplicationPath + "</p>");
        }
    }
}

```

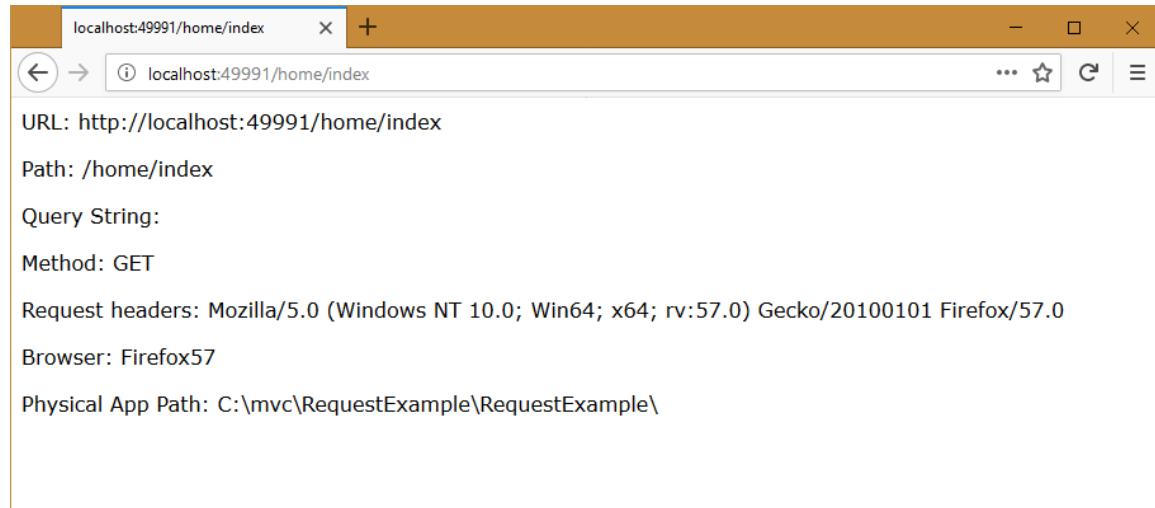


```
}
```

### Running the application

- Press "F5".
- Type "http://localhost:portnumber/Home/Index".

### Output



## The "Response" object

### Response Object

- Browser sends a "HTTP Request" to server and gets "HTTP Response" from server. "Response" is a pre-defined object in "ASP.NET MVC", which represents the data that is to be sent from "server to browser". "Response" is an object of "System.Web.HttpResponseBase" class.

### Members of "Response" object

#### 1. Response.Write()

- This method sends given data to the browser.
- Syntax: Response.Write(any data)

#### 2. Response.ContentType

- This property represents type of the response content.
- Syntax: Response.ContentType
- Examples:

text/plain  
text/html  
text/css  
text/javascript  
text/xml  
application/json  
etc.

#### 3. Response.Headers

- Represents the response headers that are to be sent from the browser to server.
- Syntax: Response.Headers.Add("key", "value");

#### 4. Response.StatusCode

- Represents the status of the response.
- Syntax: Response.StatusCode = n;
- Example:

- 200 : OK
- 302 : Redirection
- 304 : Not Modified (browser cached)
- 400 : Bad Request
- 401 : Unauthorized
- 404 : Not Found
- 500 : Internal Server Error

### Response - Example

#### **Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "ResponseExample". Type the location as "C:\Mvc". Type the solution name as "ResponseExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

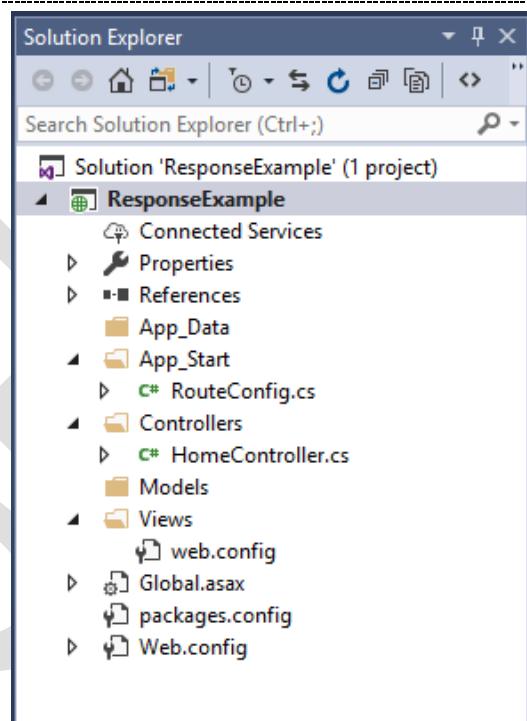
#### **Creating HomeController.cs**

- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### **Code for "HomeController.cs"**

```
using System;
using System.Web.Mvc;

namespace ResponseExample.Controllers
{
    public class HomeController : Controller
    {
        public void Index()
        {
            Response.Write("Hello");
            Response.Write("World");
            Response.Headers.Add("x", "100");
            Response.ContentType = "text/plain";
            Response.StatusCode = 500;
        }
    }
}
```



#### **Running the application**

- Press "F5". Type "http://localhost:portnumber/Home/Index".

Output:

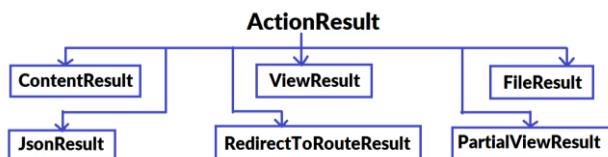
The screenshot shows a browser window with the URL `localhost:50471/`. The page content says "Hello World". Below the browser is the Microsoft F12 developer tools Network tab. It shows a single request for the root URL (`/`) with a status of 500 Internal Server Error. The Headers section shows the following details:

- Request URL: `http://localhost:50471/`
- Request method: GET
- Remote address: `[::1]:50471`
- Status code: 500 Internal Server Error
- Version: HTTP/1.1
- Response headers (407 B):
  - `Cache-Control: private`
  - `Content-Length: 11`
  - `Content-Type: text/plain; charset=utf-8`
  - `Date: Sat, 27 Jan 2018 21:23:58 GMT`
  - `Server: Microsoft-IIS/10.0`
  - `x: 100`
  - `X-AspNet-Version: 4.0.30319`
  - `X-AspNetMvc-Version: 5.2`
  - `X-Powered-By: ASP.NET`
  - `X-SourceFiles: =?UTF-8?B?RDncT25IRHIndmVrMDFl...nNIXFlrc3BvhNIRXhhhXRcZQ==?=`

Right click on the web page and click on "Inspect Element". Press "F5" to refresh the page. Click on the request to see details such as status and headers.

### ACTIONRESULT

- "ASP.NET MVC" framework recommends to specify the return type of all action methods as "ActionResult". If you specify the return type of action method as "ActionResult", you can return any one of its child classes, depending on the requirement. "ActionResult" is an abstract class, based on which, many child classes are available.



#### 1. ContentResult

- Represents any content with specified "content-type".

#### 2. ViewResult

- Represents the "result" of a view.

#### 3. FileResult

- Represents the content of a file.

#### 4. JsonResult

- Represents a "json object" / "json array".

#### 5. RedirectToRouteResult

- Represents redirection response.

#### 6. PartialViewResult

- Represents the result of partial view.

## ContentResult

### ContentResult

- This class's object represents some content, with the specified content type. The default content type is "text/html". The "Content" method creates and returns an object of "ContentResult" class.

### Syntax of Action method with ContentResult

```
public ContentResult Actionname()
{
    return Content("your text here", "text/plain");
}
```

## ContentResult - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "ContentResultExample". Type the location as "C:\Mvc". Type the solution name as "ContentResultExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

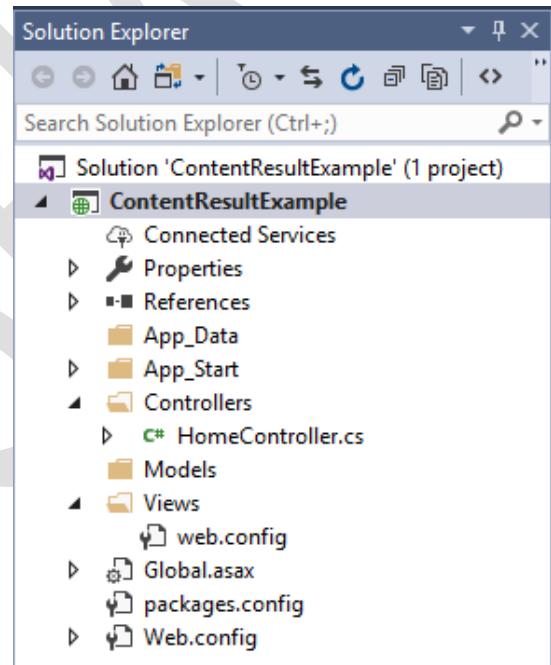
### Creating HomeController.cs

- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;

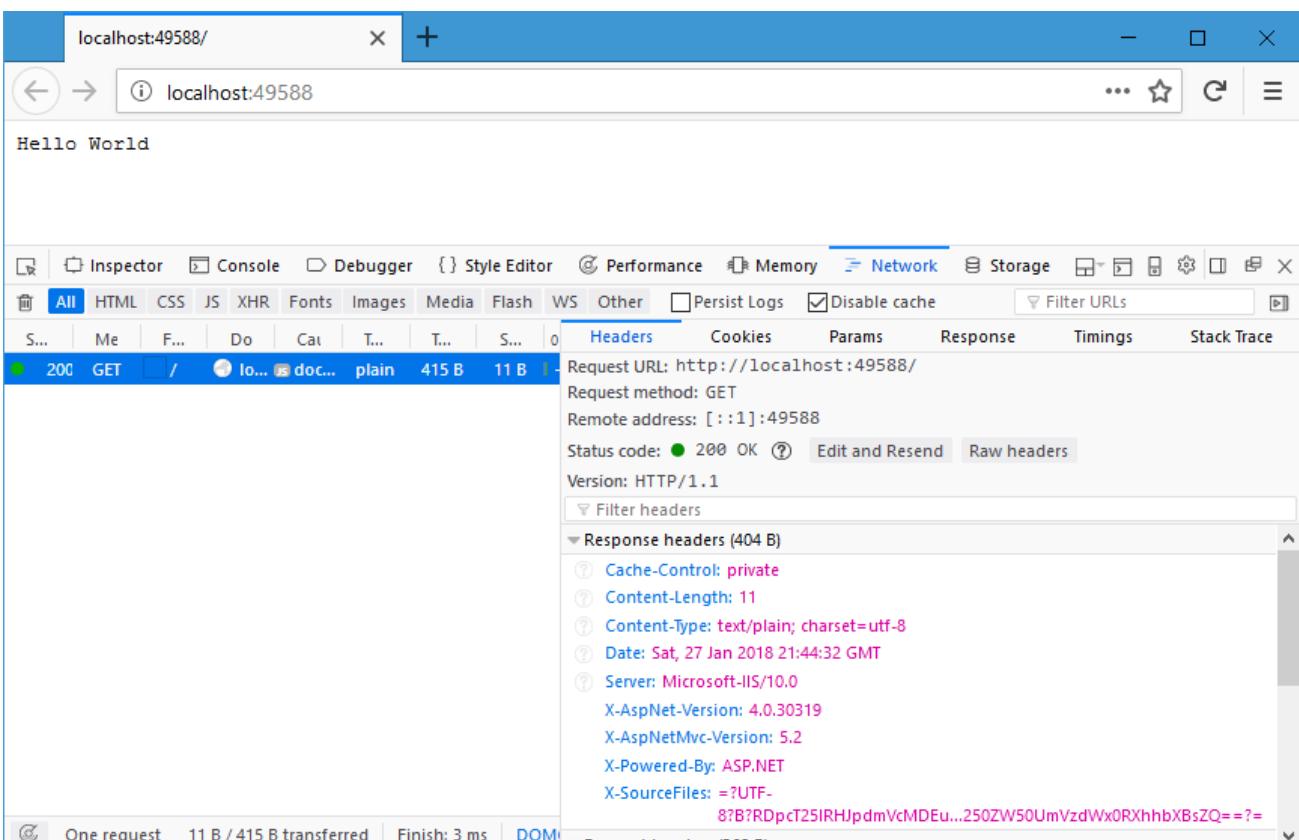
namespace ContentResultExample.Controllers
{
    public class HomeController : Controller
    {
        public ContentResult Index()
        {
            ContentResult c = new ContentResult();
            c.Content = "Hello World";
            c.ContentType = "text/plain";
            return c;
            //return Content("Hello World", "text/plain");
        }
    }
}
```



### Running the application

- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Index".

Output:



## FileResult

- This class's object represents the content of a file. The "File" method creates and returns an object of "FileResult" class.

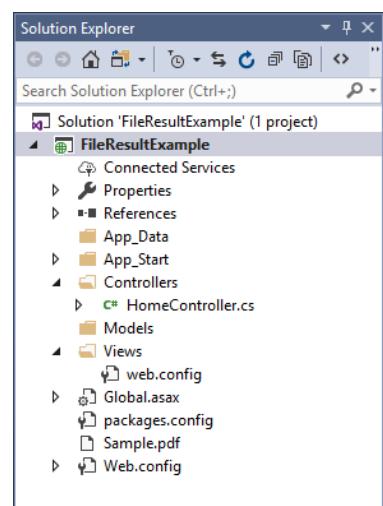
### Syntax of Action method with FileResult

```
public FileResult Actionname()
{
    return File("~/file path here", "content type here");
}
```

## FileResult - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "FileResultExample". Type the location as "C:\Mvc". Type the solution name as "FileResultExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK. Create a PDF file called "Sample.pdf" and copy-paste the file into the solution explorer.



### Creating HomeController.cs

- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;

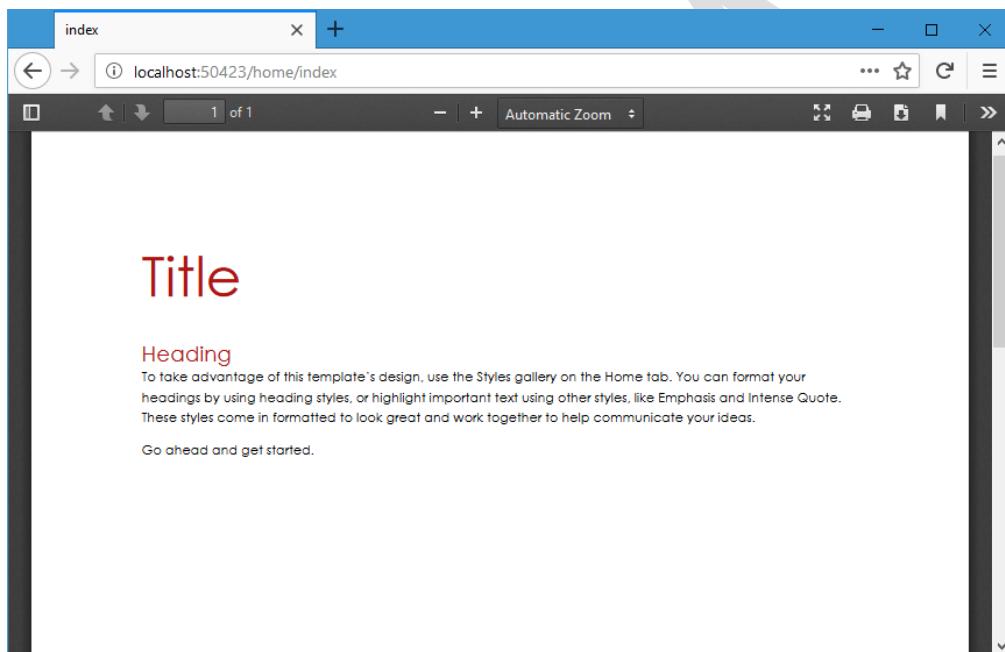
namespace FileResultExample.Controllers
```

```
{
    public class HomeController : Controller
    {
        public FileResult Index()
        {
            return File("~/Sample.pdf", "application/pdf");
        }
    }
}
```

#### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



#### RedirectToRouteResult

- This class's object represents a redirection to specified action method at specified controller. It sends HTTP 302 response to the browser. The "RedirectToAction" method creates and returns an object of "RedirectToRouteResult" class.

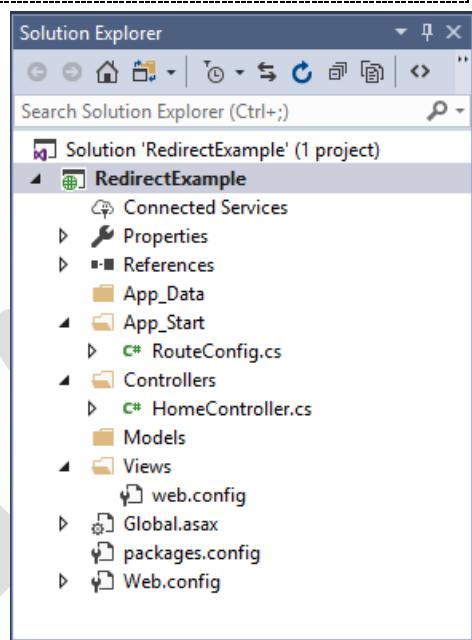
#### Syntax of Action method with RedirectToRouteResult

```
public RedirectToRouteResult Actionname()
{
    return RedirectToAction("action name", "controller name");
}
```

## RedirectToRouteResult - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "RedirectExample". Type the location as "C:\Mvc". Type the solution name as "RedirectExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.



### Creating HomeController.cs

- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;

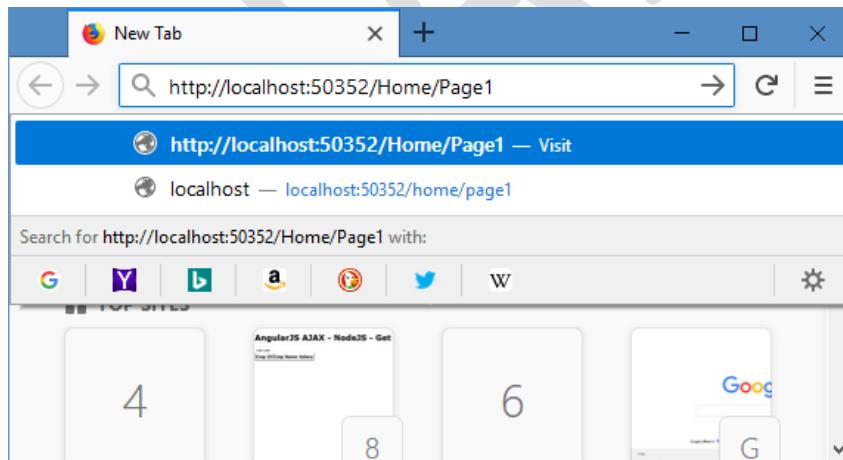
namespace RedirectExample.Controllers
{
    public class HomeController : Controller
    {
        public RedirectToRouteResult Page1()
        {
            return RedirectToAction("Page2", "Home");
        }

        public ContentResult Page2()
        {
            return Content("Page 2");
        }
    }
}
```

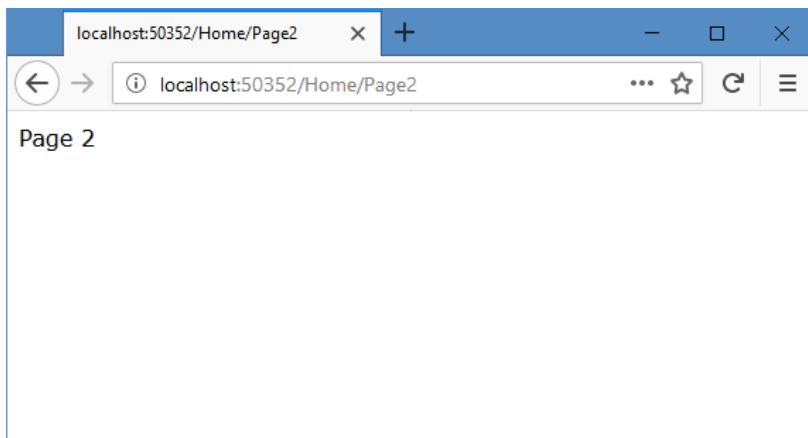
### Running the application

- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Page1".

Output:



After pressing Enter.



## Server.Transfer

- This class's object represents a request transfer to specified action method at specified controller. It transfers the current request to the specified action method at specified controller.

### Syntax of Server.TransferRequest

```
public void Actionname()
{
    return Server.TransferRequest("url");
}
```

## Server.Transfer - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "ServerTransferExample". Type the location as "C:\Mvc". Type the solution name as "ServerTransferExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating HomeController.cs

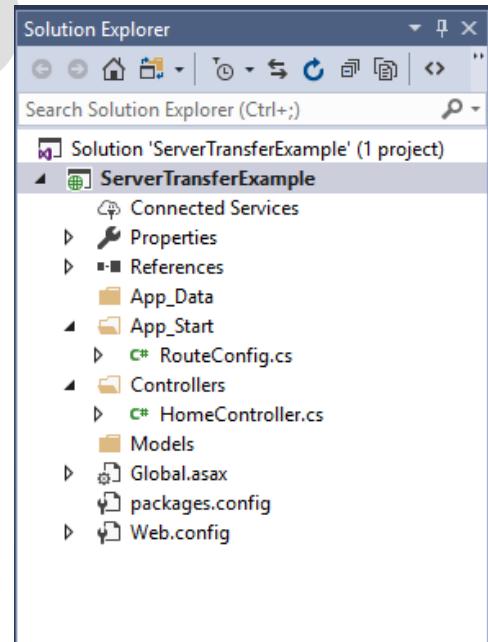
- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;

namespace ServerTransferExample.Controllers
{
    public class HomeController : Controller
    {
        public void Page1()
        {
            Response.Write("<h1>Page 1</h1>");
            Server.TransferRequest("/Home/Page2");
        }

        public ContentResult Page2()
        {
            return Content("<h1>Page 2</h1>");
        }
    }
}
```

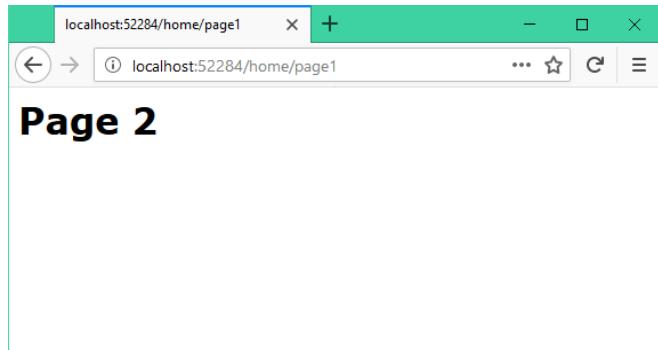


}

### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Page1".

Output:



## Views

- View is an asp.net web page that contains presentation logic. Views are present in "Views\controllername" folder. The view's file extension should be ".cshtml". View can contains two types of presentation logic:

### 1. Server side presentation logic:

- This executes on server. This executes first. This requires to use a "view engine".

### 2. Client side presentation logic:

- This executes on client (browser). This executes after server side presentation logic. For every controller, there should be a folder in the "Views" folder, where the controller name and folder name should be SAME. All the views that belongs to the controller, should be present within the corresponding folder only. It is recommended to maintain the same name for both view and action method. When you return a view, ASP.NET MVC checks for the view within the corresponding folder only (not in other controller's folder).

### Syntax of a view

```
<html>
  <head>
    <title>Title here</title>
  </head>
  <body>
    Content here
  </body>
</html>
```

### Creating a View

- Create a sub folder in "Views" folder, with same name as the controller name. Ex: If controller name is "HomeController", the folder should be "Views\Sample". Right click on "Views\controllername" folder and click on "Add" - "View". Select the template as "Empty (without model)". Type the view name. Uncheck all the checkboxes. Click on "Add".

### Syntax of Action method with ViewResult

```
public ViewResult Actionname()
{
  return View();
}
```

## Views - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "ViewExample". Type the location as "C:\Mvc". Type the solution name as "ViewExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

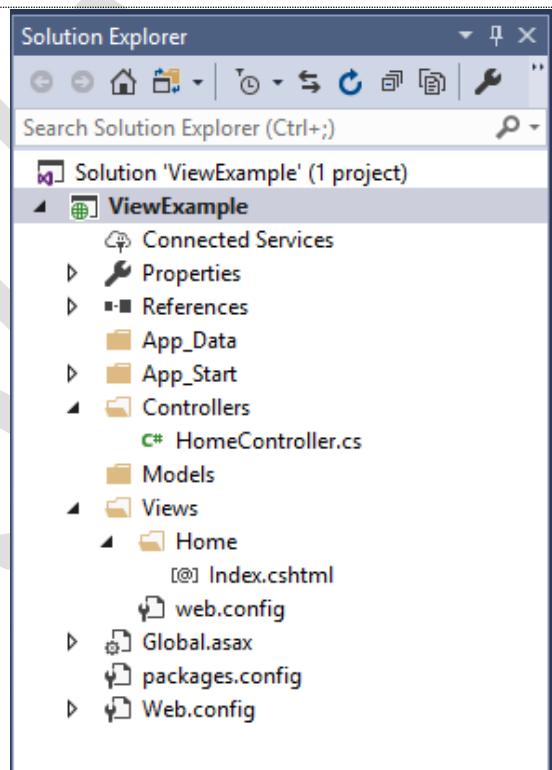
### Creating HomeController.cs

- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;

namespace ViewExample.Controllers
{
    public class HomeController : Controller
    {
        public ViewResult Index()
        {
            return View();
        }
    }
}
```



### Creating Index.cshtml

- Create "Home" folder in "Views" folder, if not exists. Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

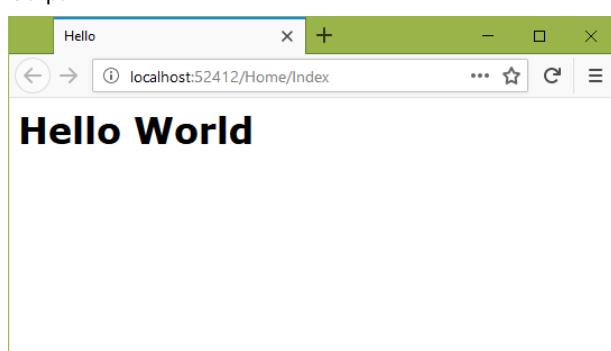
### Code for "Views\Home\Index.cshtml"

```
<html>
<head>
    <title>Hello</title>
</head>
<body>
    <h1>Hello World</h1>
</body>
</html>
```

### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



### View Name

- It is used to specify the view name while calling a view from controller. By default, it is recommended to maintain the action method name and view name as same. In case of you have a different name for the view, then you need to specify the view name in the controller, to call it.

#### Syntax of Action method with ViewResult with Name

```
public ViewResult Actionname()  
{  
    return View("view name");  
}
```

### View Name - Example

#### Creating Project

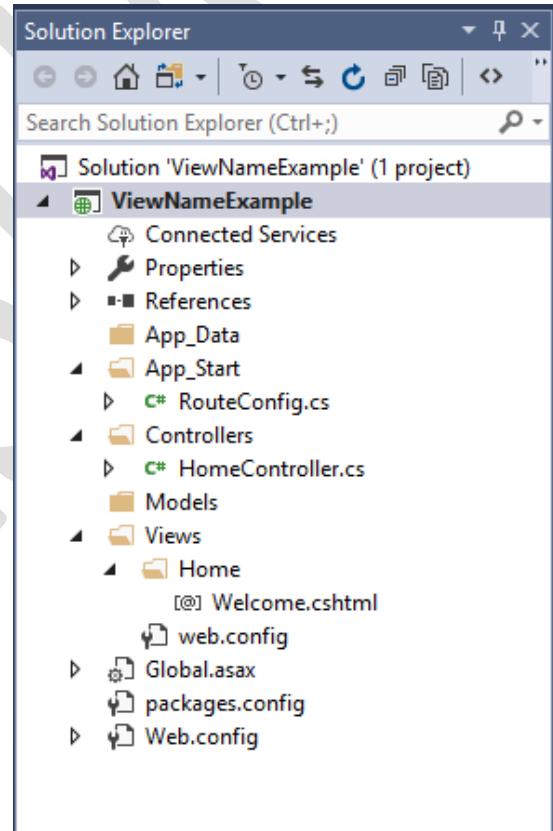
- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "ViewNameExample". Type the location as "C:\Mvc". Type the solution name as "ViewNameExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

#### Creating HomeController.cs

- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### Code for "HomeController.cs"

```
using System;  
using System.Web.Mvc;  
  
namespace ViewNameExample.Controllers  
{  
    public class HomeController : Controller  
    {  
        public ViewResult Index()  
        {  
            return View("Welcome");  
        }  
    }  
}
```



#### Creating Welcome.cshtml

- Create "Home" folder in "Views" folder, if not exists. Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Welcome". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

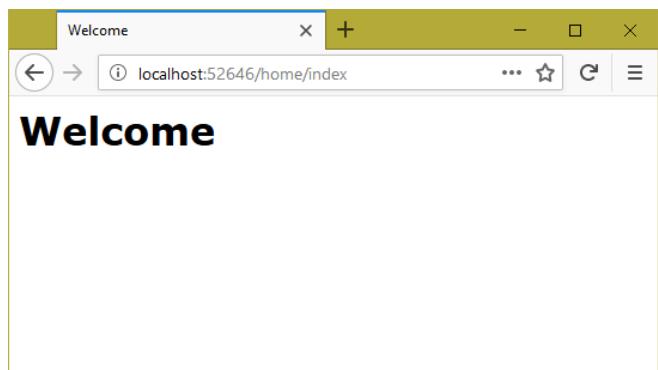
#### Code for "Views\Home\Welcome.cshtml"

```
<html>  
  <head>  
    <title>Welcome</title>  
  </head>  
  <body>  
    <h1>Welcome</h1>  
  </body>  
</html>
```

### **Running the application**

- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Index".

Output:



## RAZOR

### What is View Engine

- A view engine provides a set of rules to write c# code (server side code) in the view. ASP.NET MVC supports two types of view engines:
  1. ASPX View Engine
  2. Razor View Engine

### **ASPX and Razor - MVC Versions**

1. ASP.NET MVC 1	:	ASPX
2. ASP.NET MVC 2	:	ASPX
3. ASP.NET MVC 3	:	ASPX and Razor
4. ASP.NET MVC 4	:	ASPX and Razor
5. ASP.NET MVC 5	:	Razor
6. ASP.NET Core MVC 1.0	:	Razor
7. ASP.NET Core MVC 2.0	:	Razor

### **ASPX and Razor - Syntax Comparison**

ASPX View Engine	Razor View Engine
html code <% c#.net code %> html code	html code @{           c#.net code           }           html code

### **Syntax of Razor View Engine**

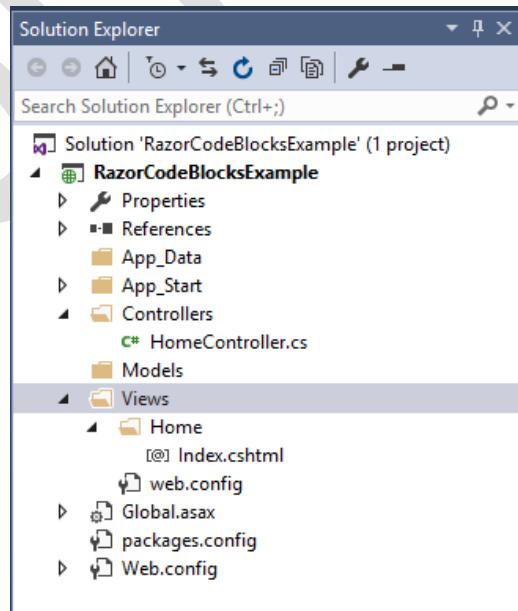
Sl. No	Concept	Syntax	Example	Output
1	Code Block	@{ c# code here }	@{ int a = 10, b = 20; int c = a + b; }	The c# code will be executed at server side.
2	Expressions	@Variablename	@c	30
3	Razor if	@if (condition) { C# code here }	@if (c == 30) { @c }	30

4	Razor for loop	<pre>@for (initialization; condition; incrementation) {     C# code here }</pre>	<pre>@for (i=1; i&lt;=10; i++) {     @i &lt;br&gt; }</pre>	1 2 3 4 5 6 7 8 9 10
5	Razor foreach loop	<pre>@foreach (var item in collection) {     C# code here }</pre>	<pre>@{     List&lt;string&gt; MyFriends = new     List&lt;string&gt; { "scott", "john", "allen" }; }  foreach (var item in MyFriends) {     @item &lt;br&gt; }</pre>	Scott John Allen
6	Razor comments	<pre>@* comment here *@</pre>	<pre>@* hai *@</pre>	[Comments will not be displayed in the output]
7	Razor Literals	<pre>@: static text here</pre>	<pre>@:hai</pre>	hai

### Razor - Code Blocks - Example

#### Creating Project

- Open Visual Studio 2017. Go to “File” - “New” - “Project”. Select “.NET Framework 4.7”. Select “Visual C#”. Select “ASP.NET Web Application (.NET Framework)”. Type the project name “RazorCodeBlocksExample”. Type the location as “C:\Mvc”. Type the solution name as “RazorCodeBlocksExample”. Click on OK. Select “Empty”. Check the checkbox “MVC”. Uncheck the checkbox “Enable Docker support”. Uncheck the checkbox “Add unit tests”. Click on OK.



#### Creating HomeController.cs

- Open Solution Explorer. Right click on “Controllers” folder and click on “Add” - “Controller”. Select “MVC 5 Controller - Empty”. Click on “Add”. Type the controller name as “HomeController”. Click on “Add”.

#### Code for “HomeController.cs”

```
using System;  
using System.Web.Mvc;  
  
namespace RazorCodeBlocksExample.Controllers  
{  
    public class HomeController : Controller  
    {  
        public ActionResult Index()  
        {  
            return View();  
        }  
    }  
}
```

#### Creating Index.cshtml

- Right click on “Views\Home” folder and click on “Add” - “View”. Type the view name as “Index”. Select the template “Empty (without model)”. Uncheck all the checkboxes. Click on “Add”.

#### Code for “Views\Home\Index.cshtml”

```
<html>  
  <head>  
    <title>Razor Code Blocks</title>  
  </head>
```

```

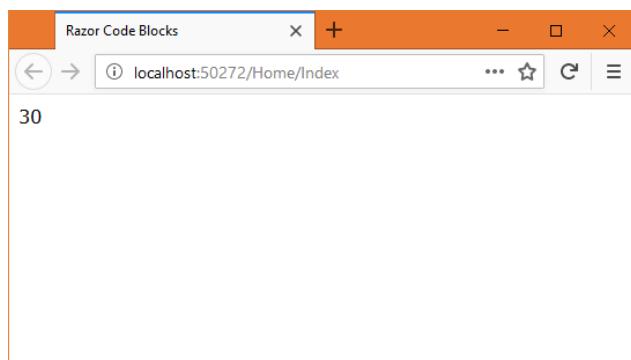
<body>
    @{
        int a, b, c;
        a = 10;
        b = 20;
        c = a + b;
        Response.Write(c);
    }
</body>
</html>

```

#### Running the application

- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Index".

#### Output



## Razor - Expressions - Example

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "ExpressionsExample". Type the location as "C:\Mvc". Type the solution name as "ExpressionsExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

#### Creating HomeController.cs

- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

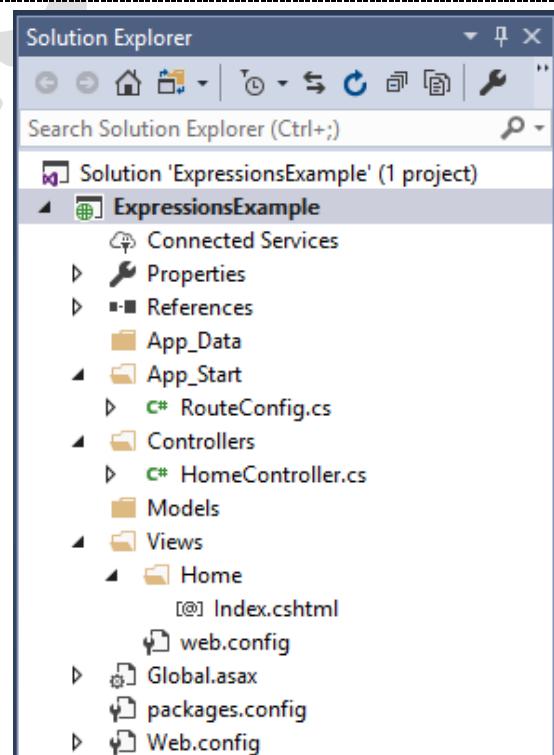
#### Code for "HomeController.cs"

```

using System;
using System.Web.Mvc;

namespace ExpressionsExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}

```



#### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Index.cshtml"**

```

<html>
<head>
<title>Expressions</title>
</head>
<body>
@{
    string message = "Hello";
    string message2 = "Welcome to ASP.NET";
}

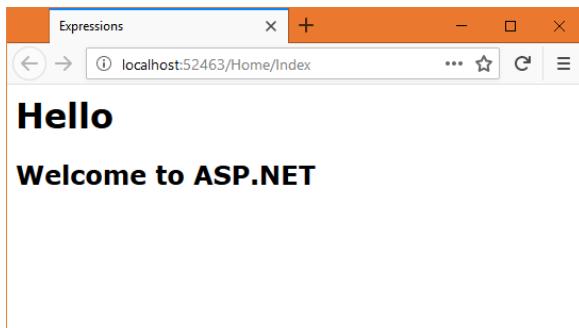
<h1>@message</h1>
<h2>@message2</h2>
</body>
</html>

```

**Running the application**

- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Index".

Output:

**Razor - If - Example****Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "RazorIfExample". Type the location as "C:\Mvc". Type the solution name as "RazorIfExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK. Open Solution Explorer.

**Creating HomeController.cs**

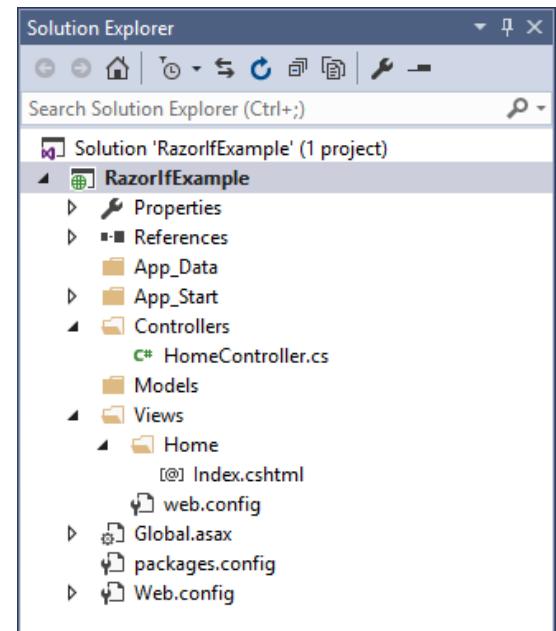
- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

**Code for "HomeController.cs"**

```

using System;
using System.Web.Mvc;
namespace RazorIfExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}

```



### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### Code for "Views\Home\Index.cshtml"

```
<html>
  <head>
    <title>Razor If</title>
  </head>
  <body>
    <h1>Razor - If</h1>
    @{
      int n=100;
    }

    @if(n >= 0)
    {
      <h2>@n is positive number</h2>
    }
    else
    {
      <h2>@n is negative number</h2>
    }
  </body>
</html>
```

### Running the application

- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Index".

Output:



## Razor - For - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "RazorForExample". Type the location as "C:\Mvc". Type the solution name as "RazorForExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating HomeController.cs

- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;

namespace RazorForExample.Controllers
{
```

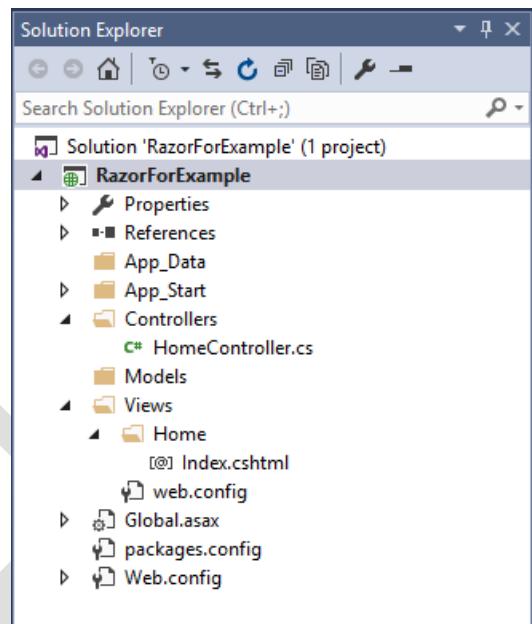
```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View();
    }
}
```

### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### Code for "Views\Home\Index.cshtml"

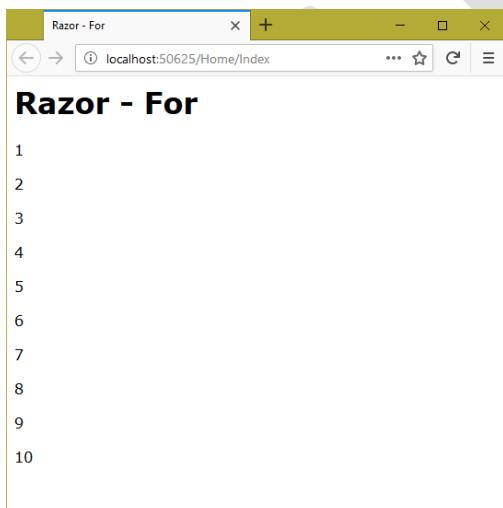
```
<html>
<head>
    <title>Razor - For</title>
</head>
<body>
    <h1>Razor - For</h1>
    @for (int i=1; i<=10; i++)
    {
        <p>@i</p>
    }
</body>
</html>
```



### Running the application

- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Index".

#### Output:



## Razor - Foreach - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "RazorForeachExample". Type the location as "C:\Mvc". Type the solution name as "RazorForeachExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating HomeController.cs

- Open Solution Explorer.

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

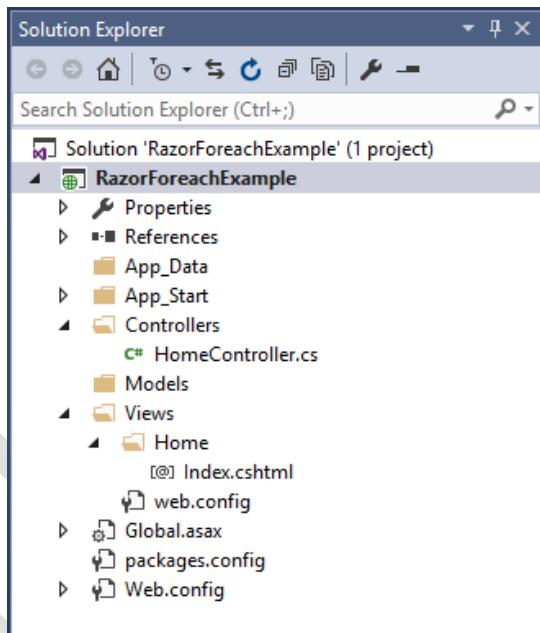
### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;

namespace RazorForEachExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".



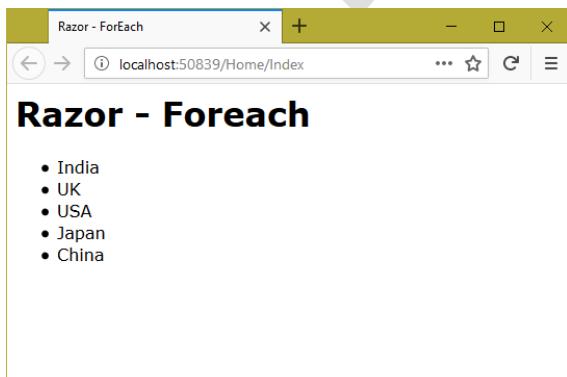
### Code for "Views\Home\Index.cshtml"

```
<html>
<head>
<title>Razor - ForEach</title>
</head>
<body>
<h1>Razor - Foreach</h1>
@{
    List<string> countries = new List<string>() { "India", "UK", "USA", "Japan", "China" };
}
<ul>
@foreach (string country in countries)
{
    <li>@country</li>
}
</ul>
</body>
</html>
```

### Running the application

- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Index".

Output:



## Razor – Literal - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "RazorLiteralExample". Type the location as "C:\Mvc". Type the solution name as "RazorLiteralExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

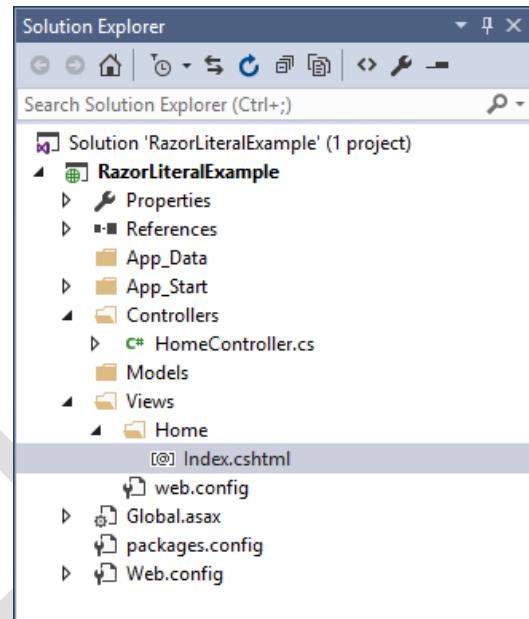
### Creating HomeController.cs

- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;

namespace RazorLiteralExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
```



### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

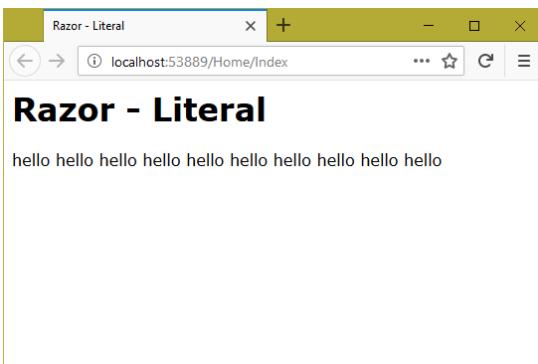
### Code for "Views\Home\Index.cshtml"

```
<html>
  <head>
    <title>Razor - Literal</title>
  </head>
  <body>
    <h1>Razor - Literal</h1>
    @for (int i=1; i <= 10; i++)
    {
      @:hello
    }
  </body>
</html>
```

### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



## Razor – Comments - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "RazorCommentsExample". Type the location as "C:\Mvc". Type the solution name as "RazorCommentsExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating HomeController.cs

- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;

namespace RazorCommentsExampe.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

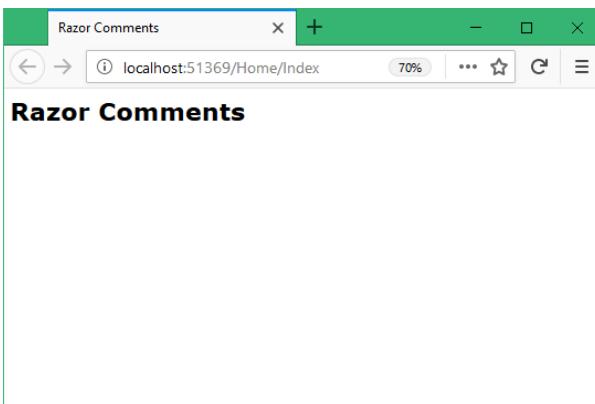
### Code for "Views\Home\Index.cshtml"

```
<html>
<head>
    <title>Razor Comments</title>
</head>
<body>
    <h1>Razor Comments</h1>
    @*comment here *@
</body>
</html>
```

### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



## STATE MANAGEMENT

### What is State Management

- State Management is a concept of sharing data from controller to view (or) controller to controller.
- There are 8 ways to pass data from controller to the view:
  1. ViewData
  2. ViewBag
  3. TempData
  4. Session
  5. Application
  6. Cache
  7. Cookies

### ViewData

- ViewData is used to transfer (pass) the data from "controller" to the "view". Controller sets data into "ViewData" and "View" gets data from "ViewData". ViewData's lifetime is "per request". When a browser sends a request to asp.net mvc web application, a new ViewData will be created. Once the response is delivered to the browser, the ViewData will be deleted automatically. If the browser sends another request, same process will be repeated; old ViewData can't be get back. ViewData internally uses Dictionary. ViewData supports to send any simple values, objects and collections also.
- **Syntax:** ViewData["key"]
- When you try to retrieve the value from ViewData, it returns "null" if the given key is not found. You have to convert the the value, if you are assigning ViewData's value into a variable.

### ViewData - Example

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "ViewDataExample". Type the location as "C:\Mvc". Type the solution name as "ViewDataExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

#### Creating HomeController.cs

- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;

namespace ViewDataExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
```

```

        ViewData["message"] = "Hello";
        return View();
    }
}

```

### **Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### **Code for "Views\Home\Index.cshtml"**

```

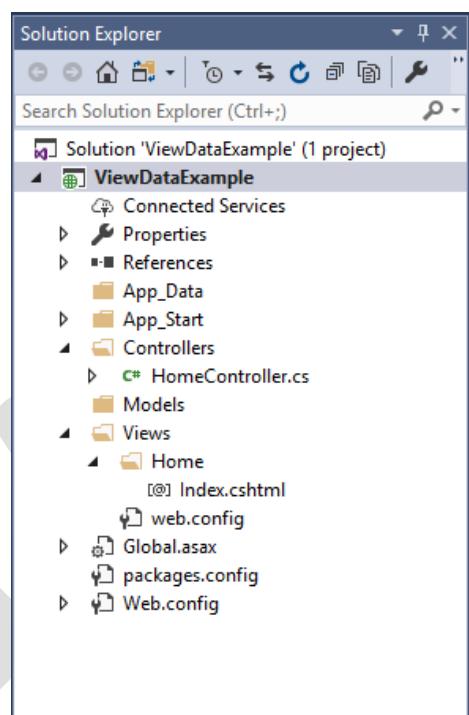
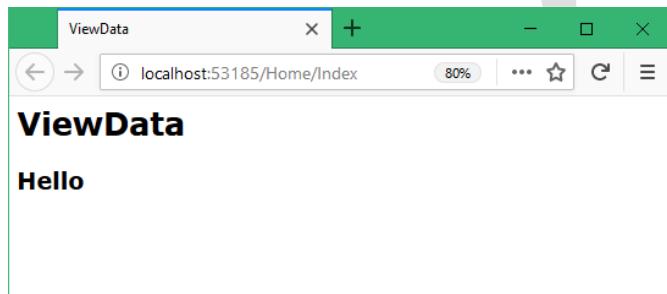
<html>
  <head>
    <title>ViewData</title>
  </head>
  <body>
    <h1>ViewData</h1>
    <h2>@ViewData["message"]</h2>
  </body>
</html>

```

### **Running the application**

- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Index".

Output:



## Models

- Model is a class that contains structure of the data to be displayed in the view. Model class contains properties to define fields. For example, "Employee" model class contains properties like "EmpID", "EmpName" and "Salary". Controller creates an object for the model class & passes it to the view. Then the view can access data from model object. Models are present in "Models" folder. Model classes are present in "Projectname.Models" namespace.

### **Syntax of Model**

```

using System;
namespace Projectname.Models
{
    public class Modelclassname
    {
        public Datatype Property1 { get; set; }
        public Datatype Property2 { get; set; }
        ...
    }
}

```

## ViewData with Models - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "ViewDataWithModelsExample". Type the location as "C:\Mvc". Type the solution name as "ViewDataWithModelsExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

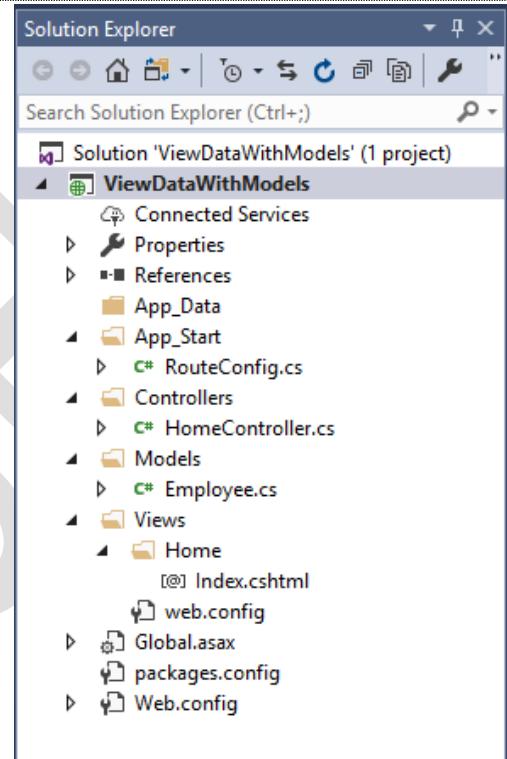
### Creating Employee.cs

- Open Solution Explorer. Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

### Code for "Employee.cs"

```
using System;

namespace ViewDataWithModels.Models
{
    public class Employee
    {
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public double Salary { get; set; }
    }
}
```



### Creating HomeController.cs

- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;
using ViewDataWithModels.Models;

namespace ViewDataWithModels.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            Employee emp = new Employee() { EmpID = 101, EmpName = "Scott", Salary = 4000 };
            ViewData["myemployee"] = emp;
            return View();
        }
    }
}
```

### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### Code for "Views\Home\Index.cshtml"

```
@using ViewDataWithModels.Models
<html>
<head>
    <title>ViewData with Models</title>
</head>
<body>
    <h1>ViewData with Models</h1>
    @{
        Employee e = (Employee) ViewData["myemployee"];
    }

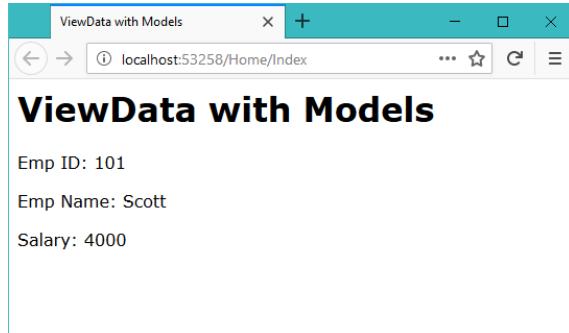
```

```
<h2>Emp ID: @e.EmpID</h2>
<h2>Emp Name: @e.EmpName</h2>
<h2>Salary: @e.Salary</h2>
</body>
</html>
```

### Running the application

- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Index".

Output:



## ViewBag

### ViewBag

- ViewBag was introduced in ASP.NET MVC 3. ViewBag internally uses "ViewData". ViewBag is also used to transfer (pass) the data from "controller" to the "view", when a request is sent from the browser. ViewBag's lifetime is "per request". When a browser sends a request to asp.net mvc web application, a new ViewBag will be created. Once the response is delivered to the browser, the ViewBag will be deleted automatically. ViewBag internally uses "dynamic". "Dynamic" means we can store any type of data; no need to declare its members.
- **Syntax:** ViewBag.key
- When you try to retrieve the value from ViewBag, it returns "null" if the given key is not found. You need not convert the the value, if you are assigning ViewBag's value into a variable.

## ViewBag - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "ViewBagExample". Type the location as "C:\Mvc". Type the solution name as "ViewBagExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating Employee.cs

- Open Solution Explorer. Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

### Code for "Employee.cs"

```
using System;

namespace ViewBagExample.Models
{
    public class Employee
    {
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public double Salary { get; set; }
    }
}
```

**Creating HomeController.cs**

- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

**Code for "HomeController.cs"**

```
using System;
using System.Web.Mvc;
using ViewBagExample.Models;

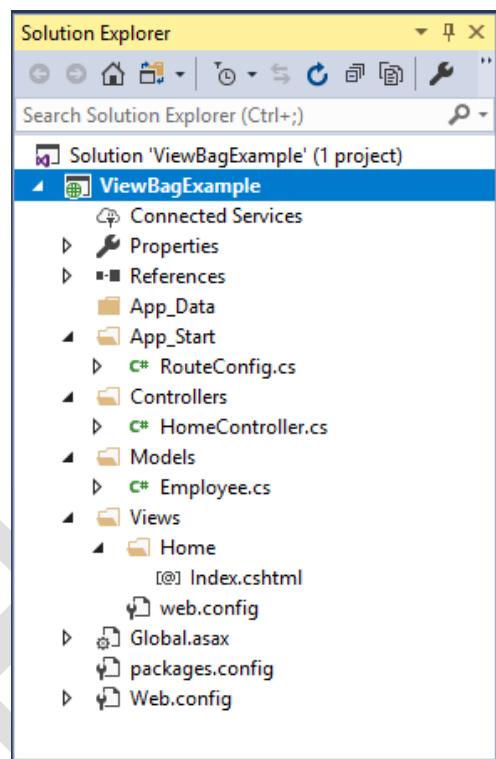
namespace ViewBagExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            ViewBag.message = "Hello";
            Employee emp = new Employee() { EmpID = 101, EmpName = "Scott",
            Salary = 4000 };
            ViewBag.myemployee = emp;
            return View();
        }
    }
}
```

**Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

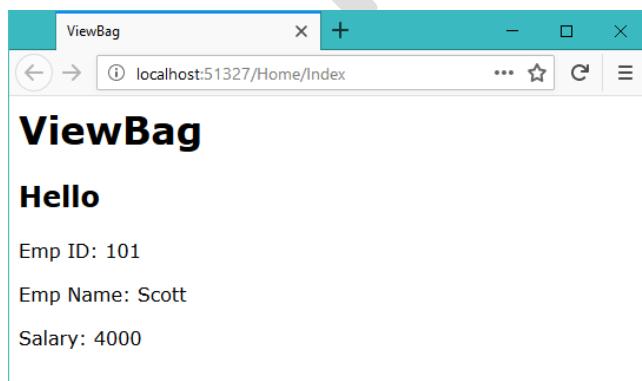
**Code for "Views\Home\Index.cshtml"**

```
<html>
<head>
<title>ViewBag</title>
</head>
<body>
<h1>ViewBag</h1>
<h2>@ViewBag.message</h2>
<p>Emp ID: @ViewBag.myemployee.EmpID</p>
<p>Emp Name: @ViewBag.myemployee.EmpName</p>
<p>Salary: @ViewBag.myemployee.Salary</p>
</body>
</html>
```

**Running the application**

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

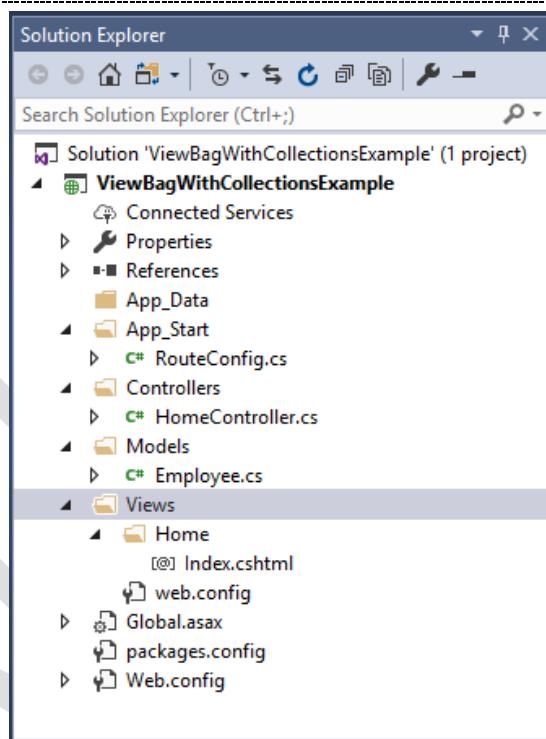
Output:



## ViewBag with Collections - Example

### **Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "ViewBagWithCollectionsExample". Type the location as "C:\Mvc". Type the solution name as "ViewBagWithCollectionsExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.



### **Creating Employee.cs**

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

### **Code for "Employee.cs"**

```
using System;

namespace ViewBagWithCollectionsExample.Models
{
    public class Employee
    {
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public double Salary { get; set; }
    }
}
```

### **Creating HomeController.cs**

- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### **Code for "HomeController.cs"**

```
using System;
using System.Collections.Generic;
using System.Web.Mvc;
using ViewBagWithCollectionsExample.Models;

namespace ViewBagWithCollectionsExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            List<Employee> emps = new List<Employee>()
            {
                new Employee() { EmpID = 1, EmpName = "Scott", Salary = 6677 },
                new Employee() { EmpID = 2, EmpName = "Allen", Salary = 2938 },
                new Employee() { EmpID = 3, EmpName = "John", Salary = 8832 }
            };
            ViewBag.myemployees = emps;
            return View();
        }
    }
}
```

### **Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Index.cshtml"**

```

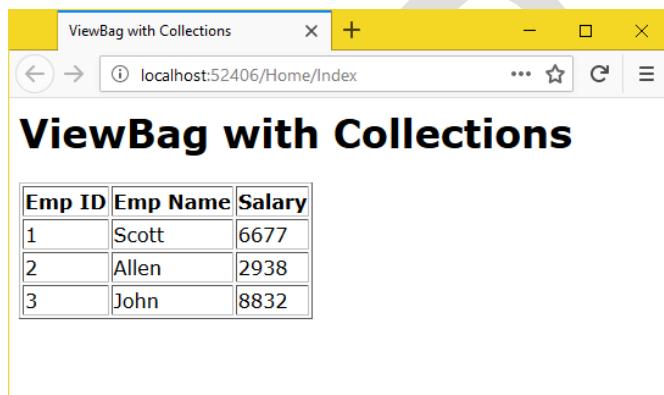
<html>
<head>
    <title>ViewBag with Collections</title>
</head>
<body>
    <h1>ViewBag with Collections</h1>
    <table border="1">
        <tr>
            <th>Emp ID</th>
            <th>Emp Name</th>
            <th>Salary</th>
        </tr>
        @foreach (var emp in ViewBag.myemployees)
        {
            <tr>
                <td>@emp.EmpID</td>
                <td>@emp.EmpName</td>
                <td>@emp.Salary</td>
            </tr>
        }
    </table>
</body>
</html>

```

**Running the application**

- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Index".

Output:

**TempData**

- TempData is used to transfer (pass) the data from "one controller" to "another controller" (or) from "one action method" to "another action method", in case of redirection. When a browser sends a request to asp.net mvc web application, a new TempData will be created. TempData's lifetime is "per request and subsequent request if redirection happens". That means when you redirect from one action method to another action method, a new request will be created, but the current TempData will be continued. Once the response is delivered to the browser, the TempData will be deleted automatically. TempData internally uses "Session".
- Syntax: TempData["key"]

**TempData - Example****Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "TempDataExample". Type the location as "C:\Mvc". Type the solution name as "TempDataExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

**Creating HomeController.cs**

- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

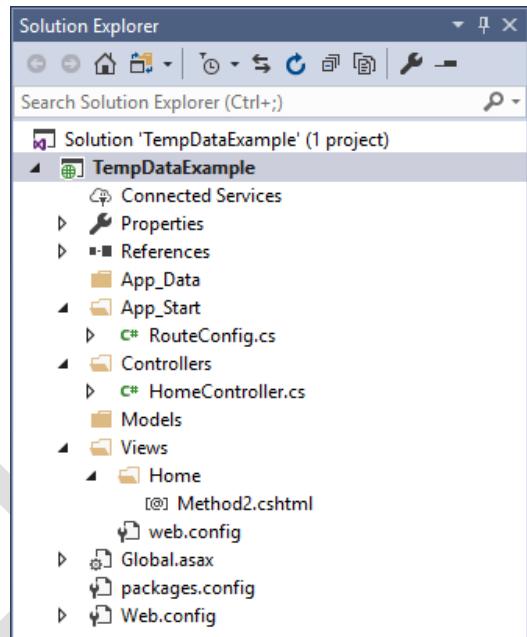
```

using System;
using System.Web.Mvc;

namespace TempDataExample.Controllers
{
    public class HomeController : Controller
    {
        //Request 1
        public ActionResult Method1()
        {
            TempData["msg"] = "Hello World";
            return RedirectToAction("Method2", "Home");
        }

        //Request 2
        public ActionResult Method2()
        {
            ViewBag.msg = TempData["msg"]; //Output: Hello World
            return View();
        }
    }
}

```



### Creating Method2.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Method2". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### Code for "Views\Home\Method2.cshtml"

```

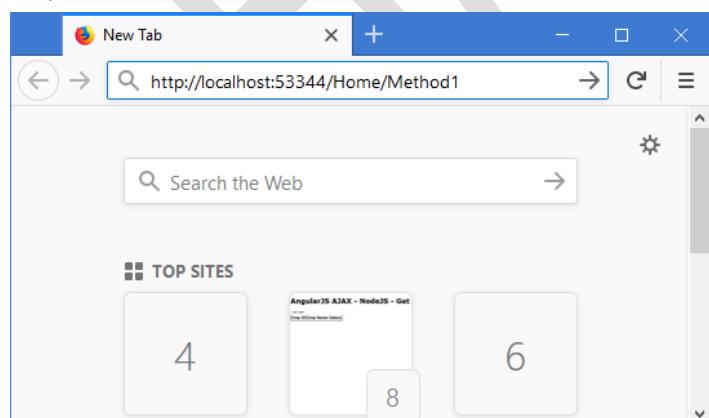
<html>
<head>
    <title>Method2</title>
</head>
<body>
    <h1>TempData</h1>
    <h2>msg: @ViewBag.msg</h2>
</body>
</html>

```

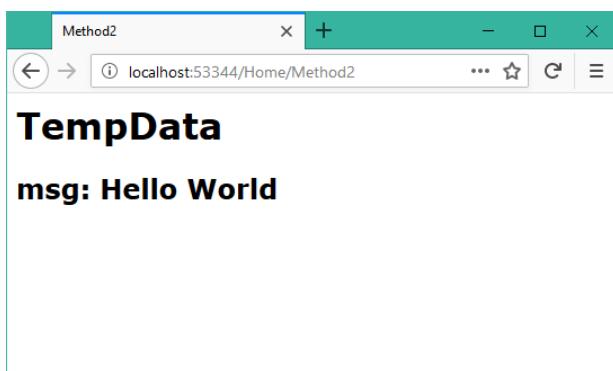
### Running the application

- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Method1".

Output:



After pressing Enter.



### TempData – Third Result - Example

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "TempDataThirdRequest". Type the location as "C:\Mvc". Type the solution name as "TempDataThirdRequest". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

#### Creating HomeController.cs

- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### Code for "HomeController.cs"

```

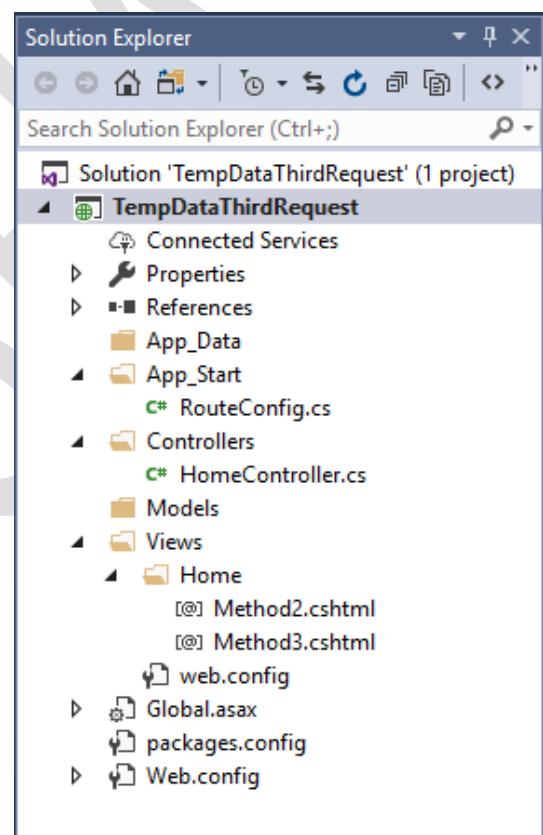
using System;
using System.Web.Mvc;

namespace TempDataThirdRequest.Controllers
{
    public class HomeController : Controller
    {
        //Request 1
        public ActionResult Method1()
        {
            TempData["msg"] = "Hello World";
            return RedirectToAction("Method2", "Home");
        }

        //Request 2
        public ActionResult Method2()
        {
            ViewBag.msg = Convert.ToString(TempData["msg"]); //Output: Hello
            //Will be marked for deletion automatically
            return View();
        }

        //Request 3
        public ActionResult Method3()
        {
            ViewBag.msg = TempData["msg"]; //Output: null
            return View();
        }
    }
}

```



### Creating Method2.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Method2". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### Code for "Views\Home\Method2.cshtml"

```
<html>
<head>
<title>Method2</title>
</head>
<body>
<h1>TempData</h1>
<h2>msg: @ViewBag.msg</h2>
</body>
</html>
```

### Creating Method3.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Method3". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

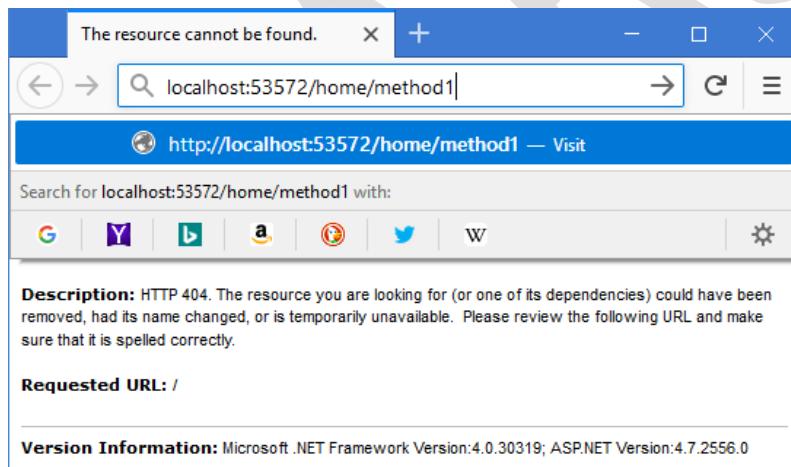
### Code for "Views\Home\Method3.cshtml"

```
<html>
<head>
<title>Method3</title>
</head>
<body>
<h1>TempData</h1>
<h2>msg: @ViewBag.msg</h2>
</body>
</html>
```

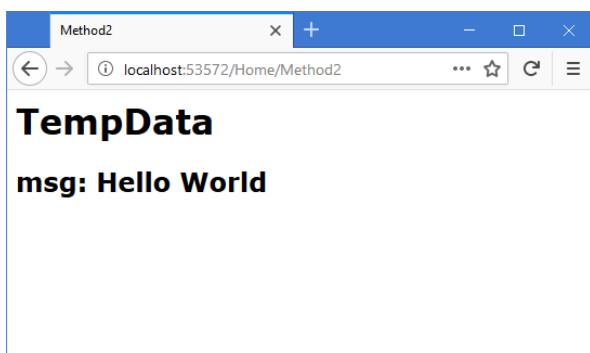
### Running the application

- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Method1".

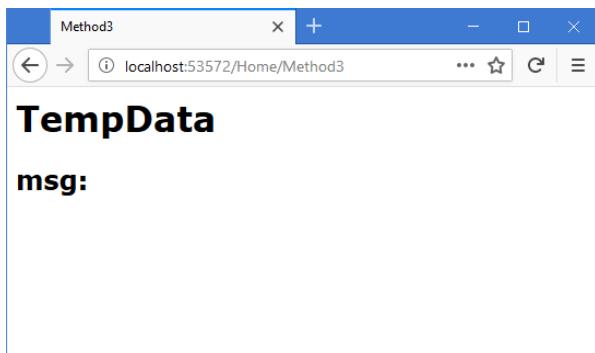
Output:



After pressing Enter.



http://localhost:portnumber/Home/Method3



## TempData – Keep - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "TempDataKeepExample". Type the location as "C:\Mvc". Type the solution name as "TempDataKeepExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating HomeController.cs

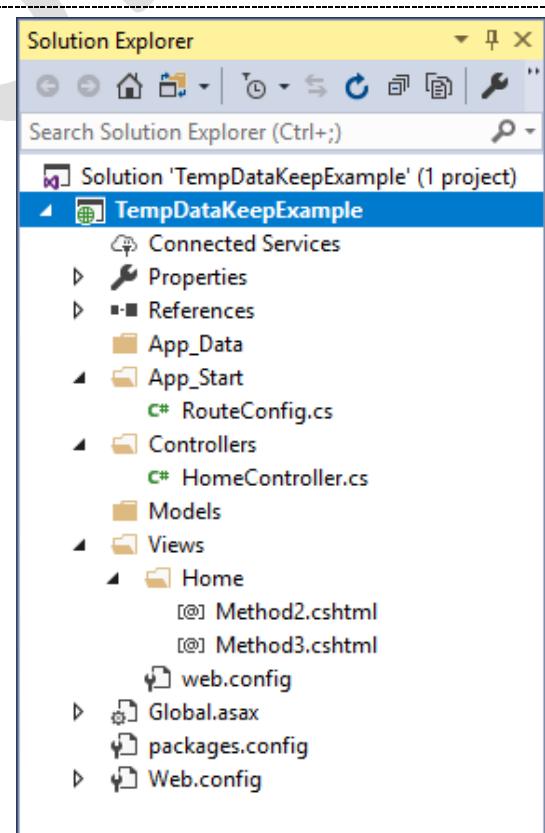
- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;

namespace TempDataKeepExample.Controllers
{
    public class HomeController : Controller
    {
        //Request 1
        public ActionResult Method1()
        {
            TempData["msg"] = "Hello World";
            return RedirectToAction("Method2", "Home");
        }

        //Request 2
        public ActionResult Method2()
        {
            ViewBag.msg = Convert.ToString(TempData["msg"]); //Output: Hello World //Will be marked for deletion automatically
            TempData.Keep("msg"); // msg will be kept in memory for next request ("Request 3") also
        }
    }
}
```



```

        return View();
    }

//Request3
public ActionResult Method3()
{
    ViewBag.msg = TempData["msg"]; //Output: Hello World //Will be marked for deletion automatically
    return View();
}
}
}
}

```

**Creating Method2.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Method2". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Method2.cshtml"**

```

<html>
<head>
    <title>Method2</title>
</head>
<body>
    <h1>TempData - Keep</h1>
    <h2>msg: @ViewBag.msg</h2>
</body>
</html>

```

**Creating Method3.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Method3". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Method3.cshtml"**

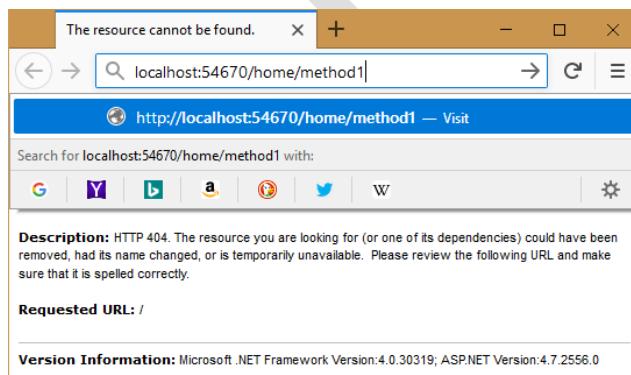
```

<html>
<head>
    <title>Method3</title>
</head>
<body>
    <h1>TempData - Keep</h1>
    <h2>msg: @ViewBag.msg</h2>
</body>
</html>

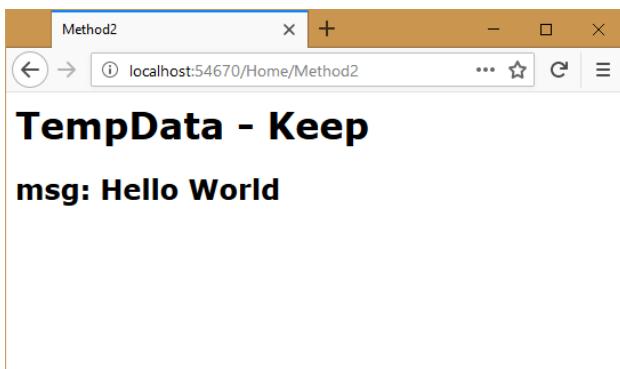
```

**Running the application**

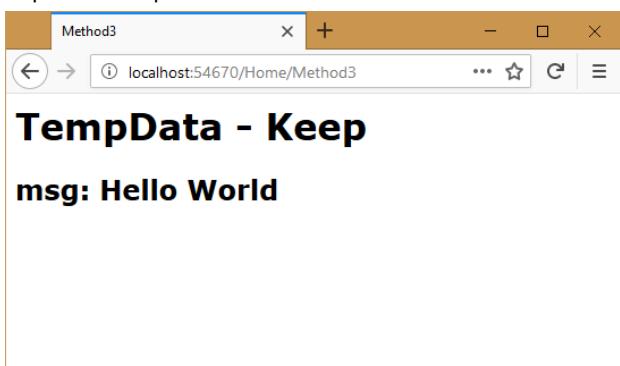
- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Method1".

**Output:**

After pressing Enter.



<http://localhost:portnumber/Home/Method3>



## TempData – Peek - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "TempData.PeekExample". Type the location as "C:\Mvc". Type the solution name as "TempData.PeekExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating HomeController.cs

- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;

namespace TempData.PeekExample.Controllers
{
    public class HomeController : Controller
    {
        //Request1
        public ActionResult Method1()
        {
            TempData["msg"] = "Hello World";
            return RedirectToAction("Method2", "Home");
        }

        //Request2
        public ActionResult Method2()
        {
            ViewBag.msg = Convert.ToString(TempData.Peek("msg")); //Output: Hello World // "msg" will be kept in memory for next
            request ("Request 3") also
        }
    }
}
```

```

        return View();
    }

//Request3
public ActionResult Method3()
{
    ViewBag.msg = TempData["msg"]; //Output: Hello World //Will be
    marked for deletion automatically
    return View();
}
}
}

```

#### Creating Method2.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Method2". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### Code for "Views\Home\Method2.cshtml"

```

<html>
<head>
<title>Method2</title>
</head>
<body>
<h1>TempData - Peek</h1>
<h2>msg: @ViewBag.msg</h2>
</body>
</html>

```

#### Creating Method3.cshtml

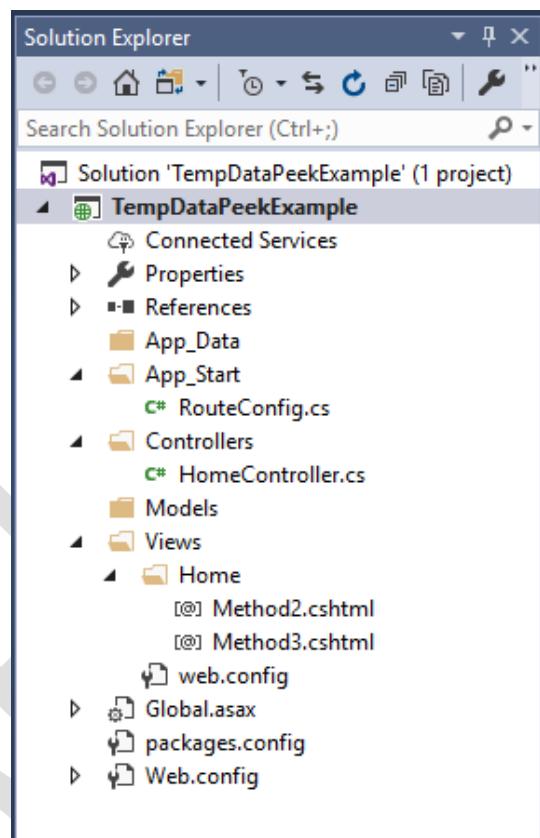
- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Method3". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### Code for "Views\Home\Method3.cshtml"

```

<html>
<head>
<title>Method3</title>
</head>
<body>
<h1>TempData - Peek</h1>
<h2>msg: @ViewBag.msg</h2>
</body>
</html>

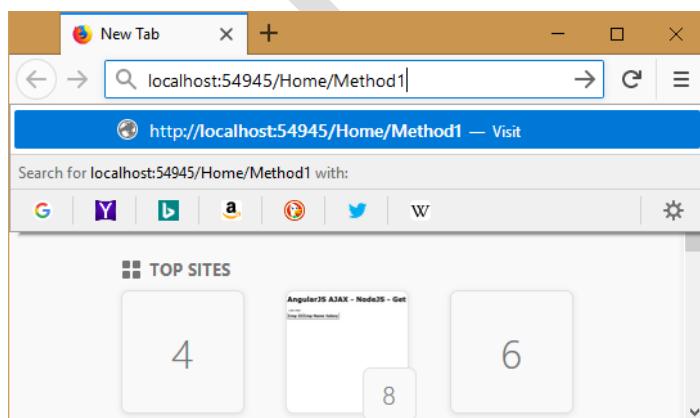
```



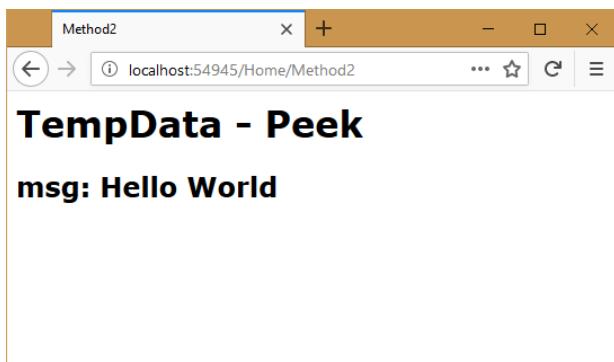
#### Running the application

- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Method1".

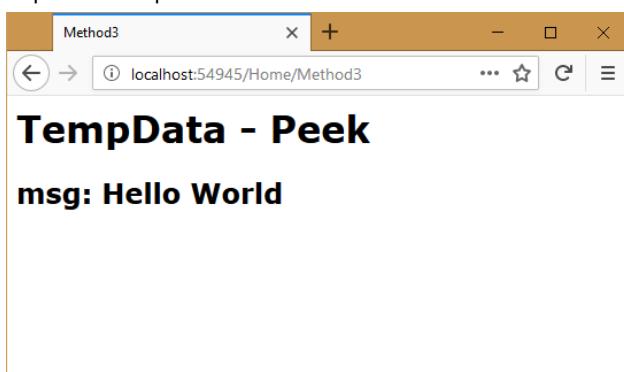
Output:



After pressing Enter.



http://localhost:portnumber/Home/Method3



## Session

- “Session” is a memory area on server, in which you can store user-specific data.
- For every browser, server maintains a session. Session will be created when a browser sends the first request and will be deleted automatically after session expiration time (default is 20 minutes). Session is used to transfer (pass) the data from one web page to another web page of the same web site and running on the same browser.
- **Note:** “ASP.NET MVC Session” is same as “asp.net Session”. Session internally uses Dictionary.
- **Syntax:** Session["key"]

## Session - Example

### Creating Project

- Open Visual Studio 2017. Go to “File” – “New” – “Project”. Select “.NET Framework 4.7”. Select “Visual C#”. Select “ASP.NET Web Application (.NET Framework)”. Type the project name “SessionExample”. Type the location as “C:\Mvc”. Type the solution name as “SessionExample”. Click on OK. Select “Empty”. Check the checkbox “MVC”. Uncheck the checkbox “Enable Docker support”. Uncheck the checkbox “Add unit tests”. Click on OK. Open Solution Explorer.

### Creating HomeController.cs

- Open Solution Explorer. Right click on “Controllers” folder and click on “Add” – “Controller”. Select “MVC 5 Controller – Empty”. Click on “Add”. Type the controller name as “HomeController”. Click on “Add”.

### Code for “HomeController.cs”

```
using System;
using System.Web.Mvc;

namespace SessionExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
```

```

Session["message"] = "Hello";
return View();
}
}
}

```

#### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### Code for "Views\Home\Index.cshtml"

```

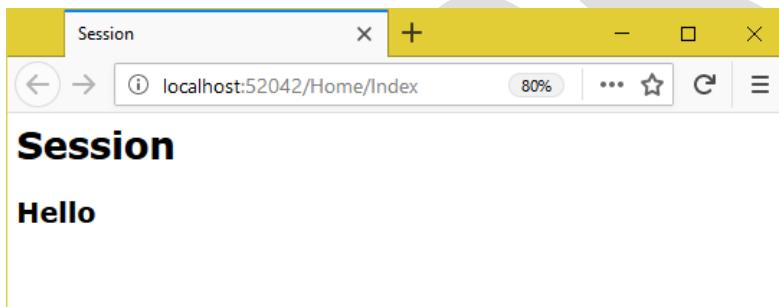
<html>
<head>
<title>Session</title>
</head>
<body>
<h1>Session</h1>
<h2>@Session["message"]</h2>
</body>
</html>

```

#### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

#### Output



### Session – With Objects - Example

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "SessionWithObjectsExample". Type the location as "C:\Mvc". Type the solution name as "SessionWithObjectsExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK. Open Solution Explorer.

#### Creating Employee.cs

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

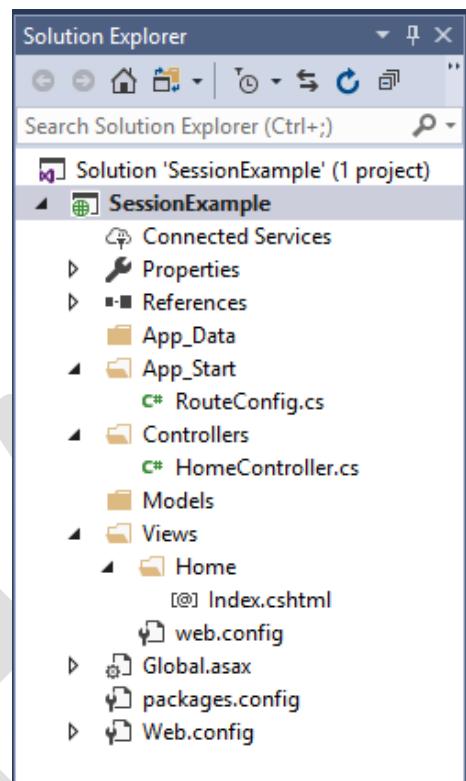
#### Code for "Employee.cs"

```

using System;

namespace SessionWithObjectsExample.Models
{
    public class Employee
    {
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public double Salary { get; set; }
    }
}

```



```
    }
}
```

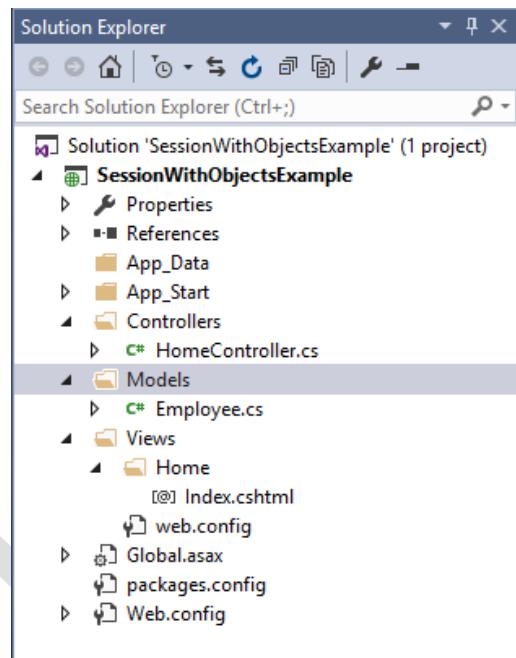
### **Creating HomeController.cs**

- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### **Code for "HomeController.cs"**

```
using System;
using System.Web.Mvc;
using SessionWithObjectsExample.Models;

namespace SessionWithObjectsExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            Employee emp = new Employee();
            emp.EmpID = 101;
            emp.EmpName = "scott";
            emp.Salary = 4000;
            Session["myemployee"] = emp;
            return View();
        }
    }
}
```



### **Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

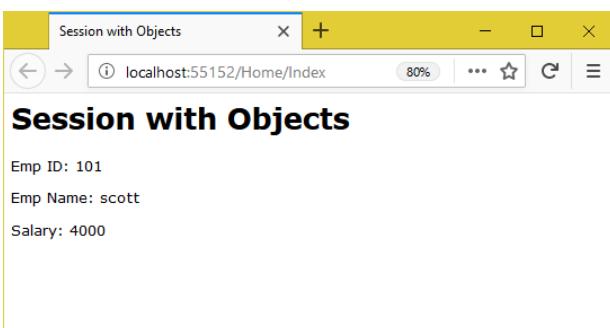
### **Code for "Views\Home\Index.cshtml"**

```
@using SessionWithObjectsExample.Models
<html>
    <head>
        <title>Session with Objects</title>
    </head>
    <body>
        <h1>Session with Objects</h1>
        @{
            Employee emp = (Employee)Session["myemployee"];
        }
        <h2>Emp ID: @emp.EmpID</h2>
        <h2>Emp Name: @emp.EmpName</h2>
        <h2>Salary: @emp.Salary</h2>
    </body>
</html>
```

### **Running the application**

- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Index".

Output:



### ViewData (vs) ViewBag (vs) TempData (vs) Session

- ViewData and ViewBag are purpose-wise same; syntax-wise different. ViewData and ViewBag both are stored in the same place internally. They are only two syntaxes at retrieving time. However they have some following differences.

Sl. No	ViewData	ViewBag	TempData	Session
1	ViewData is a dictionary object that is derived from ViewDataDictionary class.	ViewBag is a dynamic property that takes advantage of the new dynamic features in C# 4.0. Basically, it is a wrapper around the ViewData and also used to pass data from controller to corresponding view.	TempData is a dictionary object that is derived from TempDataDictionary class and stored in short lives session.	Session is an object that is derived from HttpSessionState class.
2	public ViewDataDictionary ViewData { get; set; }	public dynamic ViewBag { get; }	public TempDataDictionary TempData { get; set; }	public HttpSessionState Session { get; }
3	Available in all versions of MVC	Introduced in MVC 3.0	Available in all versions of MVC	Available in all versions of MVC
4	ViewData is a property of ControllerBase class.	ViewBag is a property of ControllerBase class.	TempData is a property of ControllerBase class.	Session is a property of HttpContext class.
5	Its life lies only during the current request.	Its life lies only during the current request.	Its life is very short and lies only in the current request and also in the subsequent request(s) in case of redirect from action to action.	Session is also used to pass data within the ASP.NET MVC application and Unlike TempData, it persists for its expiration time (by default session expiration time is 20 minutes but it can be increased).  Session is valid for all requests, not for a single redirect.
6	If redirection occurs then its value becomes null.	If redirection occurs then its value becomes null.	If redirection cycle ends, then its value becomes null.	If the current session timeout is over (by default it is 20 minutes, and of course it can be changed), the value will be null.
7	Need to typecast to get the value.	No need to typecast to get the value.	Need to typecast to get the value.	Need to typecast to get the value.
8	Usually used to store the model data.	Usually used to store model data.	Usually used to store validation error messages etc.	Usually used to store current user name etc.

### Application

- "Application" (also known as "Application state") is a memory area on server, in which you can store common data that belongs to all the users of the web site. For all the browsers, server maintains a common application state. Application state will be created when the first browser sends the first request and will be deleted automatically while server shutdown. Application state is commonly accessible in any page of the same web site, running in any browser. Application state is accessible in both controller and view. **Note:** "ASP.NET MVC Application state" is same as "Asp.net Application state". Application state internally uses Dictionary.

Syntax (in Controller):      `ControllerContext.HttpContext.Application["key"]`

Syntax (in View):      `Context.Application["key"]`

## Application - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "ApplicationExample". Type the location as "C:\Mvc". Type the solution name as "ApplicationExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

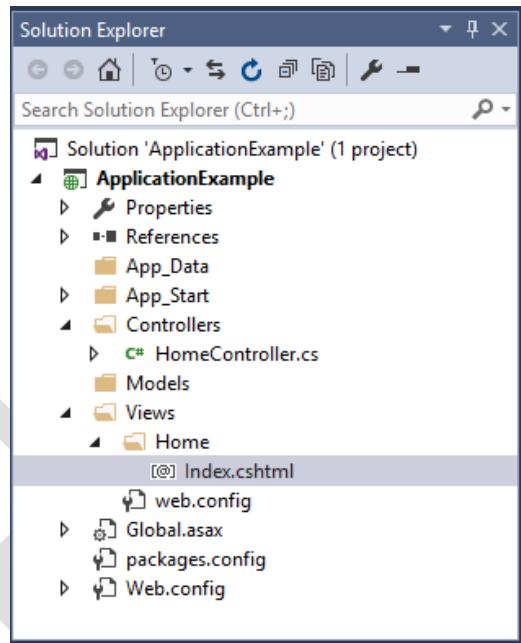
### Creating HomeController.cs

- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;

namespace ApplicationExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            ControllerContext.HttpContext.Application["x"] = 100;
            return View();
        }
    }
}
```



### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

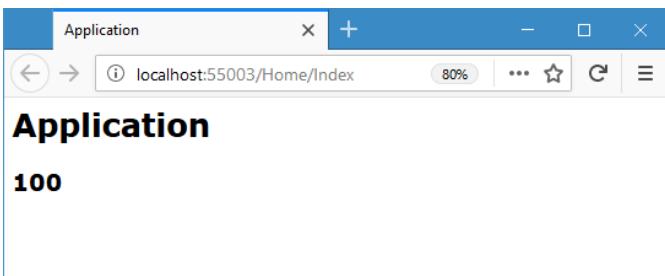
### Code for "Views\Home\Index.cshtml"

```
<html>
  <head>
    <title>Application</title>
  </head>
  <body>
    <h1>Application</h1>
    <h2>@Context.Application["x"]</h2>
  </body>
</html>
```

### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

### Output



## Cache

- "Cache" (also known as "Data Caching") is similar to "Application state". The difference is:
  - **Application state:** Doesn't support to set an "expiration time" for the data. So the data will be stored permanently while server is running.
  - **Cache:** Supports to set an "expiration time" (Ex: 5 minutes) for the data. So the data will be automatically deleted in the cache memory, after given time is completed.
- In realtime, you can store the frequently-used database data in cache memory, so that later you can get it easily from cache memory instead of from database. Retrieving data from cache memory is faster than retrieving data from database.
 

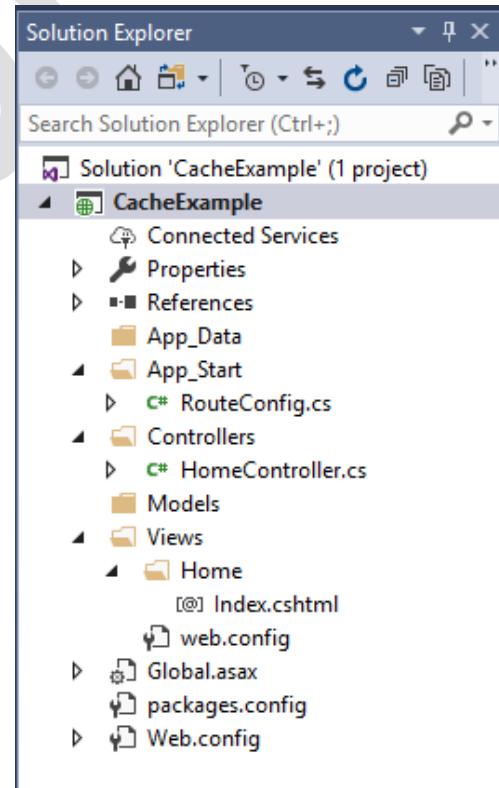
**Syntax (in Controller):** ControllerContext.HttpContext.Cache("key", "value", null, DateTime.Now.AddMinutes(no. of minutes), TimeSpan.Zero)

**Syntax (in View):** Cache["key"]

## Cache - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "CacheExample". Type the location as "C:\Mvc". Type the solution name as "CacheExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.



### Creating HomeController.cs

- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;

namespace CacheExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            HttpContext.Cache.Insert("x", 100, null,
            DateTime.Now.AddMinutes(5), TimeSpan.Zero);
            return View();
        }
    }
}
```

### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### Code for "Views\Home\Index.cshtml"

```
<html>
<head>
```

```

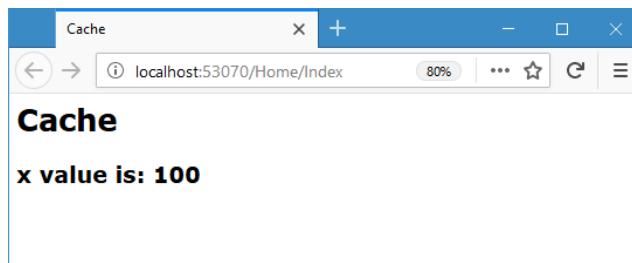
<title>Cache</title>
</head>
<body>
  <h1>Cache</h1>
  <h2>x value is: @Cache["x"]</h2>
</body>
</html>

```

#### Running the application

- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Index".

Output:



## Cookies

- "Cookies" are also used to store "browser-specified data", similar to "Session".
- The difference is:
  - Session:** Will be stored in the server.
  - Cookies:** Will be stored in the client (browser).
- Cookies are stored as "temporary files", in the client system (in the browser-specific folder). Cookies are not secured. "Request.Cookies" represent the cookies that are sent from browser to server. "Response.Cookies" represent the cookies that are to be given from the server to browser.

#### Sending a cookie from "server" to "browser"

```

System.Web.HttpCookie variable;
Variable = new System.Web.HttpCookie("key", "value");
Response.Cookies.Add(variable);

```

#### Receiving the cookie from "browser" to "server"

```
Request.Cookies["key"]
```

## Cookies - Example

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "CookiesExample". Type the location as "C:\Mvc". Type the solution name as "CookiesExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

#### Creating HomeController.cs

- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### Code for "HomeController.cs"

```

using System;
using System.Web.Mvc;
using System.Web;

namespace CookiesExample.Controllers
{
  public class HomeController : Controller
  {

```

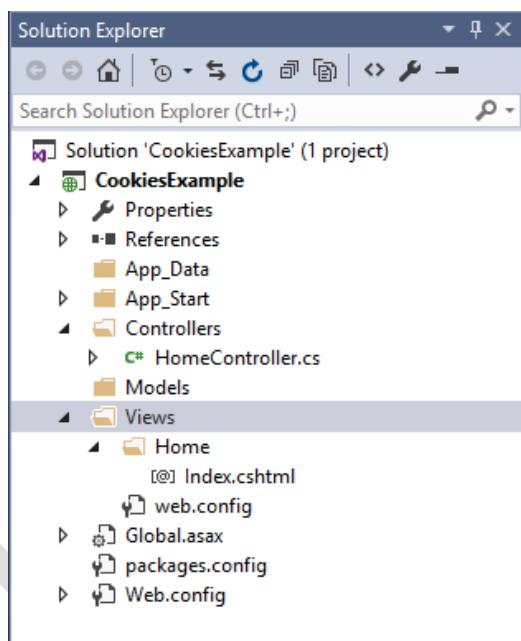
```
public ActionResult Index()
{
    HttpCookie ck = new HttpCookie("username", "scott");
    Response.Cookies.Add(ck);
    return View();
}
```

### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### Code for "Views\Home\Index.cshtml"

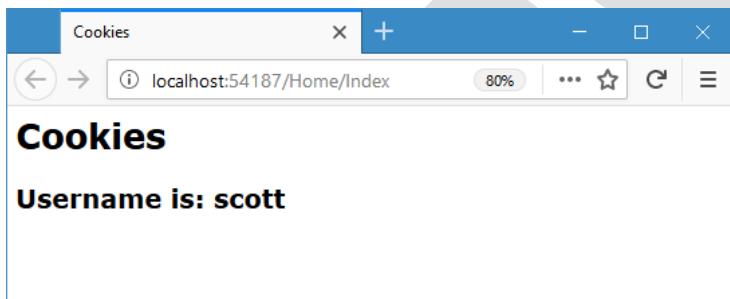
```
<html>
<head>
<title>Cookies</title>
</head>
<body>
<h1>Cookies</h1>
<h2>Username is: @Request.Cookies["username"].Value</h2>
</body>
</html>
```



### Running the application

- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Index".

Output:



## SHARED VIEWS

### What is Shared View

- Shared views are present in the "Views\Shared" folder. Shared views can be called from any controller of the same mvc project. For example, there are two controllers called "Controller1Controller" and "Controller2Controller". The views that are in the "Views\Controller1" folder are accessible from the "Controller1Controller" only. The views that are in the "Views\Controller2" folder are accessible from the "Controller2Controller" only. The views that are present in the "Views\Shared" folder are accessible from both "Controller1Controller" and also from "Controller2Controller". So, the "Views\Shared" folder contains the common views of all the controllers. The name "Views\Shared" is a fixed name, we can't change it. When you call a view, ASP.NET MVC first checks in "Views\Controllername" folder and then it checks in "Views\Shared" folder.

### Shared Views - Example

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "SharedViewsExample". Type the location as "C:\Mvc". Type the solution name as "SharedViewsExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

#### Creating Controller1Controller.cs

- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "Controller1Controller". Click on "Add".

### Code for "Controller1Controller.cs"

```
using System;
using System.Web.Mvc;

namespace SharedViewsExample.Controllers
{
    public class Controller1Controller : Controller
    {
        public ActionResult Page1()
        {
            return View();
        }
    }
}
```

### Creating Controller2Controller.cs

- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "Controller2Controller". Click on "Add".

### Code for "Controller2Controller.cs"

```
using System;
using System.Web.Mvc;

namespace SharedViewsExample.Controllers
{
    public class Controller2Controller : Controller
    {
        public ActionResult Page1()
        {
            return View();
        }
    }
}
```

### Creating Page1.cshtml

- Right click on "Views" and click on "Add" - "New Folder". Type the folder name as "Shared" and press Enter. Right click on "Shared" folder and click on "Add" - "View". Type the view name as "Page1". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### Code for "Views\Shared\Page1.cshtml"

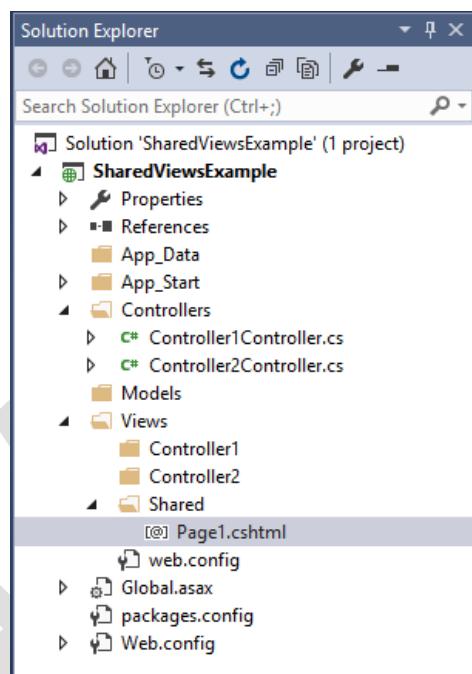
```
<html>
  <head>
    <title>Page1</title>
  </head>
  <body>
    <h1>This is Page1 (shared view)</h1>
  </body>
</html>
```

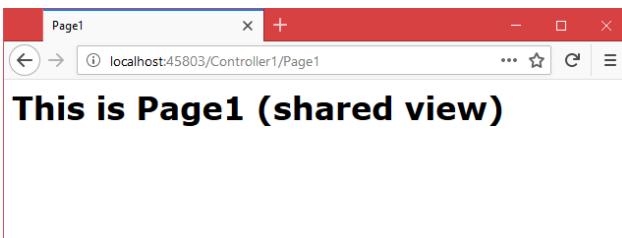
### Running the application

- Press "F5" to run the application. Type "http://localhost:portnumber/Controller1/Page1".
- And also "http://localhost:portnumber/Controller2/Page1".

http://localhost:portnumber/Controller1/Page1

Output:





http://localhost:portnumber/Controller2/Page1

Output:



## LAYOUT VIEWS

### What is Layout View

- Layout Views (or) Layout Pages are just like master pages in asp.net web forms. Layout pages contain "page template". Layout pages contain the common content (such as header, footer, side bar etc.) that should be displayed in every page. First we have to create a layout page and we have to create normal views based on the layout pages.

#### Execution flow of layout page:

Browser → Controller → View → Layout View → Generate View Result → Controller → Browser

#### Syntax of layout page:

```
<html>
  <head>
    <title>title here</title>
  </head>
  <body>
    Header here
    <div>
      @RenderBody()
    </div>
    Footer here
  </body>
</html>
```

- The `@RenderBody()` method displays the actual content of the web page.

## Layout Views - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "LayoutViewsExamples". Type the location as "C:\Mvc". Type the solution name as "LayoutViewsExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating HomeController.cs

- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

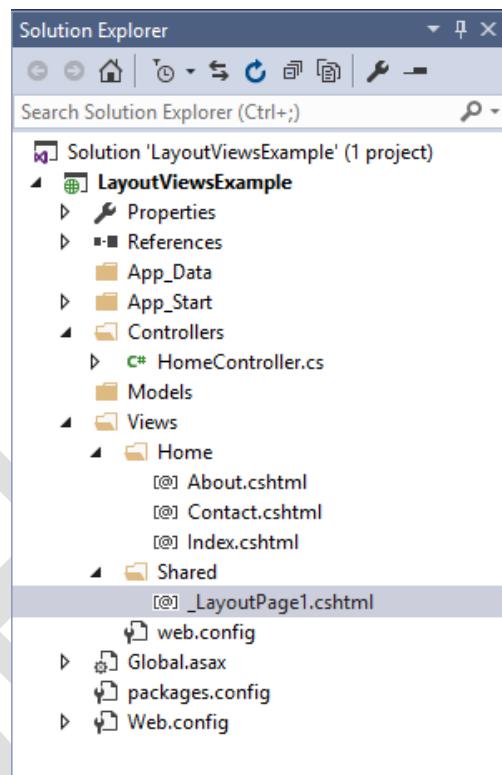
### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;

namespace LayoutViewsExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        public ActionResult About()
        {
            return View();
        }

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



### Creating \_LayoutPage1.cshtml

- Right click on "Views" and click on "Add" - "New Folder". Type the folder name as "Shared" and press Enter. Right click on "Views\Shared" folder and click on "Add" - "New Item". Click on "Web" - "MVC" - "MVC 5 Layout Page (Razor)". Type the file name as "\_LayoutPage1.cshtml". Click on "Add".

### Code for "Views\Shared\\_LayoutPage1.cshtml"

```
<html>
  <head>
    <title>@ViewBag.Title</title>
  </head>
  <body>
    <div>
      <h1>Header</h1>
    </div>
    <div style="background-color:lightblue">
      <a href="/Home/Index">Index</a>
      <a href="/Home/About">About</a>
      <a href="/Home/Contact">Contact</a>
    </div>
    <div>
      @RenderBody()
    </div>
  </body>
</html>
```

```
<div>
<p>Footer</p>
</div>
</body>
</html>
```

**Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Check the checkbox "Use a layout page" and select the path as "~/Views/Shared/\_LayoutPage1.cshtml". Click on "Add".

**Code for "Views\Home\Index.cshtml"**

```
@{
    ViewBag.Title = "Index";
    Layout = "~/Views/Shared/_LayoutPage1.cshtml";
}
<h1>Index page content</h1>
```

**Creating About.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "About". Select the template "Empty (without model)". Check the checkbox "Use a layout page" and select the path as "~/Views/Shared/\_LayoutPage1.cshtml". Click on "Add".

**Code for "Views\Home\About.cshtml"**

```
@{
    ViewBag.Title = "About";
    Layout = "~/Views/Shared/_LayoutPage1.cshtml";
}
<h1>About page content</h1>
```

**Creating Contact.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Contact". Select the template "Empty (without model)". Check the checkbox "Use a layout page" and select the path as "~/Views/Shared/\_LayoutPage1.cshtml". Click on "Add".

**Code for "Views\Home>Contact.cshtml"**

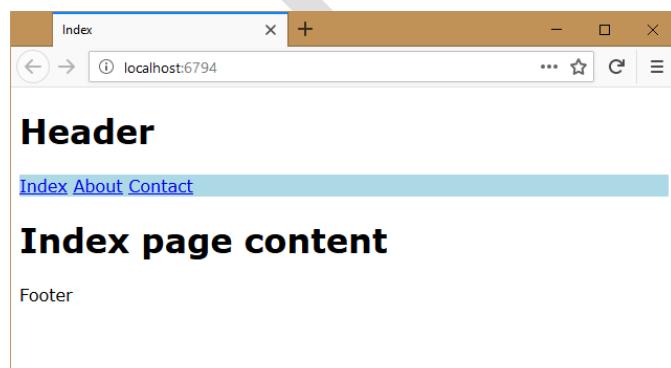
```
@{
    ViewBag.Title = "Contact";
    Layout = "~/Views/Shared/_LayoutPage1.cshtml";
}
<h1>Contact page content</h1>
```

**Running the application**

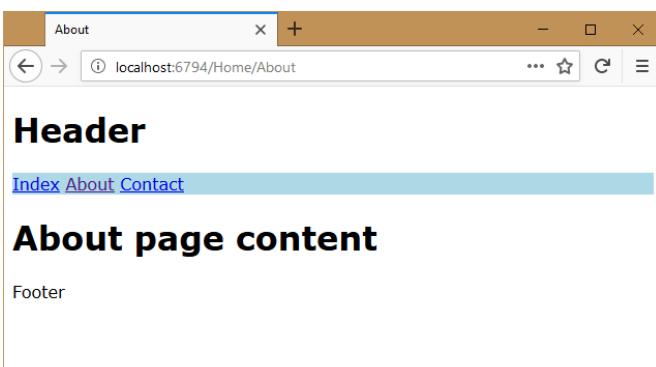
- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Index".

Output:

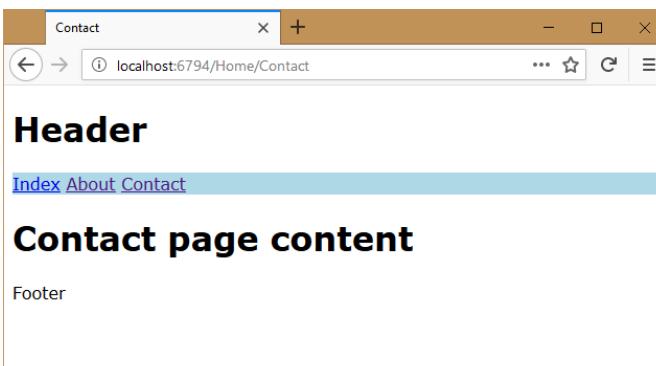
<http://localhost:portnumber/Home/Index>



<http://localhost:portnumber/Home/About>



http://localhost:portnumber/Home/Contact



## \_ViewStart.cshtml

- The “\_ViewStart.cshtml” (fixed name) is a special file that can be present either in “Views” folder or in “Views\controllername” folder. If it is present in “Views” folder, it specifies the path of the default layout view of all the views in the entire project. If it is present in “Views\controllername” folder, it specifies the path of layout view of all the views in the same folder only. Before executing the view, ASP.NET MVC calls the “\_ViewStart.cshtml” file automatically, if it is present. You can override the setting of “\_ViewStart.cshtml”, in any view, by re-assigning the value of “Layout” property. We can't write any extra code in the “\_ViewStart.cshtml” file, except setting a layout view.

### Syntax of “Views\\_ViewStart.cshtml”

```
@[
Layout = "~/Views/foldername/layout file name.cshtml";
]
```

## ViewStart.cshtml - Example

### Creating Project

- Open Visual Studio 2017. Go to “File” – “New” – “Project”. Select “.NET Framework 4.7”. Select “Visual C#”. Select “ASP.NET Web Application (.NET Framework)”. Type the project name “ViewStartExample”. Type the location as “C:\Mvc”. Type the solution name as “ViewStartExample”. Click on OK. Select “Empty”. Check the checkbox “MVC”. Uncheck the checkbox “Enable Docker support”. Uncheck the checkbox “Add unit tests”. Click on OK.

### Creating HomeController.cs

- Open Solution Explorer. Right click on “Controllers” folder and click on “Add” – “Controller”. Select “MVC 5 Controller – Empty”. Click on “Add”. Type the controller name as “HomeController”. Click on “Add”.

### Code for “HomeController.cs”

```
using System;
using System.Web.Mvc;

namespace ViewStartExample.Controllers
{
    public class HomeController : Controller
```

```

{
    public ActionResult Index()
    {
        return View();
    }

    public ActionResult About()
    {
        return View();
    }

    public ActionResult Contact()
    {
        return View();
    }
}

```

**Creating \_LayoutPage1.cshtml**

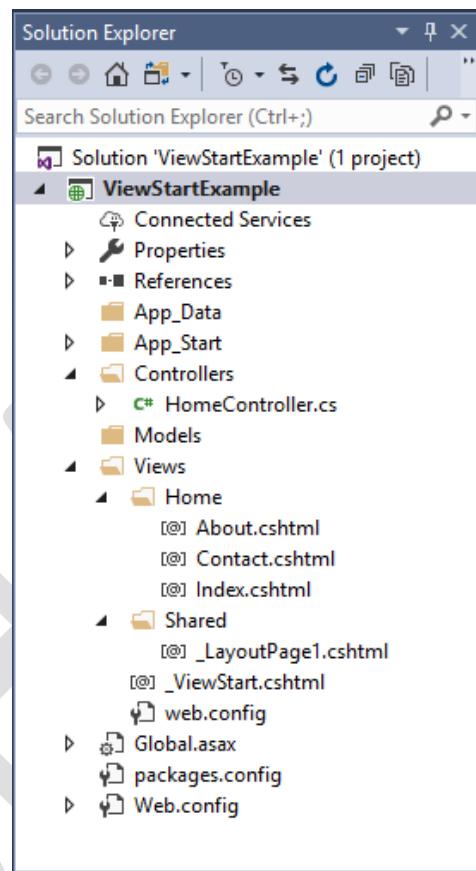
- Right click on "Views" and click on "Add" - "New Folder". Type the folder name as "Shared" and press Enter. Right click on "Views\Shared" folder and click on "Add" - "New Item". Click on "Web" - "MVC" - "MVC 5 Layout Page (Razor)". Type the file name as "\_LayoutPage1.cshtml". Click on "Add".

**Code for "Views\Shared\\_LayoutPage1.cshtml"**

```

<html>
    <head>
        <title>@ViewBag.Title</title>
    </head>
    <body>
        <div>
            <h1>Header</h1>
        </div>
        <div style="background-color:lightblue">
            <a href="/Home/Index">Index</a>
            <a href="/Home/About">About</a>
            <a href="/Home/Contact">Contact</a>
        </div>
        <div>
            @RenderBody()
        </div>
        <div>
            <p>Footer</p>
        </div>
    </body>
</html>

```

**Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Check the checkbox "Use a layout page". Click on "Add".

**Code for "Views\Home\Index.cshtml"**

```

@{
    ViewBag.Title = "Index";
}
<h1>Home page content</h1>

```

**Creating About.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "About". Select the template "Empty (without model)". Check the checkbox "Use a layout page". Click on "Add".

**Code for "Views\Home\About.cshtml"**

```

@{

```

```

ViewBag.Title = "About";
}
<h1>About page content</h1>

```

**Creating Contact.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Contact". Select the template "Empty (without model)". Check the checkbox "Use a layout page". Click on "Add".

**Code for "Views\Home\Contact.cshtml"**

```

@{
    ViewBag.Title = "Contact";
}
<h1>Contact page content</h1>

```

**Creating \_ViewStart.cshtml**

- Right click on "Views" folder and click on "Add" - "View". Type the view name as "\_ViewStart". Select the template "Empty (without model)". Check the checkbox "Use a layout page". Click on "Add".

**Code for "Views\Home\\_ViewStart.cshtml"**

```

@{
    Layout = "~/Views/Shared/_LayoutPage1.cshtml";
}

```

**Running the application**

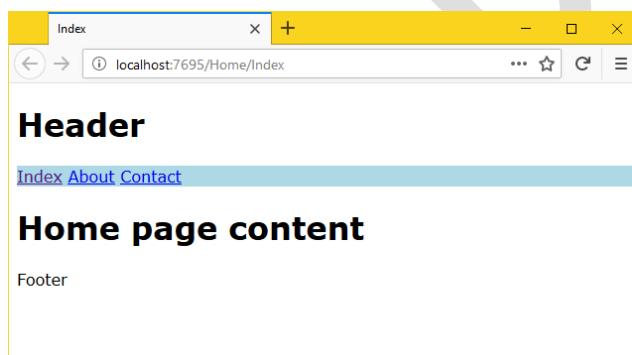
- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Index".

**Running the application**

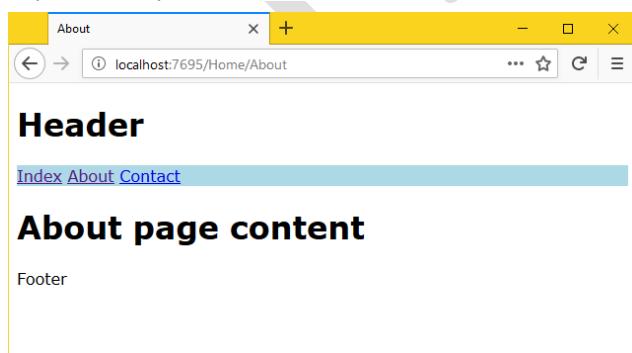
- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Index".

**Output:**

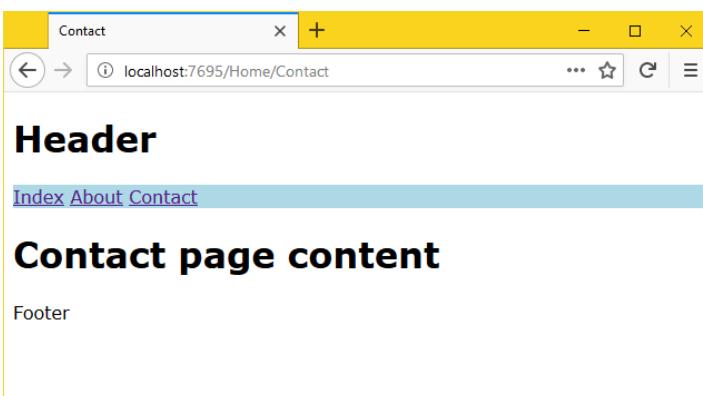
<http://localhost:portnumber/Home/Index>



<http://localhost:portnumber/Home/About>



<http://localhost:portnumber/Home/Contact>



## Sections in Layout Views

- Sections are like "content place holders" in asp.net web forms. Sections are used to display view-specific content in the layout view. Sections are defined in the normal view and rendered in the layout view.

### Syntax to define a section in the view

```
@section sectionnamehere
{
    Content here
}
```

### Syntax to render (display) the section in the layout view

```
@RenderSection("sectionnamehere", required: false)
```

**Note:** The "required:false" option makes the section as optional.

## Sections in Layout Views - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "LayoutViewsSectionExample". Type the location as "C:\Mvc". Type the solution name as "LayoutViewsSectionExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests".
- Click on OK. Open solution explorer. Right click on the project name (LayoutViewsSectionExample) and click on "Add" - "New Folder". Type the folder name as "Images" and press Enter key. Copy and paste the following images into "Images" folder in solution explorer.
  - ❖ India.png
  - ❖ Uk.png
  - ❖ Us.png

### Creating HomeController.cs

- Right click on 'Controllers' folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;

namespace LayoutViewsSectionExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View("India");
        }
    }
}
```

```

}

public ActionResult India()
{
    return View();
}

public ActionResult UK()
{
    return View();
}

public ActionResult US()
{
    return View();
}
}

```

#### **Creating \_LayoutPage1.cshtml**

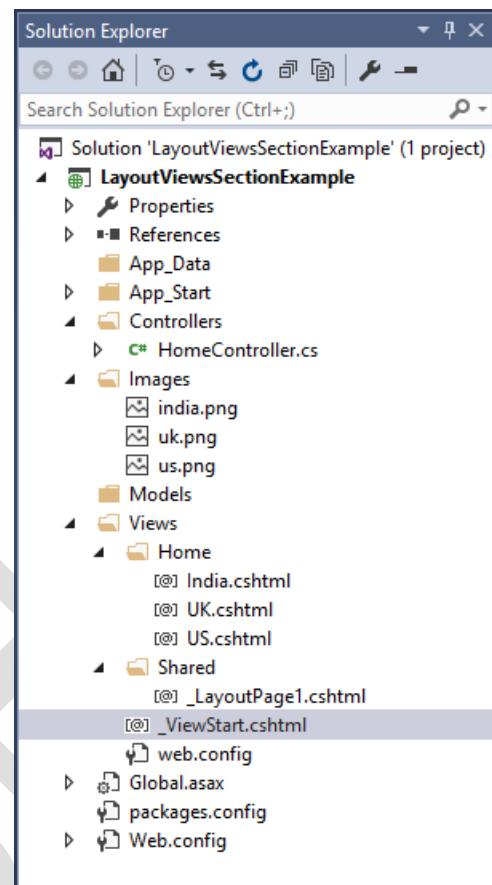
- Right click on "Views" and click on "Add" - "New Folder". Type the folder name as "Shared" and press Enter. Right click on "Views\Shared" folder and click on "Add" - "New Item". Click on "Web" - "MVC" - "MVC 5 Layout Page (Razor)". Type the file name as "\_LayoutPage1.cshtml". Click on "Add".

#### **Code for "Views\Shared\\_LayoutPage1.cshtml"**

```

<html>
    <head>
        <title>@ViewBag.Title</title>
    </head>
    <body>
        @RenderSection("Flag", required: false)
        <div>
            <h1>Header</h1>
        </div>
        <div style="background-color:skyblue">
            <a href="/Home/India">India</a>
            <a href="/Home/UK">UK</a>
            <a href="/Home/US">US</a>
        </div>
        <div>
            @RenderBody()
        </div>
        <div>
            <h1>Footer</h1>
        </div>
    </body>
</html>

```



#### **Creating India.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "India". Select the template "Empty (without model)". Check the checkbox "Use a layout page". Click on "Add".

#### **Code for "Views\Home\India.cshtml"**

```

@{
    ViewBag.Title = "India";
}
<h2>India page content here</h2>

@section Flag{
    
}

```

### Creating UK.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "UK". Select the template "Empty (without model)". Check the checkbox "Use a layout page". Click on "Add".

#### Code for "Views\Home\UK.cshtml"

```
@{  
    ViewBag.Title = "UK";  
}  
  
<h2>UK page content here</h2>  
  
@section Flag{  
      
}
```

### Creating US.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "US". Select the template "Empty (without model)". Check the checkbox "Use a layout page". Click on "Add".

#### Code for "Views\Home\US.cshtml"

```
@{  
    ViewBag.Title = "US";  
}  
  
<h2>US page content here</h2>
```

### Creating \_ViewStart.cshtml

- Right click on "Views" folder and click on "Add" - "View". Type the view name as "\_ViewStart". Select the template "Empty (without model)". Check the checkbox "Use a layout page". Click on "Add".

#### Code for "Views\Home\\_ViewStart.cshtml"

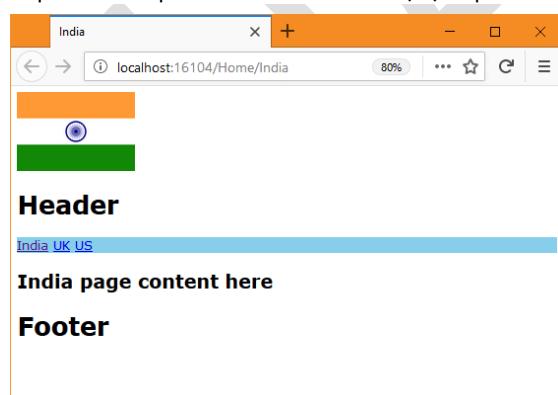
```
@{  
    Layout = "~/Views/Shared/_LayoutPage1.cshtml";  
}
```

### Running the application

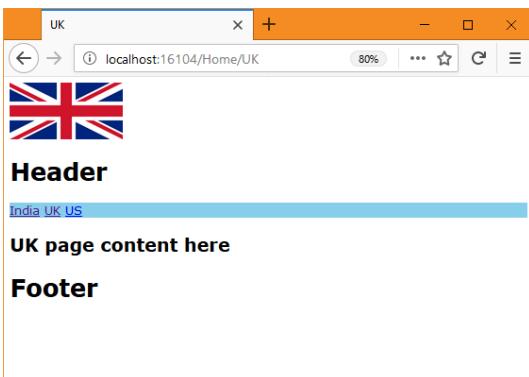
- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

#### Output:

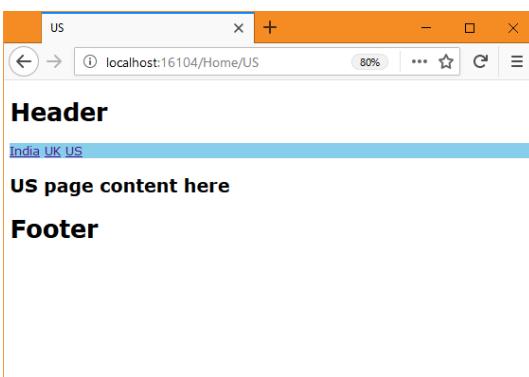
http://localhost:portnumber/Home/Index (or) http://localhost:portnumber/Home/India



http://localhost:portnumber/Home/UK



<http://localhost:portnumber/Home/US>



## PARTIAL VIEWS

### Partial Views

- Assume that, you have a specific part of the web page repeated in many web pages. Then you can make that part of the page as a separate file called "partial view" and then you can call the same in any view, anywhere. Partial views are "part of the page". Partial views are "re-usable code block". Partial views can be called in one or more views. Partial views are just like "Web User Controls" in asp.net web forms. Partial views can be present in "Views\controllername" folder or in "Views\Shared" folder.
  - If partial view is present in "Views\controllername" folder, it can be called only within the views of the same folder.
  - If partial view is present in "Views\Shared" folder, it can be called in all the views of entire project.

#### Syntax of Partial View

Views\folder\Partialviewname.cshtml

Content here

#### Syntax of Invoking Partial View

```
@[
  Html.RenderPartial("partialviewname");
]
```

## Partial Views - Example

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "PartialViewsExample". Type the location as "C:\Mvc". Type the solution name as "PartialViewsExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK. Right click on the project name (PartialViewsExample) and click on "Add" - "New Folder". Type the folder name as "Images". Copy 10 images (img1.jpg, img2.jpg, ..., img10.jpg) into "Images" in your folder. Right click on "Images" folder and click on "Paste".

### **Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### **Code for "HomeController.cs"**

```
using System;
using System.Web.Mvc;

namespace PartialViewExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

### **Creating PhotoGallery.cshtml**

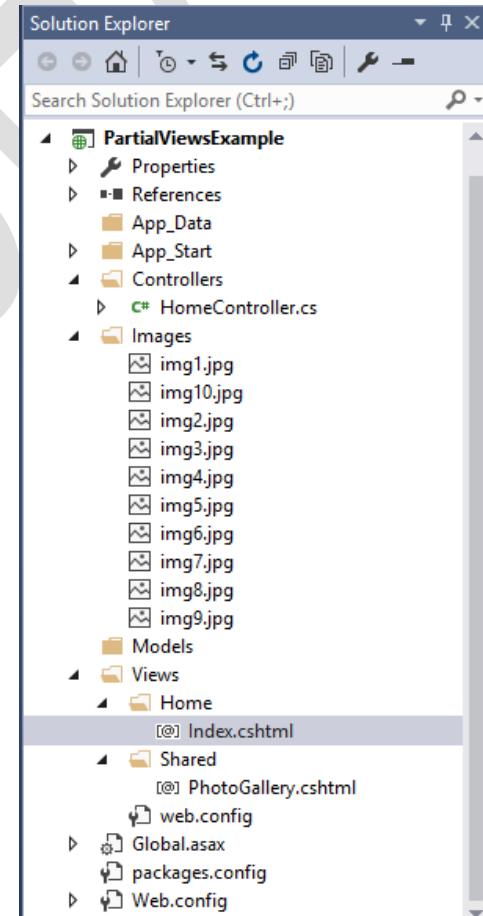
- Right click on "Shared" folder and click on "Add" - "View". Type the view name as "PhotoGallery". Select the template "Empty (without model)". Check the checkbox "Create as a partial view". Uncheck all the remaining checkboxes. Click on "Add".

### **Code for "Views\Shared\PhotoGallery.cshtml"**

```
@{
    List<string> MyImages = new List<string>()
    {
        "~/Images/img1.jpg", "~/Images/img2.jpg",
        "~/Images/img3.jpg", "~/Images/img4.jpg",
        "~/Images/img5.jpg", "~/Images/img6.jpg",
        "~/Images/img7.jpg", "~/Images/img8.jpg",
        "~/Images/img9.jpg", "~/Images/img10.jpg"
    }
}


@foreach (var item in MyImages)
    {
        
    }


```



### **Creating Index.cshtml**

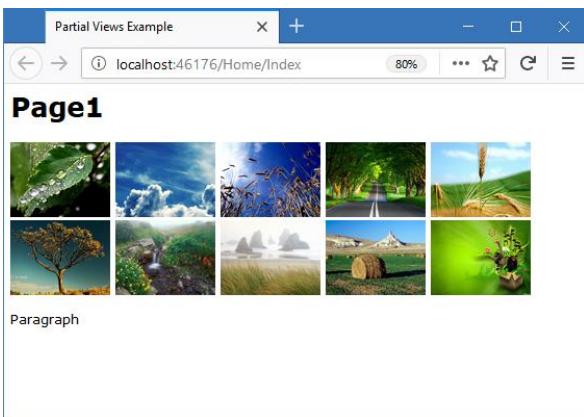
- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### **Code for "Views\Home\Index.cshtml"**

```
<html>
    <head>
        <title>Partial Views Example</title>
    </head>
    <body>
        <h1>Page1</h1>
        @{
            Html.RenderPartial("PhotoGallery");
        }
        <p>Paragraph</p>
    </body>
</html>
```

### **Running the application**

- Press "F5" to run the application.
  - Type "http://localhost:portnumber/Home/Index".
- Output:



## BUNDLING AND MINIFICATION

### What is Bundling and Minification

- By default, browser sends "one request" to get "one css file or javascript file". For example, in a web page we have used 20 files (10 javascript files and 10 css files). Then browser sends totally 20 requests to get the 20 files respectively. However, this will be a slow process, because the requests are sent in one-by-one manner. So the web page loads slowly. After lot of research, Microsoft found that reducing the no. of requests makes the page load faster. With "bundling and minification" concept in asp.net mvc, we can create a "group of files" as a "bundle". Then the browser sends one request for one bundle. For example, we have created 20 files as 4 bundles (each bundle has 5 files). The browser sends one request per one bundle. That means, the browser sends totally 4 requests but server sends all the 20 files to the browser. That means for one-request, server sends multiple files to the browser. This is possible with "bundling" concept. "Bundling" concept improves the page performances & makes the page load faster, by reducing the no. of requests.

#### Minification

- "Minification" is also called as optimization. "Minification" can be enabled by writing "EnableOptimizations=true" option in BundleConfig.cs file. Minification makes the code smaller; so that the page loads faster. When minification is enabled, asp.net mvc automatically minifies all the files in all the bundles and the minified code will be served to the browser. So minification reduces size of the code. Minification means:
  - Remove the line breaks.
  - Remove the comments.
  - Remove the un-necessary spaces.
  - Shorten the variable names.
- For the first request, it minifies all the bundled files and keep them in server's cache memory; and serve the minified files only for next request onwards.

#### Types of Bundles

- Bundles are two types:
  - ScriptBundle:** Collection of javascript files (.js files).
  - StyleBundle:** Collection of css files (.css files).

#### "Microsoft.AspNet.Web.Optimization" package

- "Microsoft.AspNet.Web.Optimization" is a NuGet package, which provides essential pre-defined classes for working with Bundling and Minification. This package can be installed by using "Manage NuGet Packages" option in Solution Explorer.

## Bundling and Minification - Example

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "BundlingAndMinificationExample". Type the location as "C:\Mvc". Type the solution name as "BundlingAndMinificationExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

#### Installing Packages

- Open Solution Explorer. Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package Microsoft.AspNet.Web.Optimization
```

- The following code will be automatically generated for "packages.config".

### **Code for "packages.config" (Automatically generated)**

```
<?xml version="1.0" encoding="utf-8"?>
<packages>
  <package id="Antlr" version="3.4.1.9004" targetFramework="net45"
userInstalled="true" />
  <package id="Microsoft.AspNet.Mvc" version="5.2.3"
targetFramework="net45" userInstalled="true" />
  <package id="Microsoft.AspNet.Razor" version="3.2.3"
targetFramework="net45" userInstalled="true" />
  <package id="Microsoft.AspNet.Web.Optimization" version="1.1.3"
targetFramework="net45" userInstalled="true" />
  <package id="Microsoft.AspNet.WebPages" version="3.2.3"
targetFramework="net45" userInstalled="true" />
  <package id="Microsoft.web.Infrastructure" version="1.0.0.0"
targetFramework="net45" userInstalled="true" />
  <package id="Newtonsoft.Json" version="5.0.4"
targetFramework="net45" userInstalled="true" />
  <package id="WebGrease" version="1.5.2" targetFramework="net45"
userInstalled="true" />
</packages>
```

### **Creating JavaScript1.js**

- Right click on the project name (BundlingAndMinificationExample) and click on "Add" - "New Folder". Type the folder name as "Scripts" and press Enter key.
- Right click on "Scripts" folder and click on "Add" - "New Item". Click on "Web" - "JavaScript File". Type the filename as "JavaScript1.js". Click on "Add".

### **Code for "Views\Scripts\JavaScript1.js"**

```
function fun1()
{
}
```

### **Creating JavaScript2.js**

- Right click on "Scripts" folder and click on "Add" - "New Item". Click on "Web" - "JavaScript File". Type the filename as "JavaScript2.js". Click on "Add".

### **Code for "Views\Scripts\JavaScript2.js"**

```
function fuz()
{
}
```

### **Creating JavaScript3.js**

- Right click on "Scripts" folder and click on "Add" - "New Item". Click on "Web" - "JavaScript File". Type the filename as "JavaScript3.js". Click on "Add".

### **Code for "Views\Scripts\JavaScript3.js"**

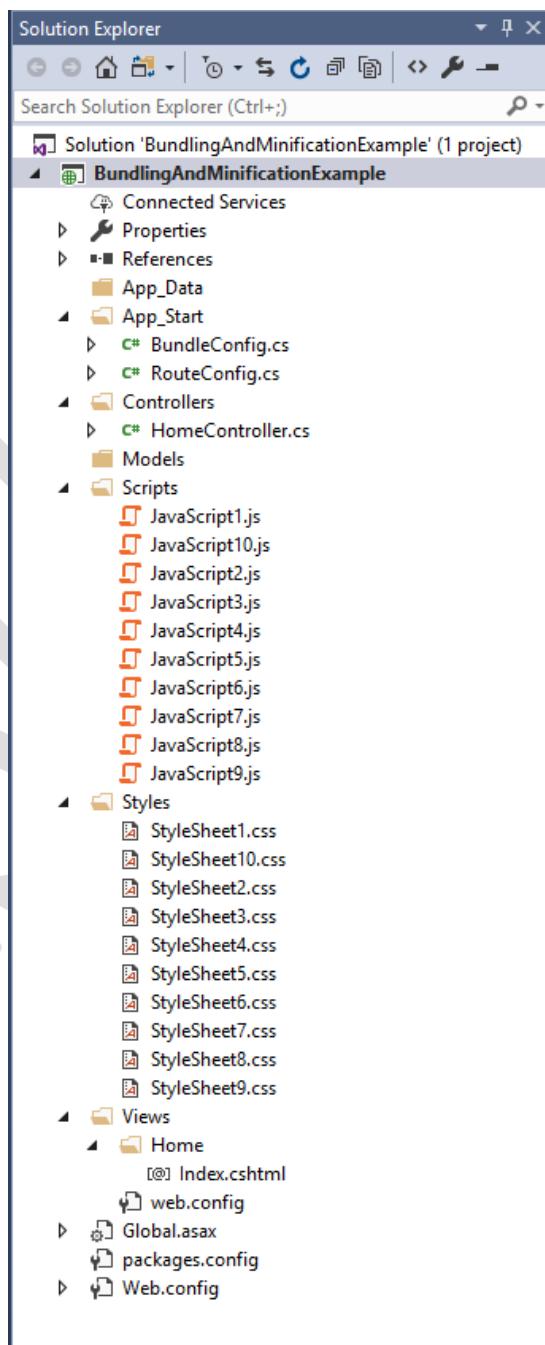
```
function fun3()
{
}
```

### **Creating JavaScript4.js**

- Right click on "Scripts" folder and click on "Add" - "New Item". Click on "Web" - "JavaScript File". Type the filename as "JavaScript4.js". Click on "Add".

### **Code for "Views\Scripts\JavaScript4.js"**

```
function fun4()
{
}
```



### Creating JavaScript5.js

- Right click on "Scripts" folder and click on "Add" - "New Item". Click on "Web" - "JavaScript File". Type the filename as "JavaScript5.js". Click on "Add".

### Code for "Views\Scripts\JavaScript5.js"

```
function fun5()  
{  
}
```

### Creating JavaScript6.js

- Right click on "Scripts" folder and click on "Add" - "New Item". Click on "Web" - "JavaScript File". Type the filename as "JavaScript6.js". Click on "Add".

### Code for "Views\Scripts\JavaScript6.js"

```
function fun6()  
{  
}
```

### Creating JavaScript7.js

- Right click on "Scripts" folder and click on "Add" - "New Item". Click on "Web" - "JavaScript File". Type the filename as "JavaScript7.js". Click on "Add".

### Code for "Views\Scripts\JavaScript7.js"

```
function fun7()  
{  
}
```

### Creating JavaScript8.js

- Right click on "Scripts" folder and click on "Add" - "New Item". Click on "Web" - "JavaScript File". Type the filename as "JavaScript8.js". Click on "Add".

### Code for "Views\Scripts\JavaScript8.js"

```
function fun8()  
{  
}
```

### Creating JavaScript9.js

- Right click on "Scripts" folder and click on "Add" - "New Item". Click on "Web" - "JavaScript File". Type the filename as "JavaScript9.js". Click on "Add".

### Code for "Views\Scripts\JavaScript9.js"

```
function fun9()  
{  
}
```

### Creating JavaScript10.js

- Right click on "Scripts" folder and click on "Add" - "New Item". Click on "Web" - "JavaScript File". Type the filename as "JavaScript10.js". Click on "Add".

### Code for "Views\Scripts\JavaScript10.js"

```
function fun10()  
{  
}
```

### Creating StyleSheet1.css

- Right click on the project name (BundlingAndMinificationExample) and click on "Add" - "New Folder". Type the folder name as "Styles" and press Enter key.
- Right click on "Styles" folder and click on "Add" - "New Item". Click on "Web" - "Style Sheet". Type the filename as "StyleSheet1.css". Click on "Add".

### Code for "Views\Styles\StyleSheet1.css"

```
body  
{  
}
```

### Creating StyleSheet2.css

- Right click on "Styles" folder and click on "Add" - "New Item". Click on "Web" - "Style Sheet". Type the filename as "StyleSheet2.css". Click on "Add".

#### Code for "Views\Styles\StyleSheet2.css"

```
body  
{  
}
```

### Creating StyleSheet3.css

- Right click on "Styles" folder and click on "Add" - "New Item". Click on "Web" - "Style Sheet". Type the filename as "StyleSheet3.css". Click on "Add".

#### Code for "Views\Styles\StyleSheet3.css"

```
body  
{  
}
```

### Creating StyleSheet4.css

- Right click on "Styles" folder and click on "Add" - "New Item". Click on "Web" - "Style Sheet". Type the filename as "StyleSheet4.css". Click on "Add".

#### Code for "Views\Styles\StyleSheet4.css"

```
body  
{  
}
```

### Creating StyleSheet5.css

- Right click on "Styles" folder and click on "Add" - "New Item". Click on "Web" - "Style Sheet". Type the filename as "StyleSheet5.css". Click on "Add".

#### Code for "Views\Styles\StyleSheet5.css"

```
body  
{  
}
```

### Creating StyleSheet6.css

- Right click on "Styles" folder and click on "Add" - "New Item". Click on "Web" - "Style Sheet". Type the filename as "StyleSheet6.css". Click on "Add".

#### Code for "Views\Styles\StyleSheet6.css"

```
body  
{  
}
```

### Creating StyleSheet7.css

- Right click on "Styles" folder and click on "Add" - "New Item". Click on "Web" - "Style Sheet". Type the filename as "StyleSheet7.css". Click on "Add".

#### Code for "Views\Styles\StyleSheet7.css"

```
body  
{  
}
```

### Creating StyleSheet8.css

- Right click on "Styles" folder and click on "Add" - "New Item". Click on "Web" - "Style Sheet". Type the filename as "StyleSheet8.css". Click on "Add".

#### Code for "Views\Styles\StyleSheet8.css"

```
body  
{  
}
```

### Creating StyleSheet9.css

- Right click on "Styles" folder and click on "Add" - "New Item". Click on "Web" - "Style Sheet". Type the filename as "StyleSheet9.css". Click on "Add".

### Code for "Views\Styles\StyleSheet9.css"

```
body  
{  
}
```

### Creating StyleSheet10.css

- Right click on "Styles" folder and click on "Add" - "New Item". Click on "Web" - "Style Sheet". Type the filename as "StyleSheet10.css". Click on "Add".

### Code for "Views\Styles\StyleSheet10.css"

```
body  
{  
}
```

### Creating BundleConfig.cs

- Right click on "App\_Start" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the filename as "BundleConfig.cs". Click on "Add".

### Code for "App\_Start\BundleConfig.cs"

```
using System;  
using System.Web.Optimization;  
  
namespace BundlingAndMinificationExample  
{  
    public class BundleConfig  
    {  
        public static void RegisterBundles(BundleCollection bundles)  
        {  
            bundles.Add(new ScriptBundle("~/bundles/scriptbundle1").Include(  
                "~/Scripts/JavaScript1.js", "~/Scripts/JavaScript2.js", "~/Scripts/JavaScript3.js", "~/Scripts/JavaScript4.js",  
                "~/Scripts/JavaScript5.js"));  
  
            bundles.Add(new ScriptBundle("~/bundles/scriptbundle2").Include(  
                "~/Scripts/JavaScript6.js", "~/Scripts/JavaScript7.js", "~/Scripts/JavaScript8.js", "~/Scripts/JavaScript9.js",  
                "~/Scripts/JavaScript10.js"));  
  
            bundles.Add(new StyleBundle("~/bundles/stylebundle1").Include(  
                "~/Styles/StyleSheet1.css", "~/Styles/StyleSheet2.css", "~/Styles/StyleSheet3.css", "~/Styles/StyleSheet4.css",  
                "~/Styles/StyleSheet5.css"));  
  
            bundles.Add(new StyleBundle("~/bundles/stylebundle2").Include(  
                "~/Styles/StyleSheet6.css", "~/Styles/StyleSheet7.css", "~/Styles/StyleSheet8.css", "~/Styles/StyleSheet9.css",  
                "~/Styles/StyleSheet10.css"));  
  
            BundleTable.EnableOptimizations = true;  
        }  
    }  
}
```

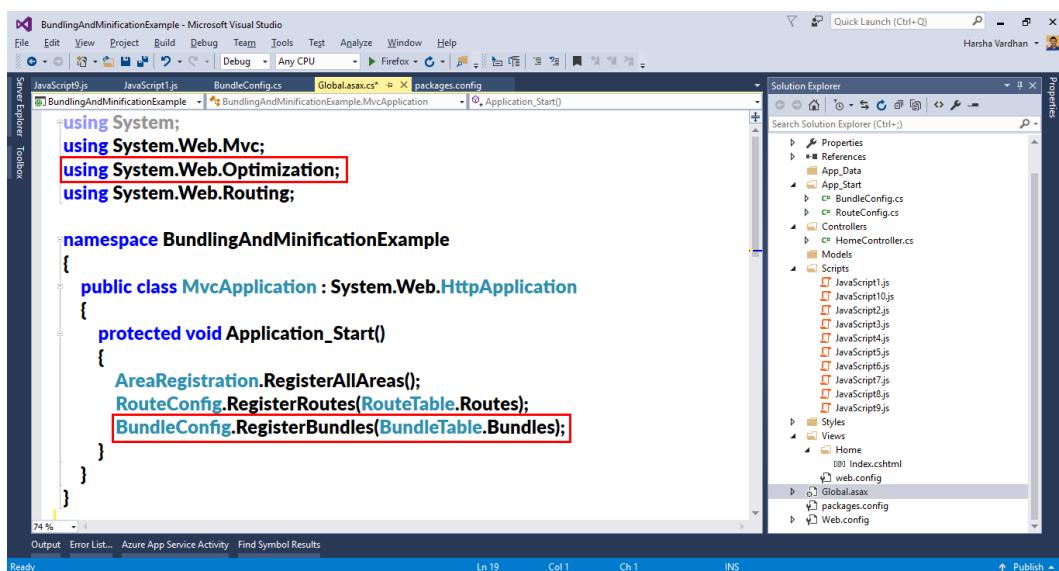
### Editing Global.asax

- Open "Global.asax" file.

### Add the following statements to "Global.asax"

```
using System.Web.Optimization;  
--- and --
```

```
BundleConfig.RegisterBundles(BundleTable.Bundles);
```



### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```

using System;
using System.Web.Mvc;

namespace BundlingAndMinificationExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}

```

### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### Code for "Views\Home\Index.cshtml"

```

@using System.Web.Optimization
<html>
<head>
    <title>Bundling and Minification</title>
    @Scripts.Render("~/bundles/scriptbundle1")
    @Scripts.Render("~/bundles/scriptbundle2")
    @Styles.Render("~/bundles/stylebundle1")
    @Styles.Render("~/bundles/stylebundle2")
</head>
<body>
    <h1>Bundling and Minification</h1>
</body>
</html>

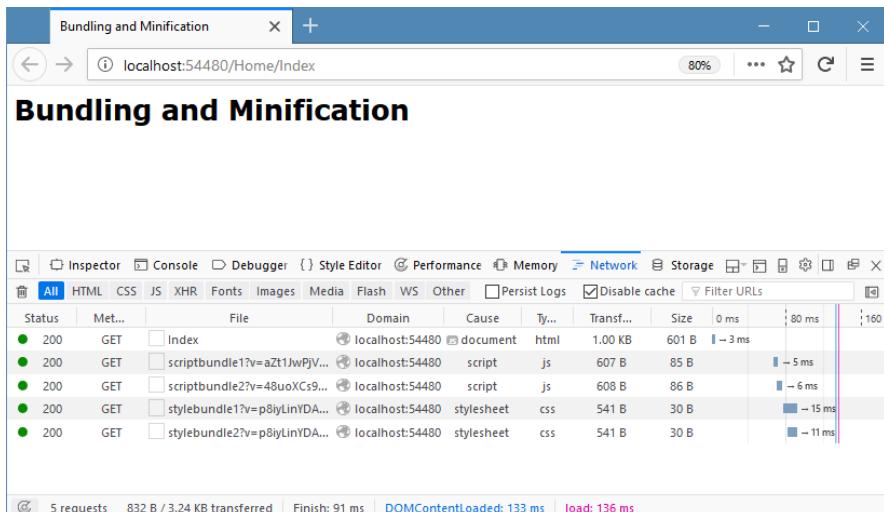
```

### Running the application

- Press "F5" to run the application.

- Type "http://localhost:portnumber/Home/Index".

Output:



- In the browser, right click on the web page and click on "Inspect Element". Click on "Network". Press "F5" to refresh. Then you can notice 4 requests are sent for 4 bundles respectively.

## FILTERS

- Filters are attributes that can be applied to an action method or a controller. Filters define behavior of action method or controller. MVC provides the following pre-defined filters:

1. [ChildActionOnly]
2. [OutputCache]
3. [ActionName]
4. [NonAction]
5. [HandleError]
6. [Route]
7. [HttpGet]
8. [HttpPost]
9. [ValidateAntiForgeryToken]
10. [Authorize]

### 1. [ChildActionOnly]

- This filter is used to specify that the action method can be called with a child request only, by using `Html.RenderAction` method. If you send a request to the action method directly from the browser, it throws an exception.

### 2. [OutputCache]

- This filter is used to enable output cache. When you enable output cache for an action method, its code and corresponding view will be executed only for the first request and asp.net mvc will store the output of the view in the server's cache memory and delivers the same as response to the browser for all the subsequent requests. So the action method and view need not be executed for second request onwards. So "Output cache" concept reduces burden on the server.

### 3. [ActionName]

- This filter is used define an alias name for the action method. So we have to call the action method with the alias name, not with its original name.

### 4. [NonAction]

- This filter is used specify that this method is a normal method only; not an action method. So then the method will not accept any request from the browser, but can be called locally within the project.

### 5. [HandleError]

- This filter is used to enable custom error page. So when an exception is raised while executing the controller or view, it automatically opens "Views\Shared\Error.cshtml" page.

### 6. [Route]

- This filter is used to specify "attribute based routing". Using attribute based routing, we can specify the url syntax for a specific action method, instead of specifying it in the RouteConfig.cs.

### 7. [HttpGet]

- This filter is used to specify that the current action method should accept "Http GET" requests only.

### 8. [HttpPost]

- This filter is used to specify that the current action method should accept "Http POST" requests only.

### 9. [ValidateAntiForgeryToken]

- This filter is used to specify that the current action method should not accept cross-domain requests.

### 10. [Authorize]

- This filter is used to specify that the current user must be login to accept the request. If the user has not logged-in, the request will be rejected.

## OutputCache

### OutputCache

- It is used to execute the action method & view only for the first request, and give the same result as response for all the subsequent requests within the specified time limit. Server internally stores the first result (html output) of the view in the cache memory (temporary memory).
- Advantage of Output Cache:** Reduce burden on server; because loading the data from cache memory is faster than executing the code.
- Syntax:** [OutputCache(Duration = seconds)]

## Output Caching - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "OutputCachingExample". Type the location as "C:\Mvc". Type the solution name as "OutputCachingExample". Click on OK. Select "Empty". Check the checkbox 'MVC'. Uncheck the checkbox 'Enable Docker support'. Uncheck the checkbox "Add unit tests". Click on OK.

### Creating HomeController.cs

- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;

namespace OutputCachingExample.Controllers
{
    public class HomeController : Controller
    {
        [OutputCache(Duration = 60)]
        public ActionResult Index()
        {
            ViewBag.CurrentTime = DateTime.Now.ToString();
            return View();
        }
    }
}
```

}

### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

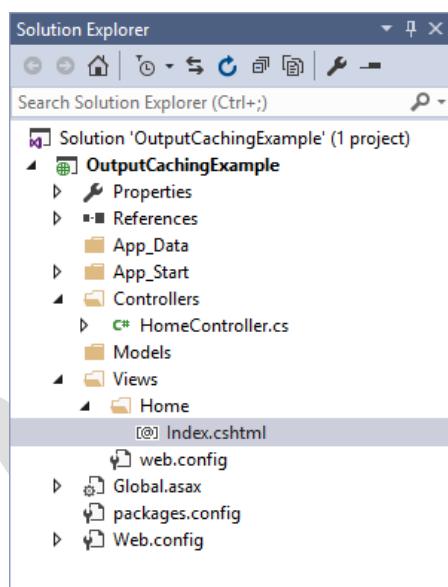
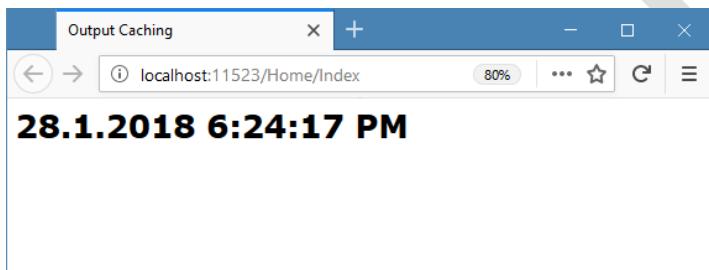
### Code for "Views\Home\Index.cshtml"

```
<html>
  <head>
    <title>Output Caching</title>
  </head>
  <body>
    <h1>@ViewBag.CurrentTime</h1>
  </body>
</html>
```

### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



- Note:** Keep on refreshing the web page repeatedly; then you can notice that it shows fixed time for 1 minute.

## ActionName

- By default, "method name" is taken as "action name" for action methods. "ActionName" is an attribute, which is used to specify the "action name", if the "method name" and "action name" are different.
- Syntax:** [ActionName("action name here")]

## ActionName - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "ActionNameExample". Type the location as "C:\Mvc". Type the solution name as "ActionNameExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK. Open Solution Explorer.

### Creating HomeController.cs

- Open Solution Explorer.
- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;

namespace ActionNameExample.Controllers
{
  public class HomeController : Controller
  {
    [ActionName("Index")]
  }
}
```

```
public ActionResult Method1()
{
    return View();
}
```

### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

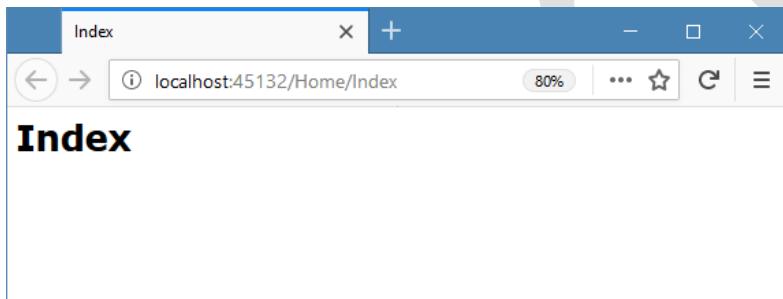
### Code for "Views\Home\Index.cshtml"

```
<html>
<head>
<title>Index</title>
</head>
<body>
<h1>Index</h1>
</body>
</html>
```

### Running the application

- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Index".

Output:



## NonAction

### NonAction

- By default, all methods present within the controller class are treated as "action methods". The "NonAction" is an attribute, which is used to give exemption for a method, not to be treated as "action method".
- Syntax: [NonAction]

## NonAction - Example

### Creating Project

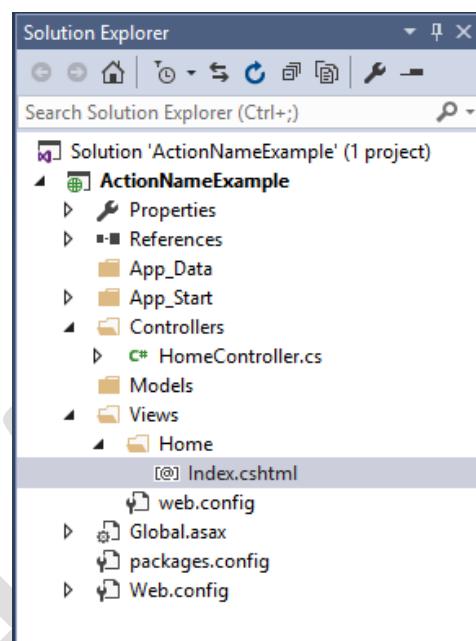
- Open Visual Studio 2017.
- Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "NonActionExample". Type the location as "C:\Mvc". Type the solution name as "NonActionExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating HomeController.cs

- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;
```



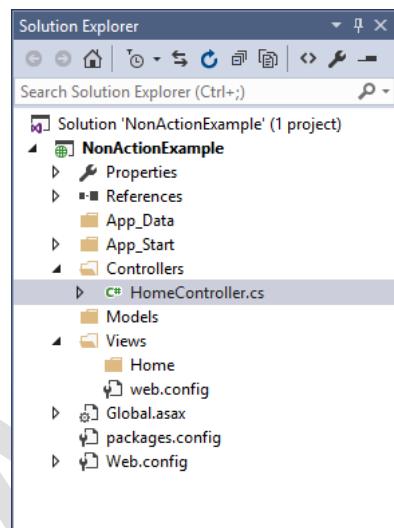
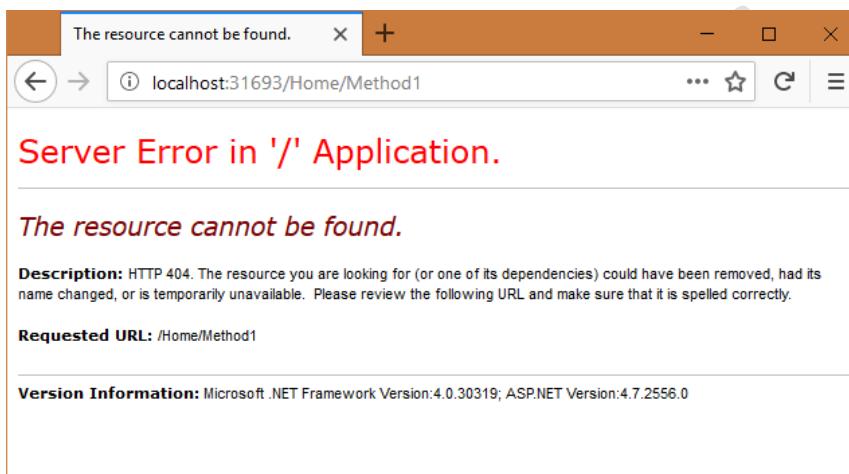
```
namespace NonActionExample.Controllers
{
    public class HomeController : Controller
    {
        [NonAction]
        public void Method1()
        {
            //some code here
        }
    }
}
```

### Running the application

- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Method1".

Output:

- It shows error as follows:



## ChildActionOnly

- Assume that, from a view, you want to call an action method and then call a partial view and display its result in the existing view, then you have to use "Child Actions".
- Partial view contain "re-usable UI". The "Child Actions" concept is used to supply some dynamic data to the partial views, which can be displayed in the partial view. Child actions are action methods that returns a partial view. To call the child action, use "Html.RenderAction" method in the view. When you call a child action, browser sends a new child request to the child action method and then it goes to the partial view and displays the output of the partial view in the main view.

### Methods to call a Partial View

- The following methods are used to call a partial view:

#### **Html.RenderPartial()**

- Calls the partial view directly. Useful when you don't want to supply any dynamic data to the partial view and display some fixed content.
- **Syntax:** @ { Html.RenderPartial("partial view file name"); }

#### **Html.RenderAction()**

- Calls the action method first; and then it goes to the partial view. Useful when you want to supply some dynamic data to the partial view, and display the dynamic data in the partial view.
- **Syntax:** @ { Html.RenderAction("action", "controller"); }
- To make the action method callable only through RenderAction() method (not directly from the browser's address bar), use [ChildActionOnly] attribute.

## ChildActionOnly - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name as "ChildActionsExample". Type the location as "C:\Mvc". Type the solution name as "ChildActionsExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK. Open Solution Explorer. Right click on the project name (ChildActionsExample) and click on "Add" - "New Folder". Type the folder name as "Images". Copy 10 images (img1.jpg, img2.jpg, ..., img10.jpg) into "Images" in your folder. Right click on "Images" folder and click on "Paste".

### Creating HomeController.cs

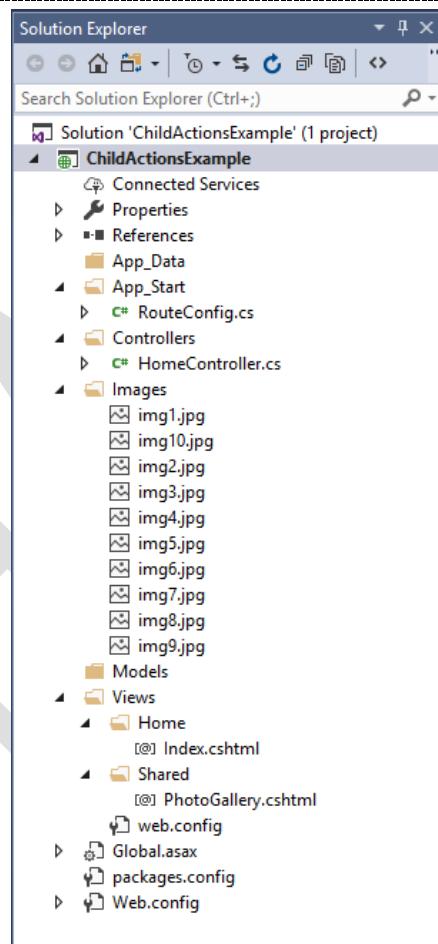
- Open Solution Explorer. Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

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

namespace ChildActionsExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        [ChildActionOnly]
        public ActionResult PhotoGallery()
        {
            ViewBag.MyImages = new List<string>()
            {
                "~/Images/img1.jpg", "~/Images/img2.jpg",
                "~/Images/img3.jpg", "~/Images/img4.jpg",
                "~/Images/img5.jpg", "~/Images/img6.jpg",
                "~/Images/img7.jpg", "~/Images/img8.jpg",
                "~/Images/img9.jpg", "~/Images/img10.jpg"
            };
            return PartialView();
        }
    }
}
```



### Creating PhotoGallery.cshtml

- Right click on "Views" and click on "Add" - "New Folder". Type the folder name as "Shared" and press Enter. Right click on "Shared" folder and click on "Add" - "View". Type the view name as "PhotoGallery". Select the template "Empty (without model)". Check the checkbox "Create as a partial view". Uncheck all the remaining checkboxes. Click on "Add".

### Code for "Views\Shared\PhotoGallery.cshtml"

```
<html>
    <head>
        <title>Child Actions</title>
    </head>
    <body>
        <h1>Header</h1>
        @{
            Html.RenderAction("PhotoGallery", "Home");
        }
        <p>Footer</p>
    </body>
</html>
```

### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

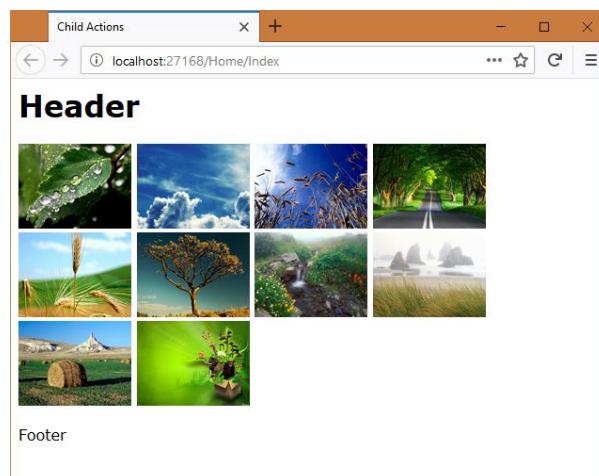
### Code for "Views\Home\Index.cshtml"

```
<html>
<head>
    <title>Child Actions</title>
</head>
<body>
    <h1>Header</h1>
    @{
        Html.RenderAction("PhotoGallery", "Home");
    }
    <p>Footer</p>
</body>
</html>
```

### Running the application

- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Index".

#### Output:



## CUSTOM FILTERS

### Introduction to Custom Filters

- "Custom Filters" are the filters created by the developer. Filters are used to create "additional functionality to action methods". Filters execute at a specific stage while the request is being processed in MVC. Filters contain two types of methods:
  - Before Methods:** Execute before action methods execute.
  - After Methods:** Execute after action methods execute.

### Types of Filters

- MVC 5 supports the following types of filters:
  - Authorization Filter:**
    - Used to implement authorization. That means, we can check whether the user is having permission to access the action method or not.
    - Implemented using "IAuthorizationFilter" interface. This interface contains one method:
      - OnAuthorization:** Executes while checking for authentication, before executing the action method.
  - Action Filter**
    - Used to implement custom logic before and after execution of action method.

- Implemented using "IActionFilter" interface. This interface contains two methods:
  - **OnActionExecuting:** Executes before executing the action method.
  - **OnActionExecuted:** Executes before executing the action method.

### 3. Result Filter

- Used to implement custom logic before and after execution of result.
- Implemented using "IResultFilter" interface. This interface contains two methods:
  - **OnResultExecuting:** Executes before executing the result.
  - **OnResultExecuted:** Executes before executing the result.

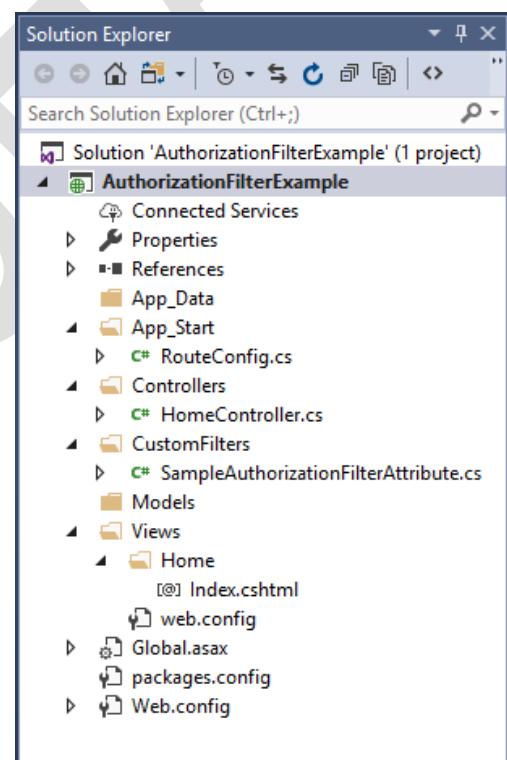
### 4. Exception Filter

- Used to implement custom logic when exception is raised while executing action method.
- Implemented using "IExceptionFilter" interface. This interface contains one method:
  - **OnException:** Executes when exception is raised while executing the action method.

## IAuthorizationFilter - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "AuthorizationFilterExample". Type the location as "C:\Mvc". Type the solution name as "AuthorizationFilterExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.



### Creating SampleAuthorizationFilterAttribute.cs

- Open Solution Explorer. Right click on the project (AuthorizationFilterExample) and click on "Add" - "New Folder". Type the folder name as "CustomFilters" and press Enter. Right click on "CustomFilters" folder and click on "Add" - "Code" - "Class". Type the filename as "SampleAuthorizationFilterAttribute". Click on "Add".

### Code for "CustomFilters\SampleAuthorizationFilterAttribute.cs"

```
using System;
using System.Web.Mvc;

namespace AuthorizationFilterExample.CustomFilters
{
    public class SampleAuthorizationFilterAttribute : FilterAttribute,
        IAuthorizationFilter
    {
        public void OnAuthorization(AuthorizationContext filterContext)
        {
            filterContext.Controller.ViewBag.Message =
                "IAuthorizationFilter.OnAuthorization";
        }
    }
}
```

### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;
using AuthorizationFilterExample.CustomFilters;

namespace AuthorizationFilterExample.Controllers
{
```

```
public class HomeController : Controller
{
    [SampleAuthorizationFilter]
    public ActionResult Index()
    {
        ViewBag.Message += " Index";
        return View();
    }
}
```

### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

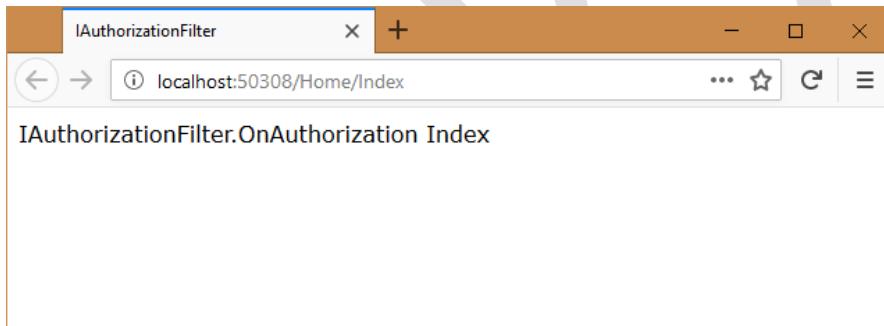
### Code for "Views\Home\Index.cshtml"

```
<html>
<head>
    <title>IAuthorizationFilter</title>
</head>
<body>
    @ViewBag.Message
</body>
</html>
```

### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



## IActionFilter - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "IActionFilterExample". Type the location as "C:\Mvc". Type the solution name as "IActionFilterExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests".
- Click on OK.

### Creating SampleActionFilterAttribute.cs

- Open Solution Explorer. Right click on the project (IActionFilterExample) and click on "Add" - "New Folder". Type the folder name as "CustomFilters" and press Enter. Right click on "CustomFilters" folder and click on "Add" - "Code" - "Class". Type the filename as "SampleActionFilterAttribute". Click on "Add".

### Code for "CustomFilters\SampleActionFilterAttribute.cs"

```
using System;
```

```

using System.Web.Mvc;

namespace IActionFilterExample.CustomFilters
{
    public class SampleActionFilterAttribute : FilterAttribute, IActionFilter
    {
        public void OnActionExecuting(ActionExecutingContext filterContext)
        {
            filterContext.Controller.ViewBag.Message =
"IACTIONFILTER.OnActionExecuting";
        }

        public void OnActionExecuted(ActionExecutedContext filterContext)
        {
            filterContext.Controller.ViewBag.Message += "
IACTIONFILTER.OnActionExecuted";
        }
    }
}

```

### **Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### **Code for "HomeController.cs"**

```

using System;
using System.Web.Mvc;
using IActionFilterExample.CustomFilters;

namespace IActionFilterExample.Controllers
{
    public class HomeController : Controller
    {
        [SampleActionFilter]
        public ActionResult Index()
        {
            ViewBag.Message += " Index";
            return View();
        }
    }
}

```

### **Creating Index.cshtml**

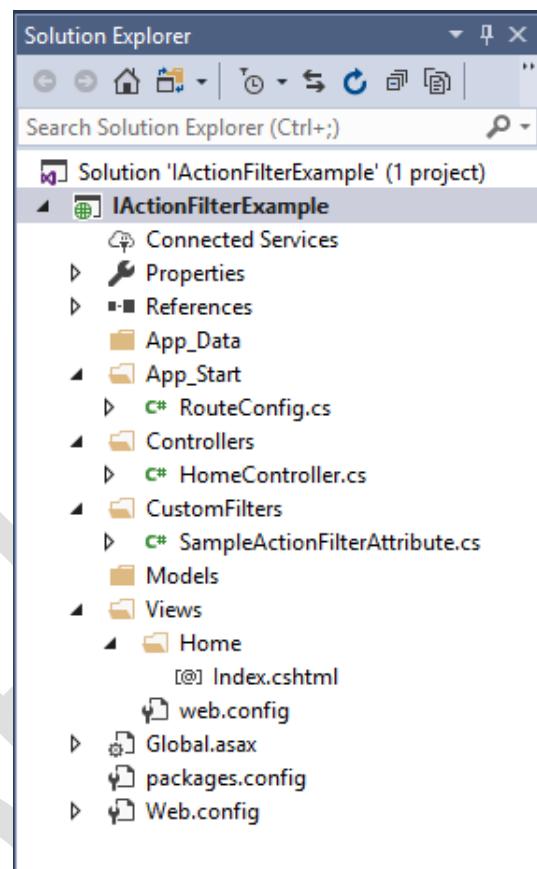
- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### **Code for "Views\Home\Index.cshtml"**

```

<html>
<head>
<title>IACTIONFILTER</title>
</head>
<body>
    @ViewBag.Message
</body>
</html>

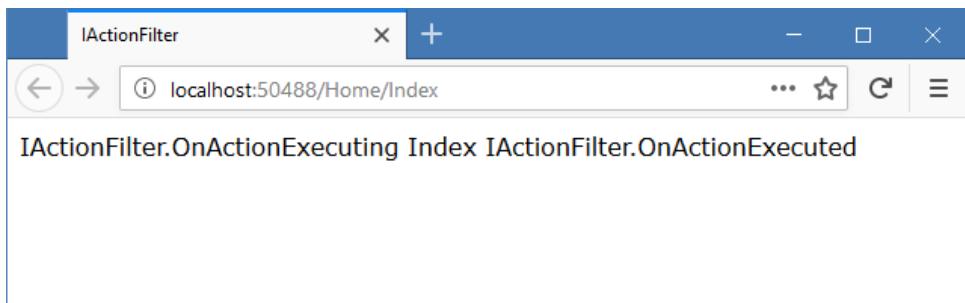
```



### **Running the application**

- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Index".

Output:



## IResultFilter - Example

**Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name as "IResultFilterExample". Type the location as "C:\Mvc". Type the solution name as "IResultFilterExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

**Creating SampleResultFilterAttribute.cs**

- Open Solution Explorer. Right click on the project (IResultFilterExample) and click on "Add" - "New Folder". Type the folder name as "CustomFilters" and press Enter. Right click on "CustomFilters" folder and click on "Add" - "Code" - "Class". Type the filename as "SampleResultFilterAttribute". Click on "Add".

**Code for "CustomFilters\SampleResultFilterAttribute.cs"**

```

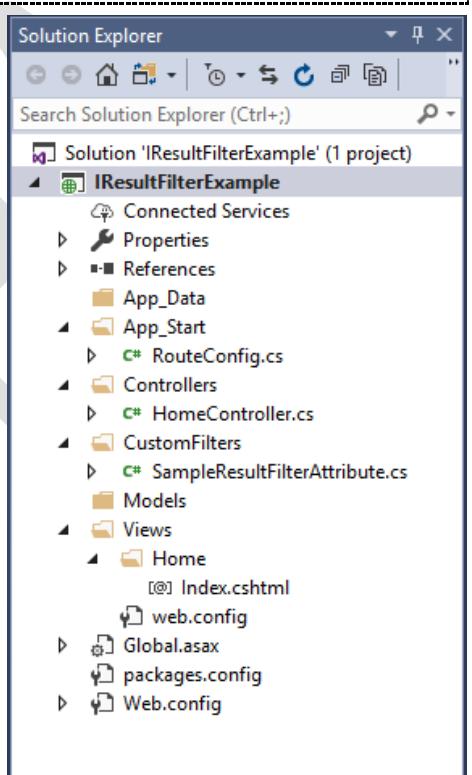
using System;
using System.Web.Mvc;

namespace IResultFilterExample.CustomFilters
{
    public class SampleResultFilterAttribute : FilterAttribute, IResultFilter
    {
        public void OnResultExecuting(ResultExecutingContext filterContext)
        {
            filterContext.Controller.ViewBag.Message += "IResultFilter.OnResultExecuting";
        }

        public void OnResultExecuted(ResultExecutedContext filterContext)
        {
            filterContext.Controller.ViewBag.Message += "IResultFilter.OnResultExecuted";
        }
    }
}

```

**Solution Explorer**



## Creating HomeController.cs

```

        ViewBag.Message += " Index";
        return View();
    }
}
}

```

#### **Creating Index.cshtml**

- Right click on “Views\Home” folder and click on “Add” - “View”. Type the view name as “Index”. Select the template “Empty (without model)”. Uncheck all the checkboxes. Click on “Add”.

#### **Code for “Views\Home\Index.cshtml”**

```

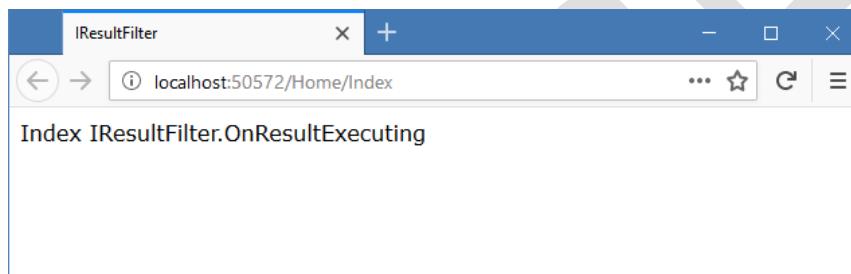
<html>
<head>
    <title>IResultFilter</title>
</head>
<body>
    @ViewBag.Message
</body>
</html>

```

#### **Running the application**

- Press “F5” to run the application. Type “<http://localhost:50572/Home/Index>”.

Output:



## IExceptionFilter - Example

#### **Creating Project**

- Open Visual Studio 2017. Go to “File” - “New” - “Project”. Select “.NET Framework 4.7”. Select “Visual C#”, Select “ASP.NET Web Application (.NET Framework)”. Type the project name “IExceptionFilterExample”. Type the location as “C:\Mvc”. Type the solution name as “[IExceptionFilterExample]”. Click on OK. Select “Empty”. Check the checkbox “MVC”. Uncheck the checkbox ‘Enable Docker support’. Uncheck the checkbox “Add unit tests”. Click on OK.

#### **Creating SampleExceptionFilterAttribute.cs**

- Open Solution Explorer. Right click on the project (IExceptionFilterExample) and click on “Add” - “New Folder”. Type the folder name as “CustomFilters” and press Enter. Right click on “CustomFilters” folder and click on “Add” - “Code” - “Class”. Type the filename as “SampleExceptionFilterAttribute”. Click on “Add”.

#### **Code for “CustomFilters\SampleExceptionFilterAttribute.cs”**

```

using System;
using System.Web.Mvc;

namespace IExceptionFilterExample.CustomFilters
{
    public class SampleExceptionFilterAttribute : FilterAttribute, IExceptionFilter
    {
        public void OnException(ExceptionContext filterContext)
        {
            filterContext.HttpContext.Application["Message"] = " IExceptionFilter.OnException";
            filterContext.HttpContext.Response.Redirect("~/Home/Error");
        }
    }
}

```

}

### **Creating HomeController.cs**

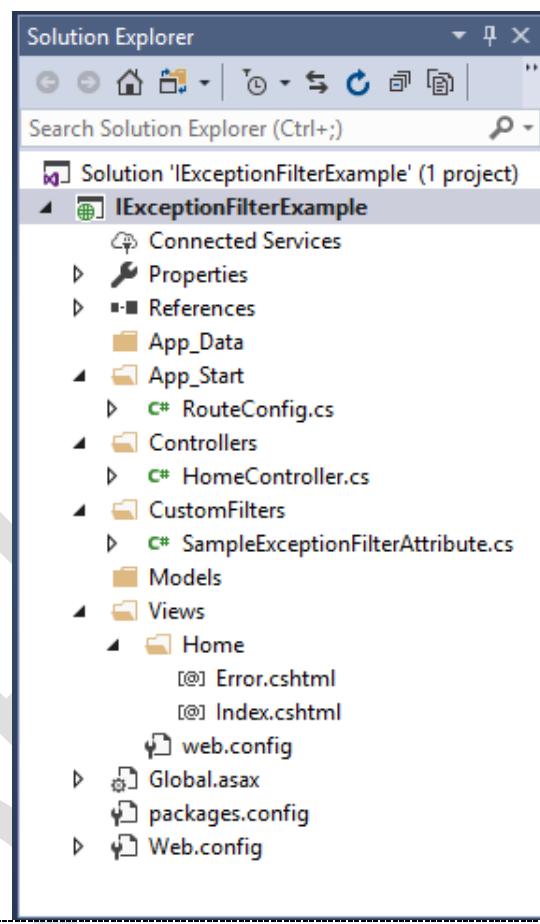
- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### **Code for "HomeController.cs"**

```
using System;
using System.Web.Mvc;
using IExceptionFilterExample.CustomFilters;

namespace IExceptionFilterExample.Controllers
{
    public class HomeController : Controller
    {
        [SampleExceptionFilter]
        public ActionResult Index()
        {
            int a = 10;
            int b = 0;
            int c = a / b; //exception raises here
            return View();
        }

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



### **Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### **Code for "Views\Home\Index.cshtml"**

```
<html>
<head>
    <title>Index</title>
</head>
<body>
    <h1>Index</h1>
</body>
</html>
```

### **Creating Error.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Error". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

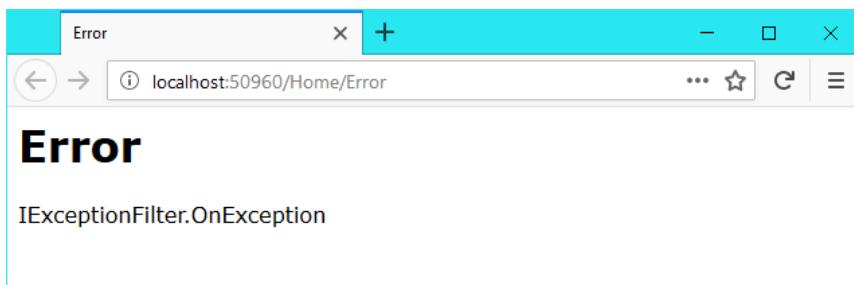
### **Code for "Views\Home\Error.cshtml"**

```
<html>
<head>
    <title>Error</title>
</head>
<body>
    <h1>Error</h1>
    @ViewContext.HttpContext.Application["Message"]
</body>
</html>
```

### **Running the application**

- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Index".

Output:



## Global Filters

### Global Filters

- If you want to apply an action filter for all the action methods of all the controllers in the entire project, use "Global action filters". Global Action Filters are written in "FilterConfig.cs". Global Action Filters are automatically applied for all the action methods of all the controllers in the current project.

### Syntax of FilterConfig.cs

```
using System;
using System.Web.Mvc;

namespace namespacehere
{
    public class FilterConfig
    {
        public static void RegisterGlobalFilters(GlobalFilterCollection filters)
        {
            filters.Add(new YourAttributename());
        }
    }
}
```

## Global Action Filters - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "GlobalFiltersExample". Type the location as "C:\Mvc". Type the solution name as "GlobalFiltersExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating SampleAuthorizationFilterAttribute.cs

- Open Solution Explorer. Right click on the project (AuthorizationFilterExample) and click on "Add" - "New Folder". Type the folder name as "CustomFilters" and press Enter. Right click on "CustomFilters" folder and click on "Add" - "Code" - "Class". Type the filename as "SampleAuthorizationFilterAttribute". Click on "Add".

### Code for "CustomFilters\SampleAuthorizationFilterAttribute.cs"

```
using System;
using System.Web.Mvc;
```

```
namespace GlobalFiltersExample.CustomFilters
```

```
{
    public class SampleAuthorizationFilterAttribute : FilterAttribute,
    IAuthorizationFilter
    {
        public void OnAuthorization(AuthorizationContext filterContext)
        {
            filterContext.Controller.ViewBag.Message += " "
            IAuthorizationFilter.OnAuthorization";
        }
    }
}
```

**Creating SampleActionFilterAttribute.cs**

- Right click on "CustomFilters" folder and click on "Add" - "Code" - "Class". Type the filename as "SampleActionFilterAttribute". Click on "Add".

**Code for "CustomFilters\SampleActionFilterAttribute.cs"**

```
using System;
using System.Web.Mvc;

namespace GlobalFiltersExample.CustomFilters
{
    public class SampleActionFilterAttribute : FilterAttribute, IActionFilter
    {
        public void OnActionExecuting(ActionExecutingContext filterContext)
        {
            filterContext.Controller.ViewBag.Message += " "
            IActionFilter.OnActionExecuting";
        }

        public void OnActionExecuted(ActionExecutedContext filterContext)
        {
            filterContext.Controller.ViewBag.Message += " "
            IActionFilter.OnActionExecuted";
        }
    }
}
```

**Creating SampleResultFilterAttribute.cs**

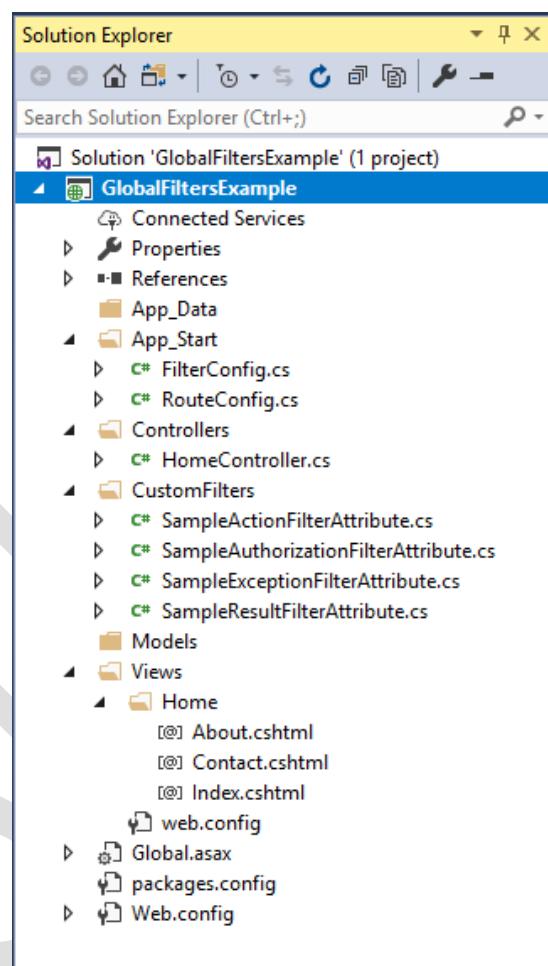
- Right click on "CustomFilters" folder and click on "Add" - "Code" - "Class". Type the filename as "SampleResultFilterAttribute". Click on "Add".

**Code for "CustomFilters\SampleResultFilterAttribute.cs"**

```
using System;
using System.Web.Mvc;

namespace GlobalFiltersExample.CustomFilters
{
    public class SampleResultFilterAttribute : FilterAttribute, IResultFilter
    {
        public void OnResultExecuting(ResultExecutingContext filterContext)
        {
            filterContext.Controller.ViewBag.Message += " IResultFilter.OnResultExecuting";
        }

        public void OnResultExecuted(ResultExecutedContext filterContext)
        {
            filterContext.Controller.ViewBag.Message += " IResultFilter.OnResultExecuted";
        }
    }
}
```



### **Creating SampleExceptionFilterAttribute.cs**

- Right click on "CustomFilters" folder and click on "Add" - "Code" - "Class". Type the filename as "SampleExceptionFilterAttribute". Click on "Add".

### **Code for "CustomFilters\SampleExceptionFilterAttribute.cs"**

```
using System;
using System.Web.Mvc;

namespace GlobalFiltersExample.CustomFilters
{
    public class SampleExceptionFilterAttribute : FilterAttribute, IExceptionFilter
    {
        public void OnException(ExceptionContext filterContext)
        {
            filterContext.Controller.ViewBag.Message += " IExceptionFilter.OnException";
        }
    }
}
```

### **Creating FilterConfig.cs**

- Right click on "App\_Start" folder and click on "Add" - "Code" - "Class". Type the filename as "FilterConfig.cs". Click on "Add".

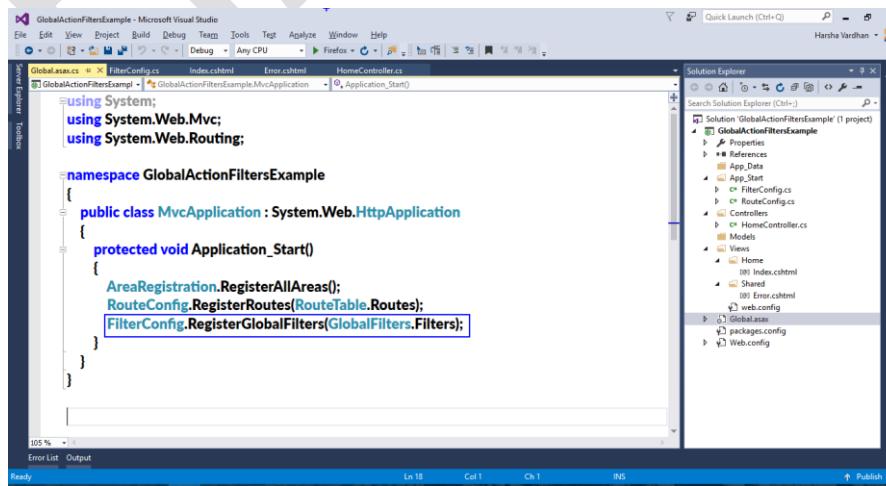
### **Code for "App\_Start\FilterConfig.cs"**

```
using System;
using System.Web.Mvc;
using GlobalFiltersExample.CustomFilters;

namespace GlobalFiltersExample
{
    public class FilterConfig
    {
        public static void RegisterGlobalFilters(GlobalFilterCollection filters)
        {
            filters.Add(new SampleAuthorizationFilterAttribute());
            filters.Add(new SampleActionFilterAttribute());
            filters.Add(new SampleResultFilterAttribute());
            filters.Add(new SampleExceptionFilterAttribute());
        }
    }
}
```

### **Adding Code to "Global.asax"**

FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);



**[ Creating HomeController.cs ]**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

**Code for "HomeController.cs"**

```
using System;
using System.Web.Mvc;
using GlobalFiltersExample.CustomFilters;

namespace GlobalFiltersExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            ViewBag.Message += " Index";
            return View();
        }

        public ActionResult About()
        {
            ViewBag.Message += " About";
            return View();
        }

        public ActionResult Contact()
        {
            ViewBag.Message += " Contact";
            return View();
        }
    }
}
```

**[ Creating Index.cshtml ]**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Index.cshtml"**

```
<html>
<head>
    <title>Global Filters</title>
</head>
<body>
    <div>
        <a href="/Home/Index">Index</a>
        <a href="/Home/About">About</a>
        <a href="/Home/Contact">Contact</a>
    </div>
    <h1>Index</h1>
    @ViewBag.Message
</body>
</html>
```

**[ Creating About.cshtml ]**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "About". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\About.cshtml"**

```
<html>
<head>
    <title>Global Filters</title>
</head>
<body>
```

```

<div>
  <a href="/Home/Index">Index</a>
  <a href="/Home/About">About</a>
  <a href="/Home/Contact">Contact</a>
</div>
<h1>About</h1>
@ViewBag.Message
</body>
</html>

```

#### Creating Contact.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Contact". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### Code for "Views\Home>Contact.cshtml"

```

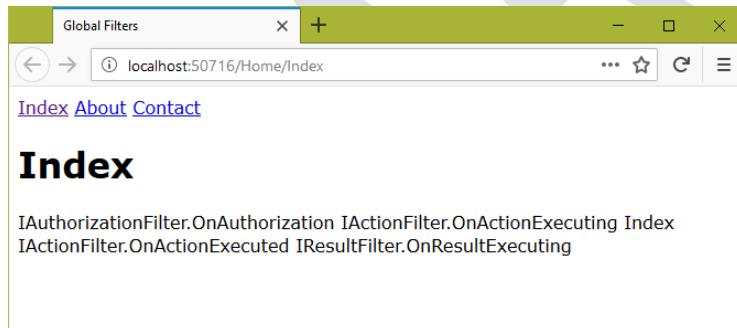
<html>
<head>
  <title>Global Filters</title>
</head>
<body>
<div>
  <a href="/Home/Index">Index</a>
  <a href="/Home/About">About</a>
  <a href="/Home/Contact">Contact</a>
</div>
<h1>Contact</h1>
@ViewBag.Message
</body>
</html>

```

#### Running the application

- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Index".

Output:



## EXCEPTION HANDLING IN MVC

- We can handle exception handling in six ways in MVC:
  - Try Catch
  - OnException
  - IExceptionFilter
  - HandleError
  - HTTP Errors
  - Application\_Error

### Try Catch

- This is very standard way of handling exceptions in any type of application.
- Advantage:** Easy to use.

- **Disadvantage:** We need to write try-catch syntax in every action method manually; and we are not getting the real advantages of MVC in exception handling.
- **Syntax:**

```

try
{
    code here
}
catch
{
    return View("viewname");
}

```

### TryCatch - Example

#### **Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "TryCatchExample". Type the location as "C:\Mvc". Type the solution name as "TryCatchExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

#### **Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

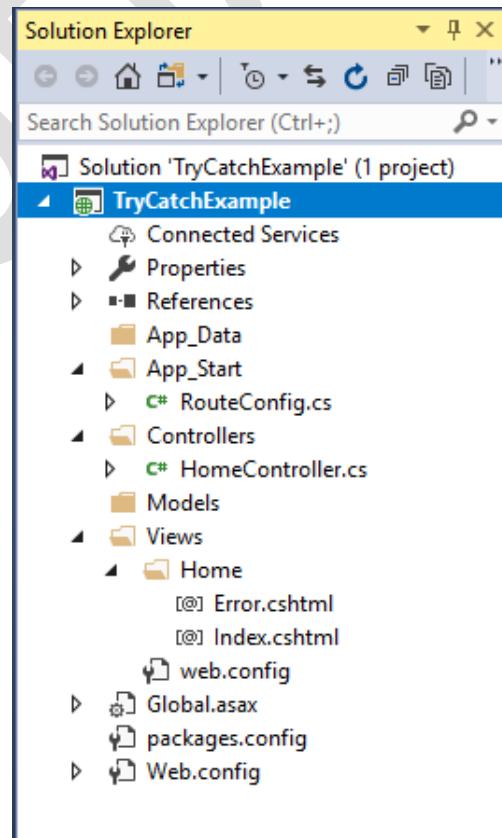
#### **Code for "HomeController.cs"**

```

using System;
using System.Web.Mvc;

namespace TryCatchExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            try
            {
                int a = 10;
                int b = 0;
                int c = a / b;
                return View();
            }
            catch (Exception ex)
            {
                ViewBag.Message = ex.Message;
                return View("Error");
            }
        }
    }
}

```



#### **Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### **Code for "Views\Home\Index.cshtml"**

```

<html>
  <head>
    <title>Index</title>
  </head>
  <body>

```

```
<h1>Index</h1>
</body>
</html>
```

#### Creating Error.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Error". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

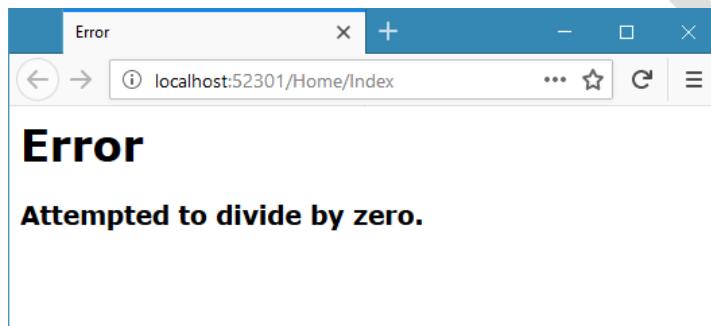
#### Code for "Views\Home\Error.cshtml"

```
<html>
<head>
<title>Error</title>
</head>
<body>
<h1>Error</h1>
<h3>@ViewBag.Message</h3>
</body>
</html>
```

#### Running the application

- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Index".

Output:



## OnException

- This is second way of exception handling in MVC. OnException() is a virtual method in System.Web.Mvc.Controller class, which can be overridden in any controller. This method executes automatically when an exception is raised while executing any action method of the current controller.
- Advantage:** Easy to use.
- Disadvantage:** The OnException method is applied only on the same controller. If needed, we need to defined OnException() method in all the controllers manually.
- Syntax:**

```
using System.Web.Mvc;
public class classnameController : Controller
{
    protected override void OnException(ExceptionContext filterContext)
    {
        base.OnException(filterContext);
        filterContext.ExceptionHandled = true;
        filterContext.Result = View("viewname");
    }
}
```

## OnException - Example

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "OnExceptionExample". Type the location as "C:\Mvc". Type the solution

name as "OnExceptionExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### **Creating HomeController.cs**

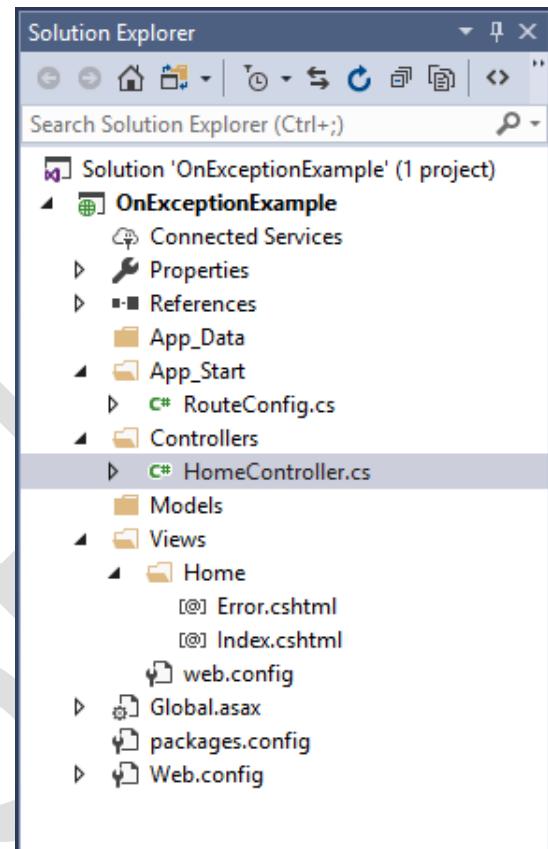
- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### **Code for "HomeController.cs"**

```
using System;
using System.Web.Mvc;

namespace OnExceptionExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            int a = 10;
            int b = 0;
            int c = a / b;
            return View();
        }

        protected override void OnException(ExceptionContext filterContext)
        {
            base.OnException(filterContext);
            filterContext.ExceptionHandled = true;
            ViewData["Message"] = filterContext.Exception.Message;
            filterContext.Result = View("Error", ViewData);
        }
    }
}
```



### **Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### **Code for "Views\Home\Index.cshtml"**

```
<html>
<head>
    <title>Index</title>
</head>
<body>
    <h1>Index</h1>
</body>
</html>
```

### **Creating Error.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Error". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

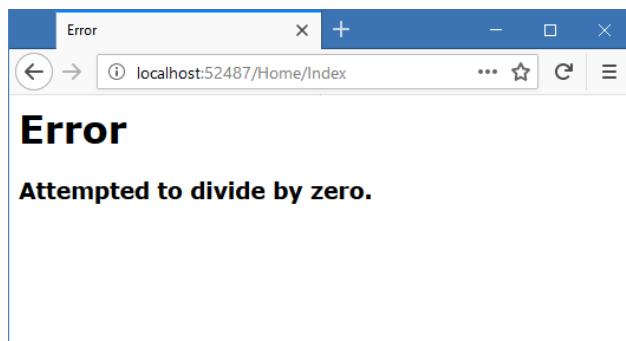
### **Code for "Views\Home\Error.cshtml"**

```
<html>
<head>
    <title>Error</title>
</head>
<body>
    <h1>Error</h1>
    <h3>@ViewBag.Message</h3>
</body>
</html>
```

### Running the application

- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Index".

Output:



## IExceptionFilter

- This is third way of exception handling in MVC. IExceptionFilter executes automatically when an exception is raised while executing any action method of the controller.
- **Advantage:** Can be applied on multiple controllers at-a-time.
- **Disadvantage:** Difficult than previous approaches.
- **Syntax:**

```
using System.Web.Mvc;
public class classnameAttribute : FilterAttribute, IExceptionFilter
{
    protected void OnException(ExceptionContext filterContext)
    {
        filterContext.ExceptionHandled = true;
        filterContext.Result = new ViewResult() { ViewName = "viewname" };
    }
}
```

## IExceptionFilter - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "IExceptionFilterExample". Type the location as "C:\Mvc". Type the solution name as "IExceptionFilterExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating SampleExceptionFilterAttribute.cs

- Open Solution Explorer. Right click on the project (IExceptionFilterExample) and click on "Add" - "New Folder". Type the folder name as "CustomFilters" and press Enter. Right click on "CustomFilters" folder and click on "Add" - "Code" - "Class". Type the filename as "SampleExceptionFilterAttribute". Click on "Add".

### Code for "CustomFilters\SampleExceptionFilterAttribute.cs"

```
using System;
using System.Web.Mvc;

namespace IExceptionFilterExample.CustomFilters
{
    public class SampleExceptionFilterAttribute : FilterAttribute, IExceptionFilter
    {
        public void OnException(ExceptionContext filterContext)
        {
            filterContext.ExceptionHandled = true;
            filterContext.Controller.ViewBag.Message += filterContext.Exception.Message;
            filterContext.Controller.ViewBag.ControllerName = filterContext.Controller.GetType();
            filterContext.Controller.ViewBag.ActionName = Convert.ToString(filterContext.RouteData.Values["Action"]);
        }
    }
}
```

```

        filterContext.Result = new ViewResult() { ViewName = "Error", ViewData = filterContext.Controller.ViewData };
    }
}
}

```

### Creating FilterConfig.cs

- Right click on "App\_Start" folder and click on "Add" - "Code" - "Class". Type the filename as "FilterConfig.cs". Click on "Add".

### Code for "App\_Start\FilterConfig.cs"

```

using System;
using System.Web.Mvc;
using IExceptionFilterExample.CustomFilters;

```

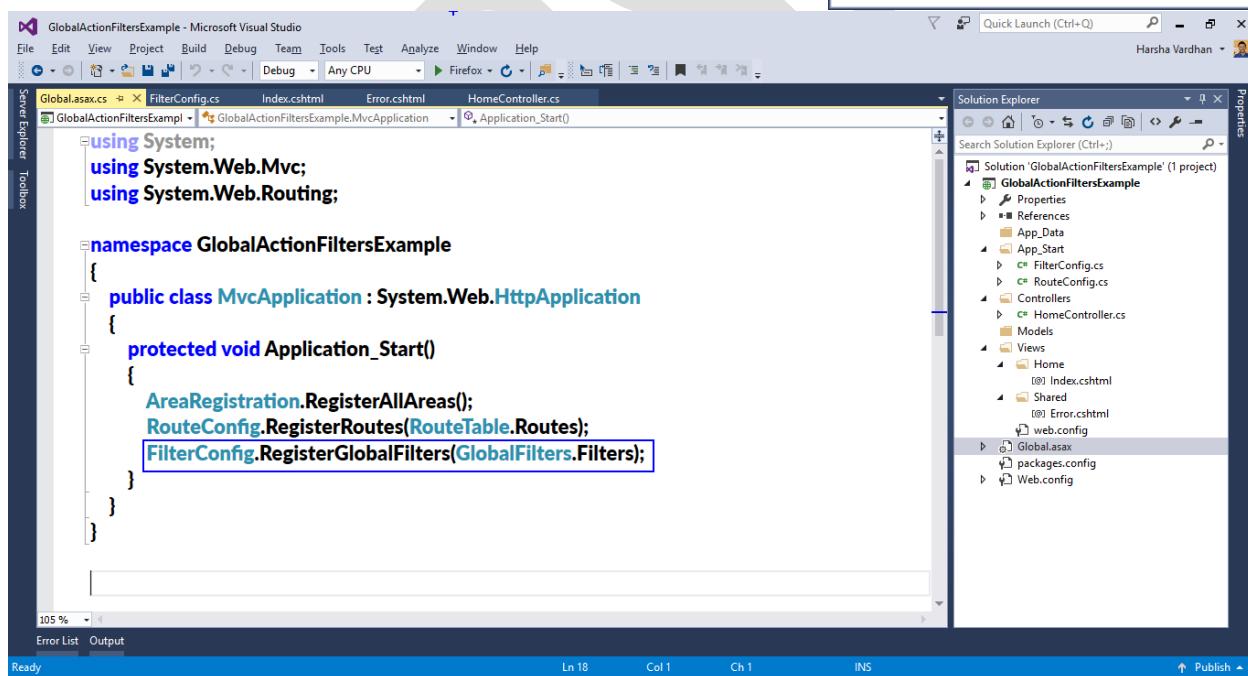
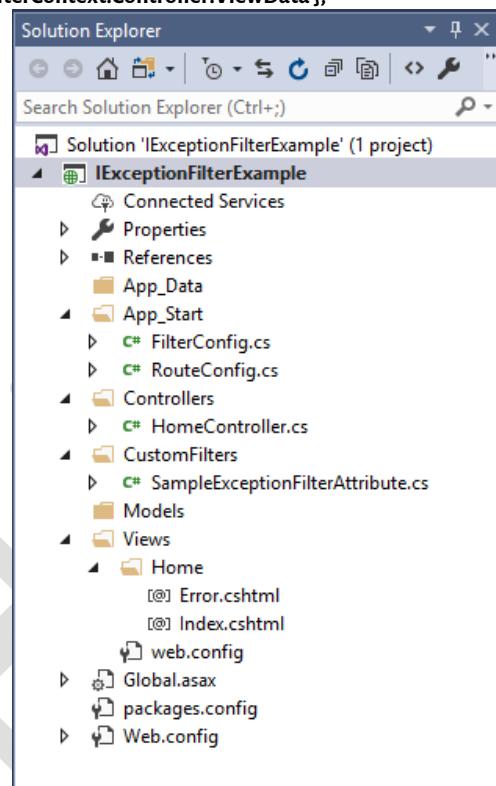
```

namespace IExceptionFilterExample
{
    public class FilterConfig
    {
        public static void RegisterGlobalFilters(GlobalFilterCollection filters)
        {
            filters.Add(new SampleExceptionFilterAttribute());
        }
    }
}

```

### Adding Code to "Global.asax"

```
FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
```



### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```

using System;
using System.Web.Mvc;

```

```
namespace IExceptionFilterExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            int a = 10;
            int b = 0;
            int c = a / b;
            return View();
        }
    }
}
```

**Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Index.cshtml"**

```
<html>
  <head>
    <title>Index</title>
  </head>
  <body>
    <h1>Index</h1>
  </body>
</html>
```

**Creating Error.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Error". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

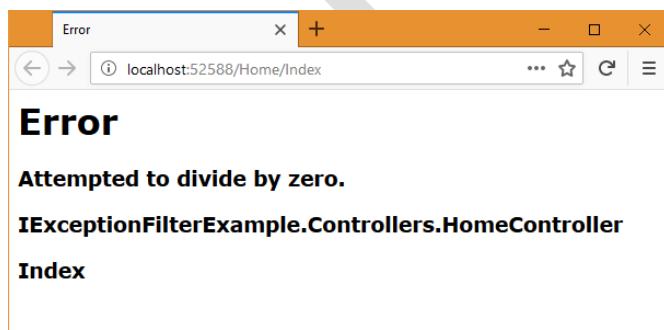
**Code for "Views\Home\Error.cshtml"**

```
<html>
  <head>
    <title>Error</title>
  </head>
  <body>
    <h1>Error</h1>
    <h3>@ViewBag.Message</h3>
    <h3>@ViewBag.ControllerName</h3>
    <h3>@ViewBag.ActionName</h3>
  </body>
</html>
```

**Running the application**

- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Index".

Output:



## HandleError

- This is fourth way of exception handling in MVC. The pre-defined class called "HandleErrorAttribute" already implements IExceptionFilter; so the developer need not implement that interface again, but you can call "Error.cshtml" view when exception is raised. The "HandleErrorInfo" class represents exception details, current action name, current controller name.
- Advantage:** No need to implement IExceptionFilter interface manually.
- Disadvantage:** Not possible to add custom logic on exception.

**Syntax for Controller Level:**

```
using System.Web.Mvc;
[HandleError( ExceptionType = typeof(Exceptionclassname), View = "viewname" )]
public class classname : Controller
{
    ...
}
```

**Syntax for Global Level:**

```
filters.Add(new HandleErrorAttribute() { ExceptionType = typeof(Exceptionclassname), View = "viewname" });
```

## HandleError - Example

**Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "HandleErrorExample". Type the location as "C:\Mvc". Type the solution name as "HandleErrorExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

**Creating FilterConfig.cs**

- Right click on "App\_Start" folder and click on "Add" - "Code" - "Class". Type the filename as "FilterConfig.cs". Click on "Add".

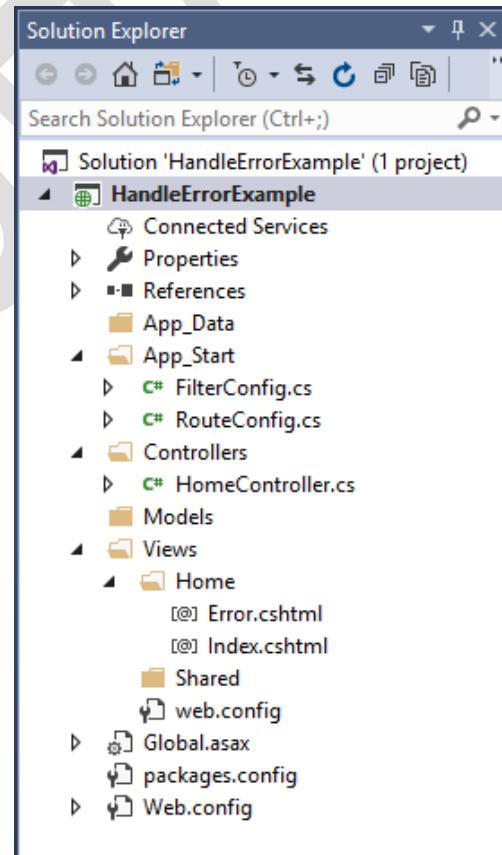
**Code for "App\_Start\FilterConfig.cs"**

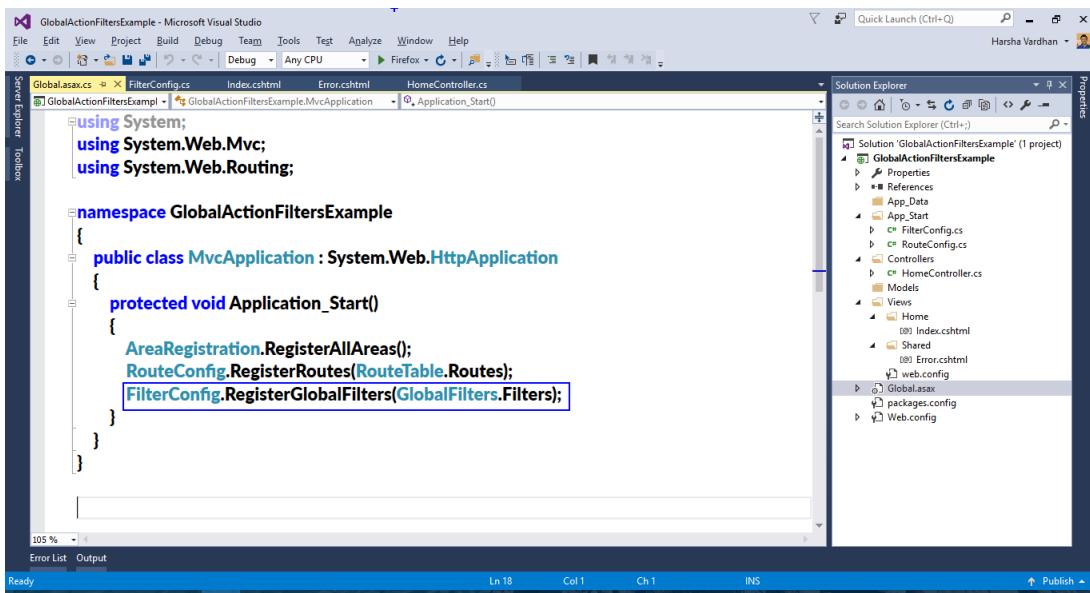
```
using System;
using System.Web.Mvc;

namespace HandleErrorExample
{
    public class FilterConfig
    {
        public static void RegisterGlobalFilters(GlobalFilterCollection filters)
        {
            filters.Add(new HandleErrorAttribute() { ExceptionType =
typeof(Exception), View = "Error" });
        }
    }
}
```

**Adding Code to "Global.asax"**

```
FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
```





## Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

## Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;

namespace HandleErrorExample.Controllers
{
    public class HomeController : Controller
    {
        [HandleError(ExceptionType = typeof(Exception), View = "Error")]
        public ActionResult Index()
        {
            int a = 10;
            int b = 0;
            int c = a / b;
            return View();
        }
    }
}
```

## Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

## Code for "Views\Home\Index.cshtml"

```
<html>
<head>
    <title>Index</title>
</head>
<body>
    <h1>Index</h1>
</body>
</html>
```

## Creating Error.cshtml

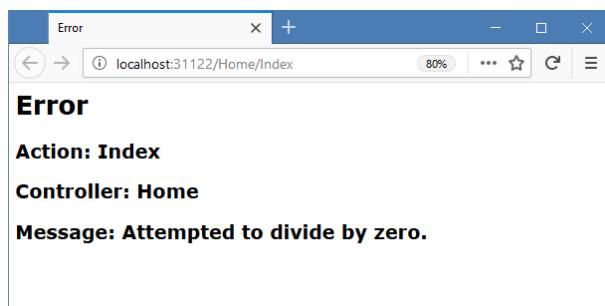
- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Error". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Error.cshtml"**

```
@model HandleErrorInfo
<html>
<head>
<title>Error</title>
</head>
<body>
<h1>Error</h1>
<h2>Action: @Model.ActionName</h2>
<h2>Controller: @Model.ControllerName</h2>
<h2>Message: @Model.Exception.Message</h2>
</body>
</html>
```

**Running the application**

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

**Output:****HTTP Errors**

- This concept is used to handle the http errors like "404 - Page Not Found", "401 - Unauthorized", "500 - Internal Server Error" etc.
- **Syntax in Web.config:**

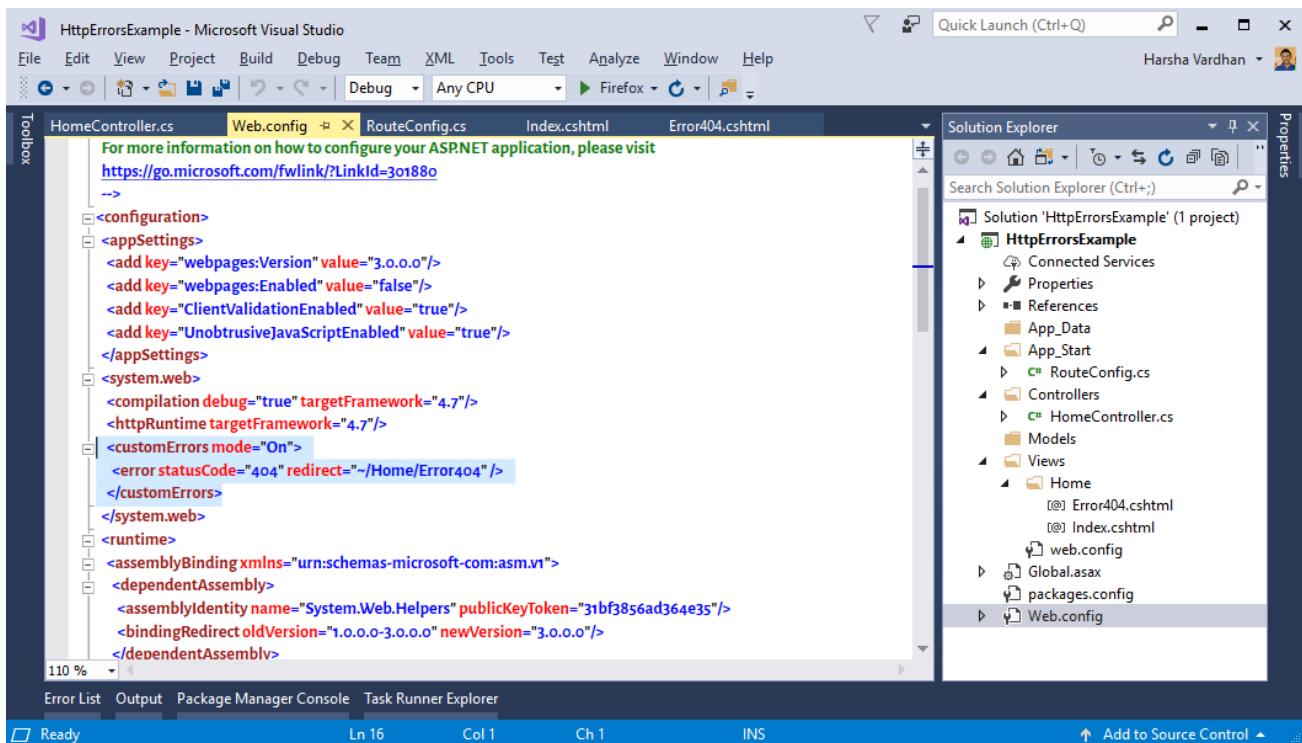
```
<customErrors mode="On">
<error statusCode="404" redirect="~/controller/action" />
<error statusCode="401" redirect="~/controller/action" />
<error statusCode="500" redirect="~/controller/action" />
...
</customErrors>
```

**HTTP Errors - Example****Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "HttpErrorsExample". Type the location as "C:\Mvc". Type the solution name as "HttpErrorsExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

**Adding Code to "Web.config"**

```
<customErrors mode="On">
<error statusCode="404" redirect="~/Home/Error404" />
</customErrors>
```



### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;

namespace HttpErrorsExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        public ActionResult Error404()
        {
            return View();
        }
    }
}
```

### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### Code for "Views\Home\Index.cshtml"

```
<html>
<head>
    <title>Index</title>
</head>
```

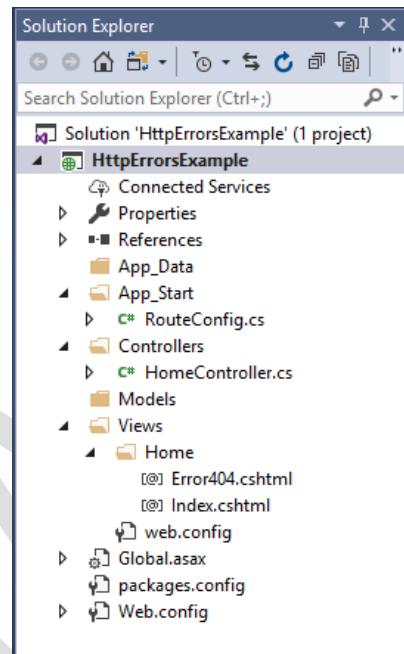
```
<body>
<h1>Index</h1>
</body>
</html>
```

### Creating Error404.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Error404". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### Code for "Views\Home\Error.cshtml"

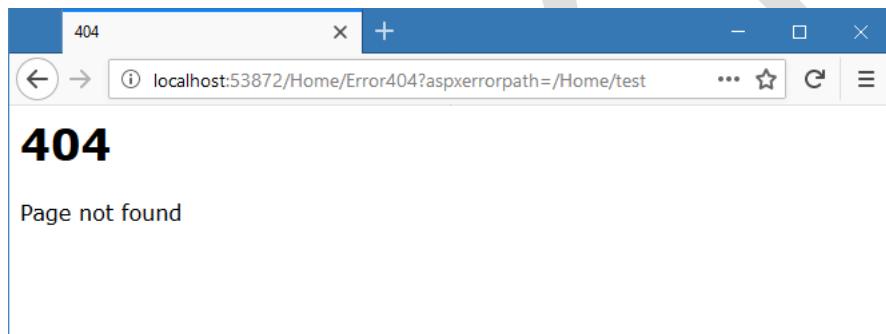
```
<html>
<head>
<title>404</title>
</head>
<body>
<h1>404</h1>
<p>Page not found</p>
</body>
</html>
```



### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/test".

#### Output:



## Application\_Error

- This concept is used to handle all exceptions.
- Syntax in Global.asax:**

```
protected void Application_Error()
{
    Response.Redirect("~/controller/action");
}
```

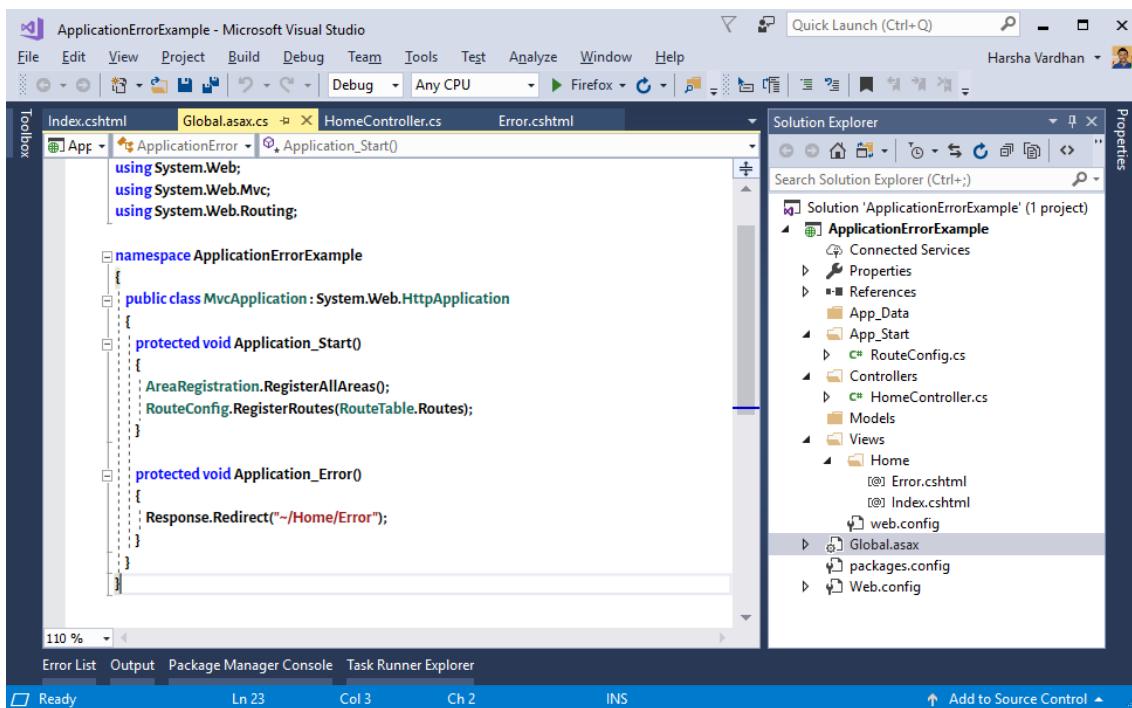
## Application\_Error - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "ApplicationErrorExample". Type the location as "C:\Mvc". Type the solution name as "ApplicationErrorExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Adding Code to "Global.asax"

```
protected void Application_Error()
{
    Response.Redirect("~/Home/Error");
}
```



### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```

using System;
using System.Web.Mvc;

namespace ApplicationErrorExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            int a = 10;
            int b = 0;
            int c = a / b;
            return View();
        }

        public ActionResult Error()
        {
            return View();
        }
    }
}

```

### Creating Index.cshtml

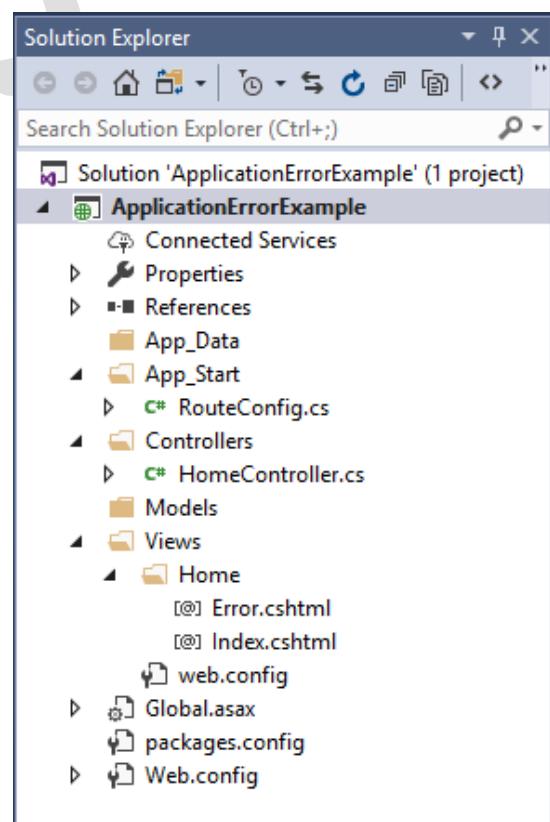
- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### Code for "Views\Home\Index.cshtml"

```

<html>
<head>

```



```
<title>Index</title>
</head>
<body>
  <h1>Index</h1>
</body>
</html>
```

#### Creating Error.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Error". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

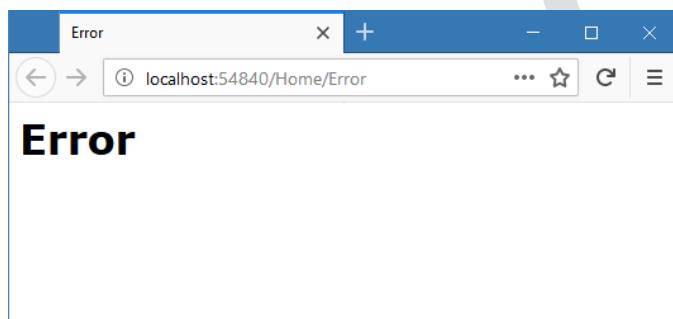
#### Code for "Views\Home\Error.cshtml"

```
<html>
  <head>
    <title>Error</title>
  </head>
  <body>
    <h1>Error</h1>
  </body>
</html>
```

#### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



## ATTRIBUTE ROUTING

### Intro to Attribute Routing

- "Attribute Routing" (also known as "Attribute Based Routing") is the new way of routing. **Advantage:** We will define the url route within the action method itself, instead of defining in the RouteConfig.cs. So that there is no chance of conflicts between routes. To enable Attribute routing, use the following statement in "RouteConfig.cs" file:

```
routes.MapMvcAttributeRoutes();
```

#### Syntax of Attribute Routing

```
[Route("url template here")]
```

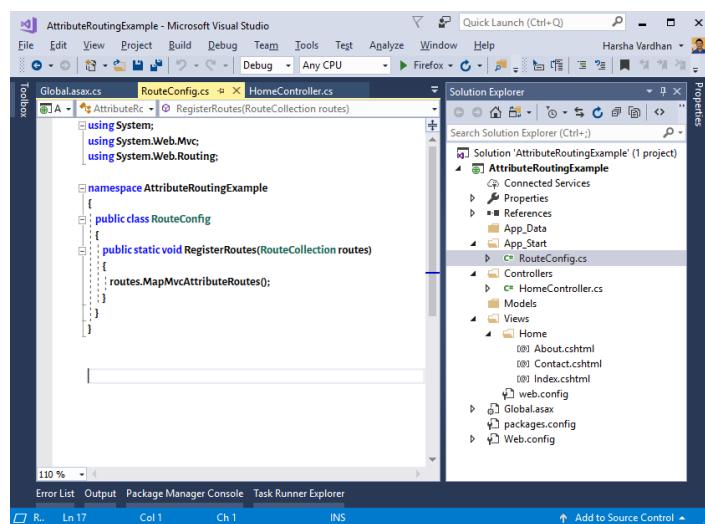
### Attribute Routing - Example

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "AttributeRoutingExample". Type the location as "C:\Mvc". Type the solution name as "AttributeRoutingExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

#### Adding Code to "RouteConfig.cs"

```
public static void RegisterRoutes(RouteCollection routes)
{
  routes.MapMvcAttributeRoutes();
}
```



## Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

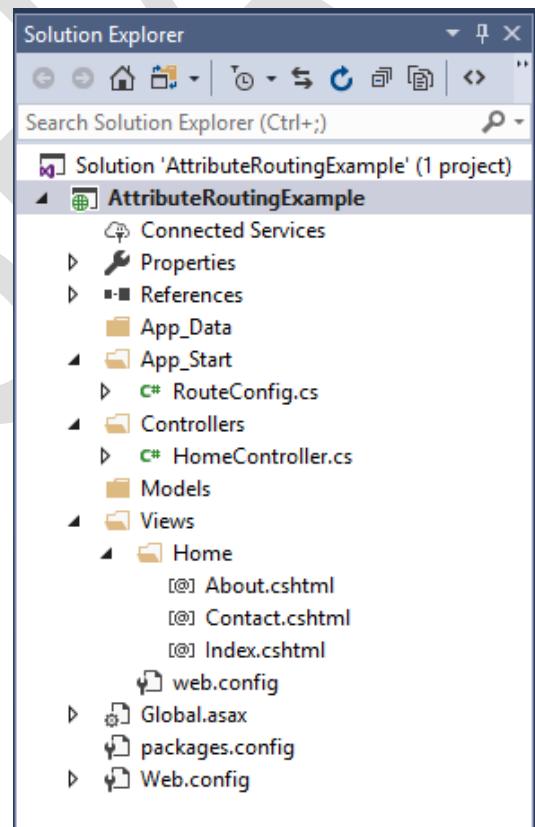
## Code for "HomeController.cs"

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

namespace AttributeRoutingExample.Controllers
{
    public class HomeController : Controller
    {
        [Route("Index")]
        public ActionResult Index()
        {
            return View();
        }

        [Route("About")]
        public ActionResult About()
        {
            return View();
        }

        [Route("Contact")]
        public ActionResult Contact()
        {
            return View();
        }
    }
}
```



## Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

## Code for "Views\Home\Index.cshtml"

```
<html>
<head>
    <title>Home</title>
</head>
```

```
<body>
<div>
<a href="/Index">Index</a>
<a href="/About">About</a>
<a href="/Contact">Contact</a>
</div>
<h1>Home page</h1>
</body>
</html>
```

**Creating About.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "About". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\About.cshtml"**

```
<html>
<head>
<title>About</title>
</head>
<body>
<div>
<a href="/Index">Index</a>
<a href="/About">About</a>
<a href="/Contact">Contact</a>
</div>
<h1>About page</h1>
</body>
</html>
```

**Creating Contact.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Contact". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home>Contact.cshtml"**

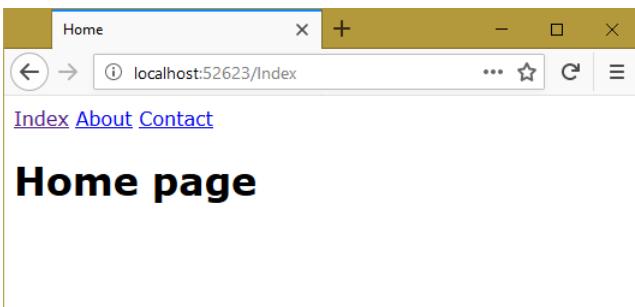
```
<html>
<head>
<title>Contact</title>
</head>
<body>
<div>
<a href="/Index">Index</a>
<a href="/About">About</a>
<a href="/Contact">Contact</a>
</div>
<h1>Contact page</h1>
</body>
</html>
```

**Running the application**

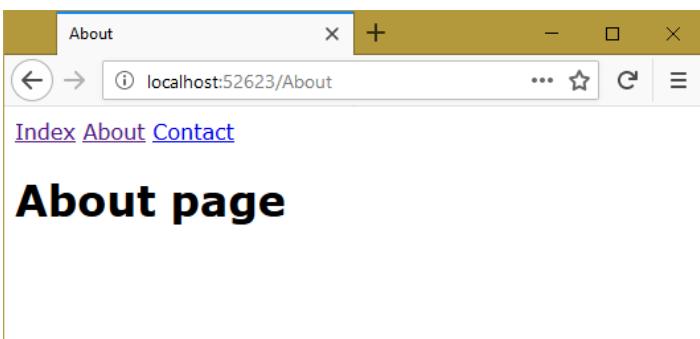
- Press "F5" to run the application.
- Type "http://localhost:portnumber/Index".

Output:

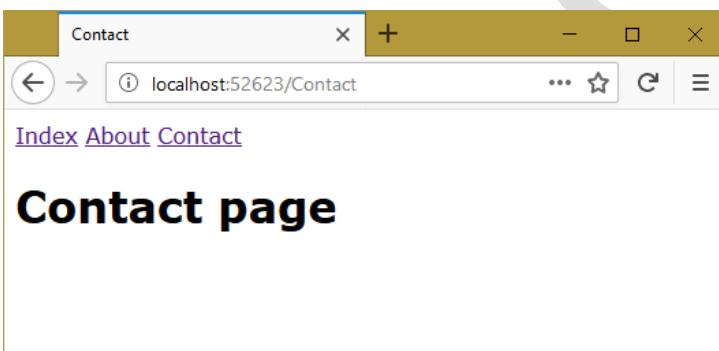
http://localhost:portnumber/Index



<http://localhost:portnumber/>



<http://localhost:portnumber/>



## RoutePrefix

- RoutePrefix is an attribute, which is applied on the controller, to set prefix for all routes of all action methods of the current controller.
- **Adantage:** The user can easily identify the routes.

### Syntax of RoutePrefix

```
[RoutePrefix("prefix here")]
```

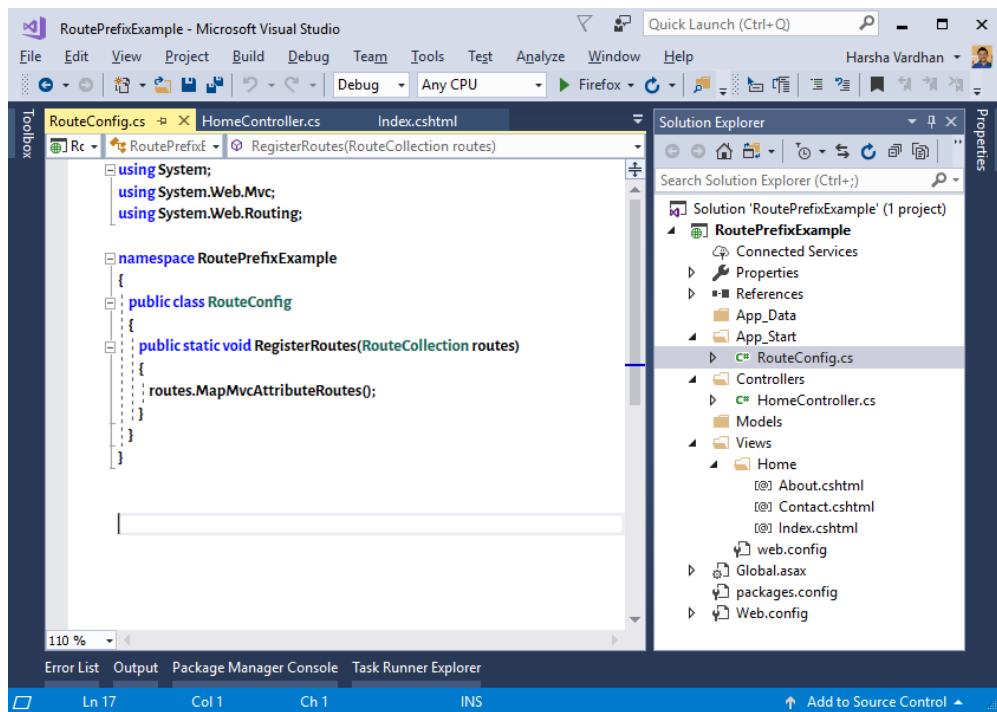
## RoutePrefix - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "RoutePrefixExample". Type the location as "C:\Mvc". Type the solution name as "RoutePrefixExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Adding Code to "RouteConfig.cs"

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.MapMvcAttributeRoutes();
}
```



### **Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### **Code for "HomeController.cs"**

```
using System;
using System.Web.Mvc;

namespace RoutePrefixExample.Controllers
{
    [RoutePrefix("Home")]
    public class HomeController : Controller
    {
        [Route("Index")]
        public ActionResult Index()
        {
            return View();
        }

        [Route("About")]
        public ActionResult About()
        {
            return View();
        }

        [Route("~/Contact")]
        public ActionResult Contact()
        {
            return View();
        }
    }
}
```

### **Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Index.cshtml"**

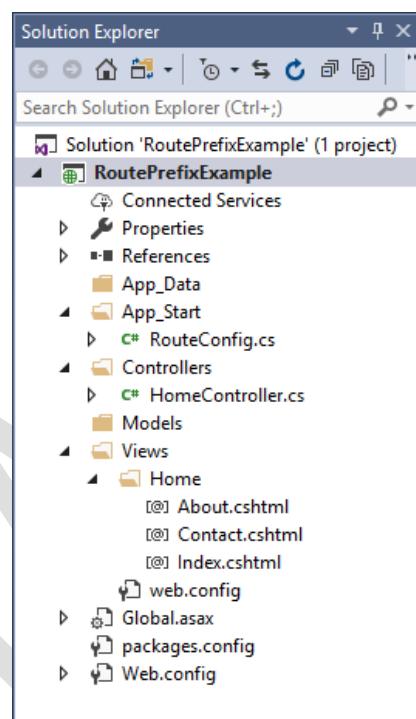
```
<html>
<head>
<title>Home</title>
</head>
<body>
<div>
<a href="/Home/Index">Index</a>
<a href="/Home/About">About</a>
<a href="/Contact">Contact</a>
</div>
<h1>Home page</h1>
</body>
</html>
```

**Creating About.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "About". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\About.cshtml"**

```
<html>
<head>
<title>About</title>
</head>
<body>
<div>
<a href="/Home/Index">Index</a>
<a href="/Home/About">About</a>
<a href="/Contact">Contact</a>
</div>
<h1>About page</h1>
</body>
</html>
```

**Creating Contact.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Contact". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

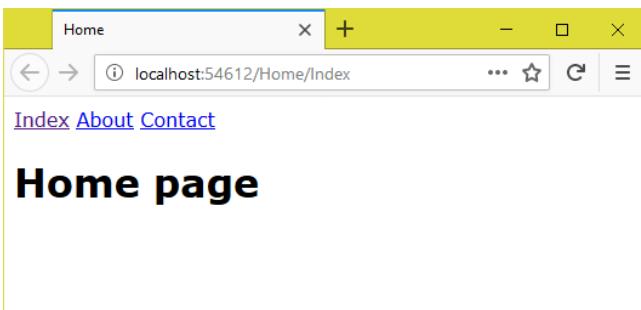
**Code for "Views\Home>Contact.cshtml"**

```
<html>
<head>
<title>Contact</title>
</head>
<body>
<div>
<a href="/Home/Index">Index</a>
<a href="/Home/About">About</a>
<a href="/Contact">Contact</a>
</div>
<h1>Contact page</h1>
</body>
</html>
```

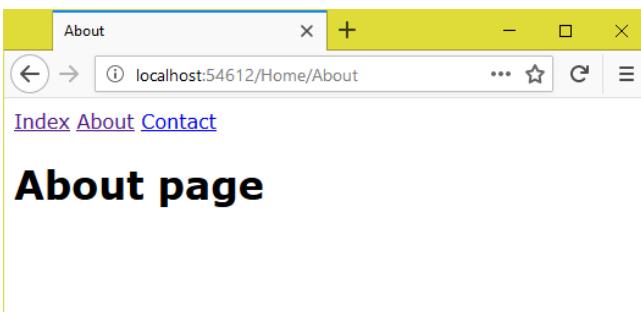
**Running the application**

- Press "F5" to run the application.
  - Type "http://localhost:portnumber/Home/Index".
- Output:

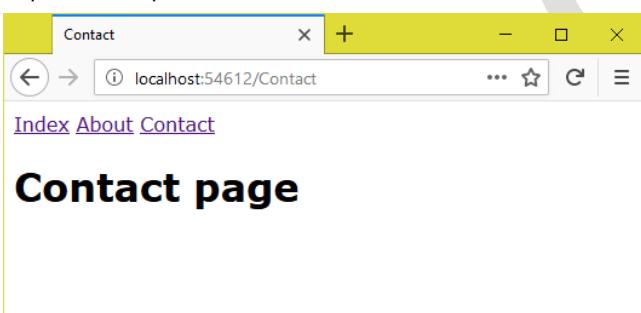
<http://localhost:portnumber/Home/Index>



<http://localhost:portnumber/Home/About>



<http://localhost:portnumber/Contact>



## Route Parameters

- Parameters are the dynamic values accepted in the route.
- We can specify parameter name, data type and also add "?", if the parameter is optional.
- Advantage:** The user can pass any values at run time.

### Syntax of RouteParameters

[Route("url/{parameter:datatype?}")]

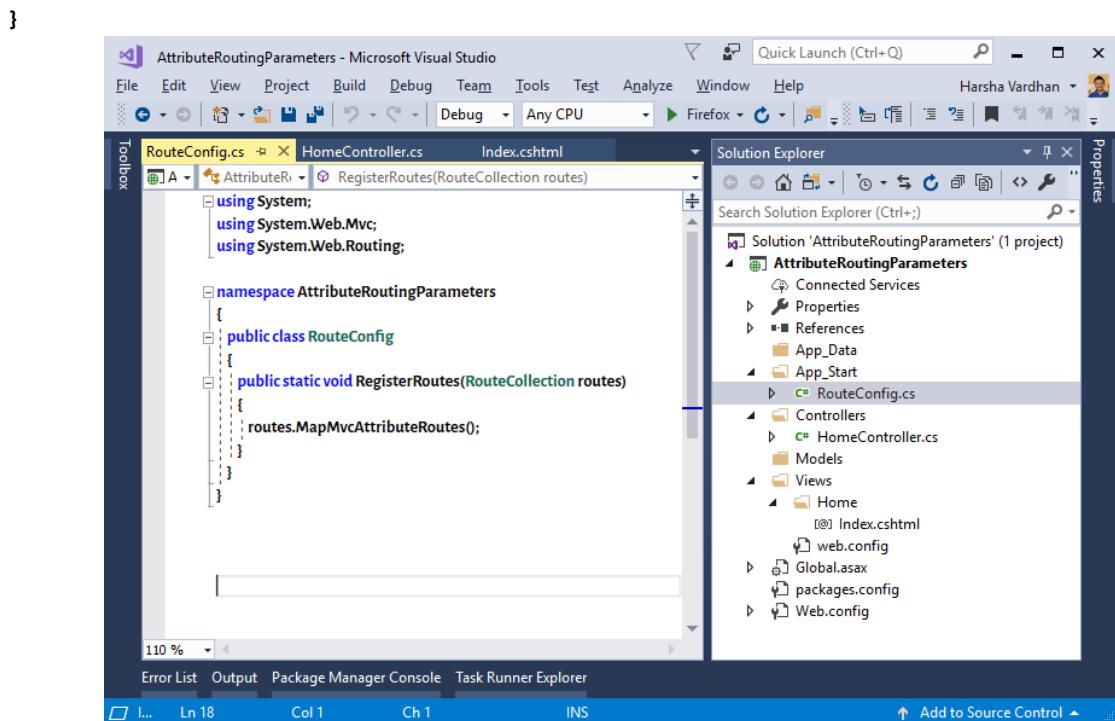
## Route Parameters - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "AttributeRoutingParameters". Type the location as "C:\Mvc". Type the solution name as "AttributeRoutingParameters". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Adding Code to "RouteConfig.cs"

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.MapMvcAttributeRoutes();
```



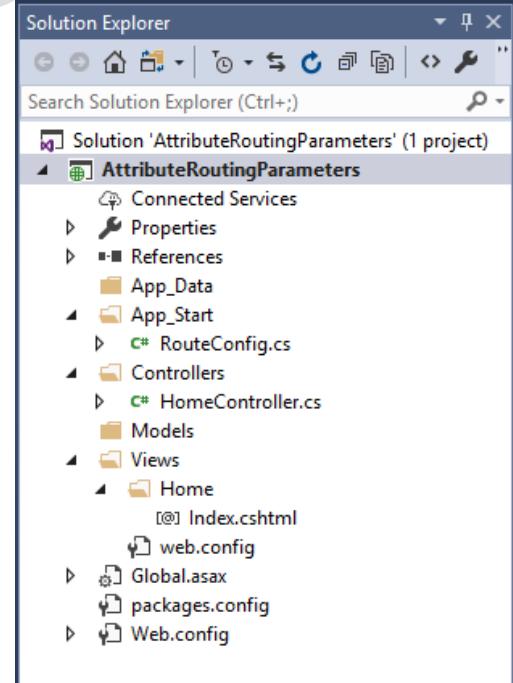
### **Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### **Code for "HomeController.cs"**

```
using System;
using System.Web.Mvc;

namespace AttributeRoutingParameters.Controllers
{
    public class HomeController : Controller
    {
        [Route("Home/Index/{id:int?}")]
        public ActionResult Index(int? id)
        {
            ViewBag.msg = id;
            return View();
        }
    }
}
```



### **Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### **Code for "Views\Home\Index.cshtml"**

```
<html>
<head>
    <title>Attribute Routing - Parameters</title>
</head>
<body>
    <h1>Attribute Routing - Parameters</h1>
    <h2>ID is: @ViewBag.msg</h2>
</body>
</html>
```

**Running the application**

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index/100".

Output:

**AREAS****Introduction to Areas**

- Areas are used to divide the project into multiple parts; each part is called as "area".
  - Ex: Admin, Agents, Users etc.
- Each area contains its own set of controllers, models and views. We can easily navigate from one area to another area, by using the following url syntax:

{area}/{controller}/{action}

**Syntax of area folder**

- Areas
  - AreaName
    - Controllers
    - Models
    - Views

**Syntax of Area Registration**

```
using System.Web.Mvc;

namespace namespaceName
{
    public class AreanameAreaRegistration : AreaRegistration
    {
        public override string AreaName
        {
            get
            {
                return "name here";
            }
        }

        public override void RegisterArea(AreaRegistrationContext context)
        {
            context.MapRoute(
                "routename",
                "UrlPrefix/{controller}/{action}/{id}",
                new { action = "Index", id = UrlParameter.Optional });
        }
    }
}
```

## Areas - Example

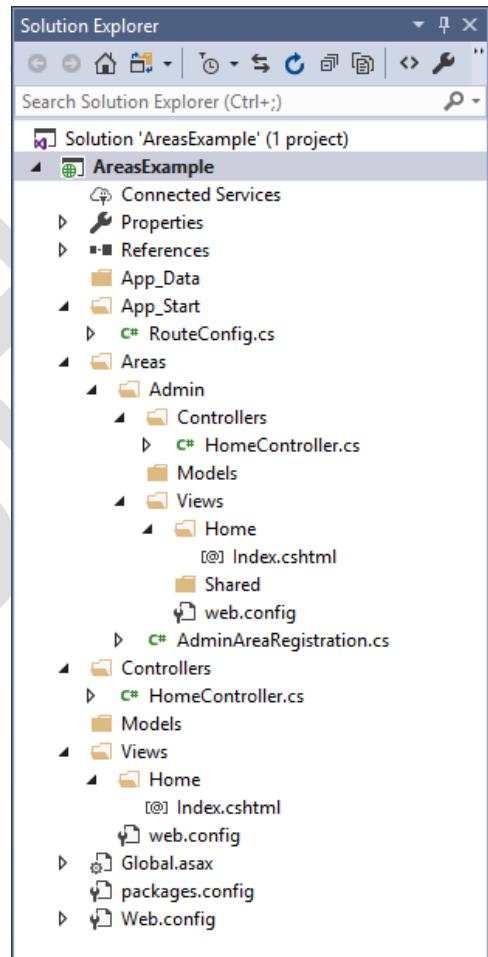
### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "AreasExample". Type the location as "C:\Mvc". Type the solution name as "AreasExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests".
- Click on OK.

### Code for RouteConfig.cs

```
using System;
using System.Web.Mvc;
using System.Web.Routing;

namespace AreasExample
{
    public class RouteConfig
    {
        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { controller = "Home", action = "Index", id =
UrlParameter.Optional },
                namespaces: new string[] { "AreasExample.Controllers" }
            );
        }
    }
}
```



### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;

namespace AreasExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### Code for "Views\Home\Index.cshtml"

```
<html>
<head>
    <title>Home/Index</title>
</head>
<body>
    <h1>Home/Index</h1>
```

```
<a href="/Admin/Home/Index">Go to Admin/Home/Index</a>
</body>
</html>
```

### Creating "Admin" Area

- Right click on the project (AreasExample) and click on "Add" - "Area". Type the area name as "Admin" and click on "Add".

### Code for "Areas\Admin\AdminAreaRegistration.cs"

```
using System.Web.Mvc;

namespace AreasExample.Areas.Admin
{
    public class AdminAreaRegistration : AreaRegistration
    {
        public override string AreaName
        {
            get
            {
                return "Admin";
            }
        }

        public override void RegisterArea(AreaRegistrationContext context)
        {
            context.MapRoute(
                "Admin_default",
                "Admin/{controller}/{action}/{id}",
                new { action = "Index", id = UrlParameter.Optional });
        }
    }
}
```

### Creating HomeController.cs in "Admin" Area

- Right click on "Controllers" folder in "Areas\Admin" and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "Areas\Admin\Controllers\HomeController.cs"

```
using System;
using System.Web.Mvc;

namespace AreasExample.Areas.Admin.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

### Creating Index.cshtml in "Admin" Area

- Right click on "Views\Home" folder in "Areas\Admin" and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### Code for "Areas\Admin\Views\Home\Index.cshtml"

```
<html>
<head>
    <title>Admin/Home/Index</title>
</head>
```

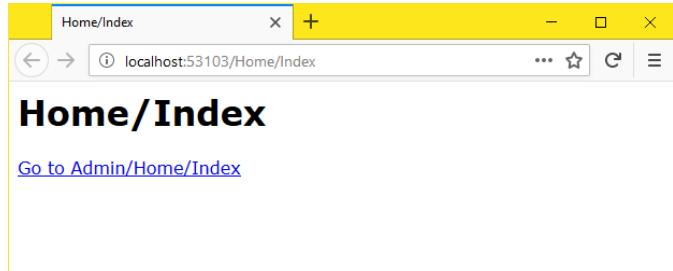
```
<body>
<h1>Admin/Home/Index</h1>
<a href="/Home/Index">Go to Home/Index</a>
</body>
</html>
```

### Running the application

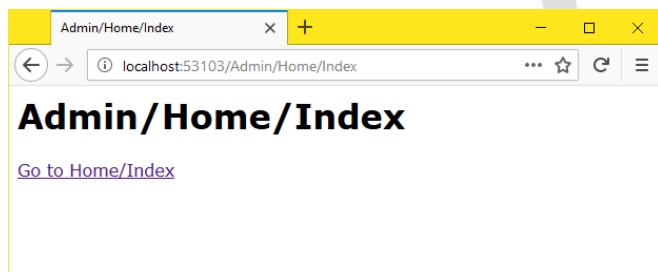
- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:

http://localhost:portnumber/Home/Index



http://localhost:portnumber/Admin/Home/Index



## INTEGRATIONS

### Npm Integration

- "Npm" stands for "NodeJS Package Manager". "Npm" is a Package Repository, which contains thousands of packages available on the server. The developer can install any package through command line / visual studio.
  - Ex: jQuery, AngularJS, Gulp, Grunt etc.

### Syntax of package.json

```
{
  "version": "1.0.0",
  "name": "asp.net",
  "private": true,
  "devDependencies": {
    "packagename": "version"
  }
}
```

## Npm - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name as "NpmIntegrationExample". Type the location as "C:\Mvc". Type the solution name as "NpmIntegrationExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

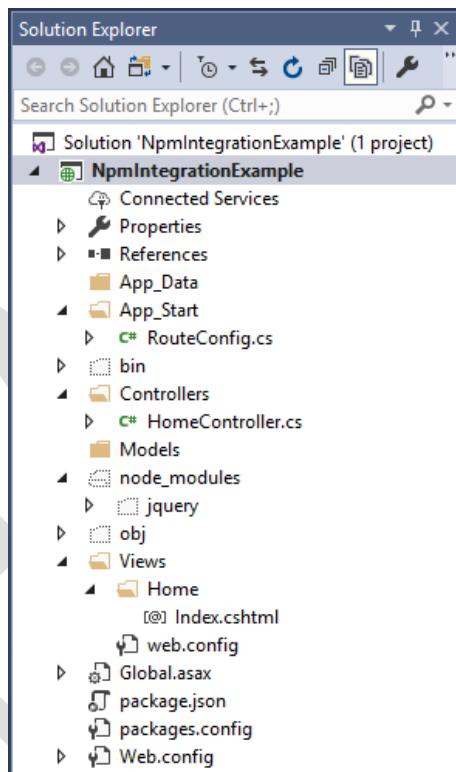
### Creating package.json

- Right click on the project (NpmIntegrationExample) and click on "Add" - "New Item" - "Web" - "Npm Configuration File". Type the filename as "package.json" (default) and click on "Add".

### Code for "package.json"

```
{
  "version": "1.0.0",
  "name": "asp.net",
  "private": true,
  "devDependencies":
  {
    "jquery": "latest"
  }
}
```

- To install the specified packages (Ex: jquery), right click on "package.json" file and click on "Restore Packages". To see the installed packages, go to "Project" menu - "Show All Files". It shows "node\_modules" folder with the installed packages. Each package is shown as a package. For example, "jquery" package is a folder.



### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;

namespace NpmIntegrationExample.Controllers
{
  public class HomeController : Controller
  {
    public ActionResult Index()
    {
      return View();
    }
  }
}
```

### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

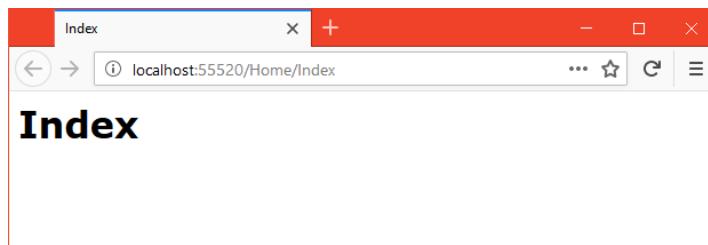
### Code for "Views\Home\Index.cshtml"

```
<html>
  <head>
    <title>Index</title>
  </head>
  <body>
    <h1>Index</h1>
  </body>
</html>
```

### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



## jQuery Integration

- "jQuery" is a "JavaScript Library", which provides a set of functions to perform DOM Manipulations and AJAX easily.

### How to Work with jQuery

#### Import jQuery in package.json

```
{  
  "version": "1.0.0",  
  "name": "asp.net",  
  "private": true,  
  "devDependencies":  
  {  
    "jquery": "latest"  
  }  
}
```

#### Import jQuery in the view

```
<script src="~/node_modules/jquery/dist/jquery.js"></script>
```

#### jQuery Important Functions

- `$(selector)` : It selects the set of elements based on the condition.
- `html()` : It sets / gets inner html of the tag.
- `val()` : It sets / gets value of `<input>` tag or `<select>` tag.
- `append()` : It appends new content to inner html of the tag.
- `remove()` : It removes the tag.

## jQuery - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "jQueryIntegrationExample". Type the location as "C:\Mvc". Type the solution name as "jQueryIntegrationExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating package.json

- Right click on the project (jQueryIntegrationExample) and click on "Add" - "New Item" - "Web" - "Npm Configuration File". Type the filename as "package.json" (default) and click on "Add".

#### Code for "package.json"

```
{  
  "version": "1.0.0",  
  "name": "asp.net",  
  "private": true,  
  "devDependencies":  
  {  
    "jquery": "latest"  
  }  
}
```

```

"name": "asp.net",
"private": true,
"devDependencies":
{
  "jquery": "latest"
}
  
```

- To install the specified packages (Ex: jquery), right click on "package.json" file and click on "Restore Packages". To see the installed packages, go to "Project" menu - "Show All Files". It shows "node\_modules" folder with the installed packages.

### **Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### **Code for "HomeController.cs"**

```

using System;
using System.Web.Mvc;

namespace jQueryIntegrationExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
  
```

### **Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### **Code for "Views\Home\Index.cshtml"**

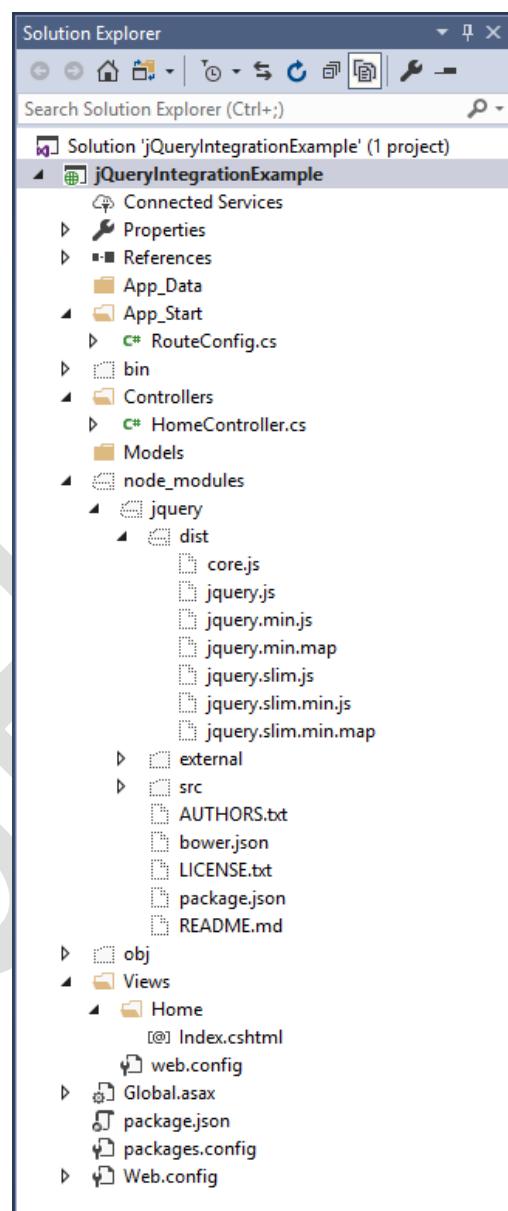
```

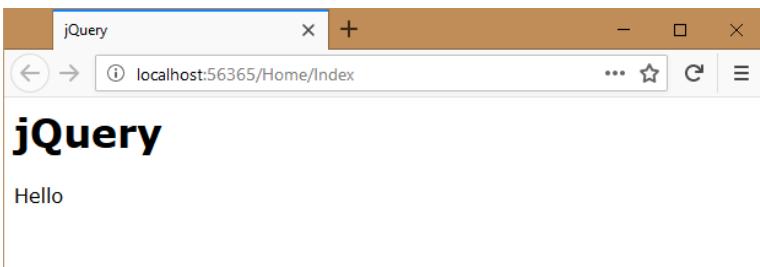
<html>
  <head>
    <title>jQuery</title>
  </head>
  <body>
    <h1>jQuery</h1>
    <p id="p1"></p>
    <script src="~/node_modules/jquery/dist/jquery.js"></script>
    <script>
      $("#p1").html("Hello");
    </script>
  </body>
</html>
  
```

### **Running the application**

- Press "F5" to run the application.
- Type "<http://localhost:portnumber/Home/Index>".

Output:





## Bootstrap Integration

- "Bootstrap" is a "CSS Library", which provides a set of pre-defined CSS classes to apply styles to the web page, without having CSS knowledge. Bootstrap works based on "jQuery". So it is must to import "jquery" package also, apart from "bootstrap" package.

### How to Work with Bootstrap

#### Import Bootstrap and jQuery in package.json

```
{
  "version": "1.0.0",
  "name": "asp.net",
  "private": true,
  "devDependencies": {
    "jquery": "latest",
    "bootstrap": "latest"
  }
}
```

#### Import Bootstrap in the view

```
<link href="~/node_modules/bootstrap/dist/css/bootstrap.css" rel="stylesheet" />
<script src="~/node_modules/jquery/dist/jquery.js"></script>
<script src="~/node_modules/bootstrap/dist/js/bootstrap.js"></script>
```

#### Bootstrap Important Classes

- container : It acts as a outer container.
- btn : It applies style to the button.

## Bootstrap - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "BootstrapIntegrationExample". Type the location as "C:\Mvc". Type the solution name as "BootstrapIntegrationExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating package.json

- Right click on the project (BootstrapIntegrationExample) and click on "Add" - "New Item" - "Web" - "Npm Configuration File". Type the filename as "package.json" (default) and click on "Add".

#### Code for "package.json"

```
{
  "version": "1.0.0",
  "name": "asp.net",
  "private": true,
  "devDependencies": {
    "jquery": "latest",
  }
}
```

```

"bootstrap": "latest"
}
}

```

- To install the specified packages (Ex: bootstrap), right click on "package.json" file and click on "Restore Packages". To see the installed packages, go to "Project" menu - "Show All Files". It shows "node\_modules" folder with the installed packages.

### **Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### **Code for "HomeController.cs"**

```

using System;
using System.Web.Mvc;

namespace BootstrapIntegrationExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}

```

### **Creating Index.cshtml**

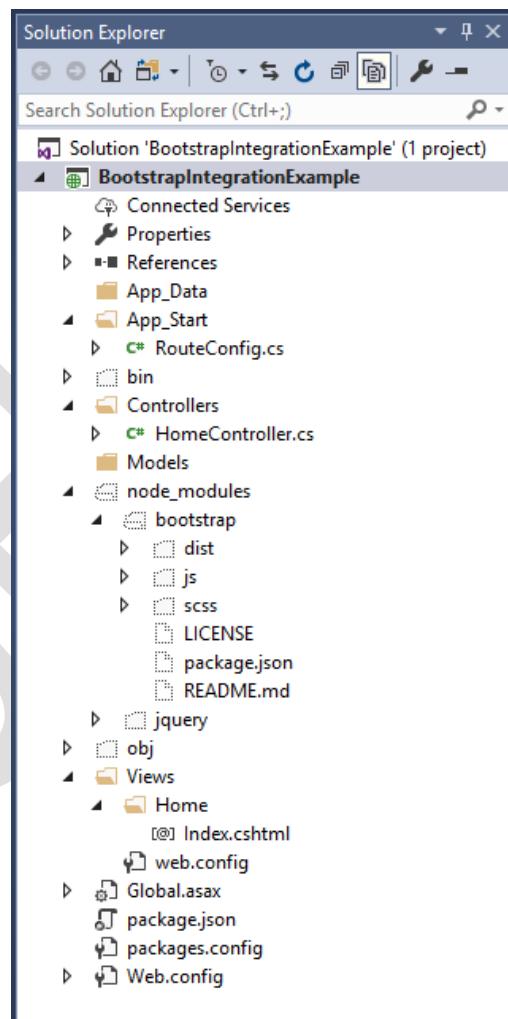
- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### **Code for "Views\Home\Index.cshtml"**

```

<html>
    <head>
        <title>Bootstrap</title>
        <link href="~/node_modules/bootstrap/dist/css/bootstrap.css" rel="stylesheet" />
        <script src="~/node_modules/jquery/dist/jquery.js"></script>
        <script src="~/node_modules/bootstrap/dist/js/bootstrap.js"></script>
    </head>
    <body>
        <div class="container">
            <h1>Bootstrap</h1>
        </div>
    </body>
</html>

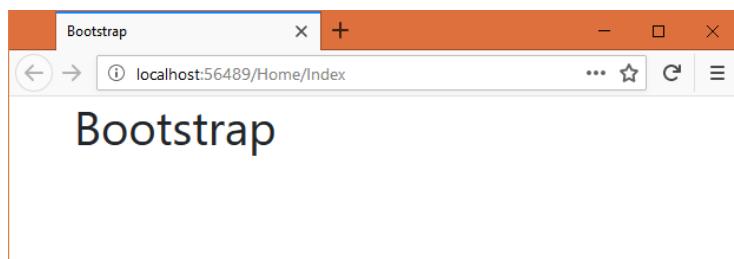
```



### **Running the application**

- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Index".

Output:



## AngularJS Integration

- AngularJS is a "JavaScript Framework", which is used to create data bindings in web pages, using MVC design pattern.
- We will see how to integrate AngularJS 1" in this example.

### **How to Work with AngularJS**

#### **Import AngularJS in package.json**

```
{
  "version": "1.0.0",
  "name": "asp.net",
  "private": true,
  "devDependencies":
  {
    "angular": "latest"
  }
}
```

#### **Import AngularJS in the view**

```
<script src="~/node_modules/angular/angular.js"></script>
```

#### **Creating Basic Application in AngularJS**

```
<script>
  var app = angular.module("module name", []);
  app.controller("controller name", function ($scope)
  {
    $scope.property = value;
  })
</script>

<div ng-controller="controller name">
  {{property}}
</div>
```

## AngularJS - Example

### **Creating Project**

- Open Visual Studio 2017.
- Go to "File" - "New" - "Project".
- Select ".NET Framework 4.7".
- Select "Visual C#".
- Select "ASP.NET Web Application (.NET Framework)".
- Type the project name "AngularJSIntegrationExample".
- Type the location as "C:\Mvc".
- Type the solution name as "AngularJSIntegrationExample".
- Click on OK.
- Select "Empty".
- Check the checkbox "MVC".
- Uncheck the checkbox "Enable Docker support".
- Uncheck the checkbox "Add unit tests".
- Click on OK.

### **Creating package.json**

- Right click on the project (AngularJSIntegrationExample) and click on "Add" - "New Item" - "Web" - "Npm Configuration File". Type the filename as "package.json" (default) and click on "Add".

**Code for "package.json"**

```
{
  "version": "1.0.0",
  "name": "asp.net",
  "private": true,
  "devDependencies":
  {
    "angular": "latest"
  }
}
```

- To install the specified packages (Ex: angular), right click on "package.json" file and click on "Restore Packages".
- To see the installed packages, go to "Project" menu - "Show All Files".
- It shows "node\_modules" folder with the installed packages.

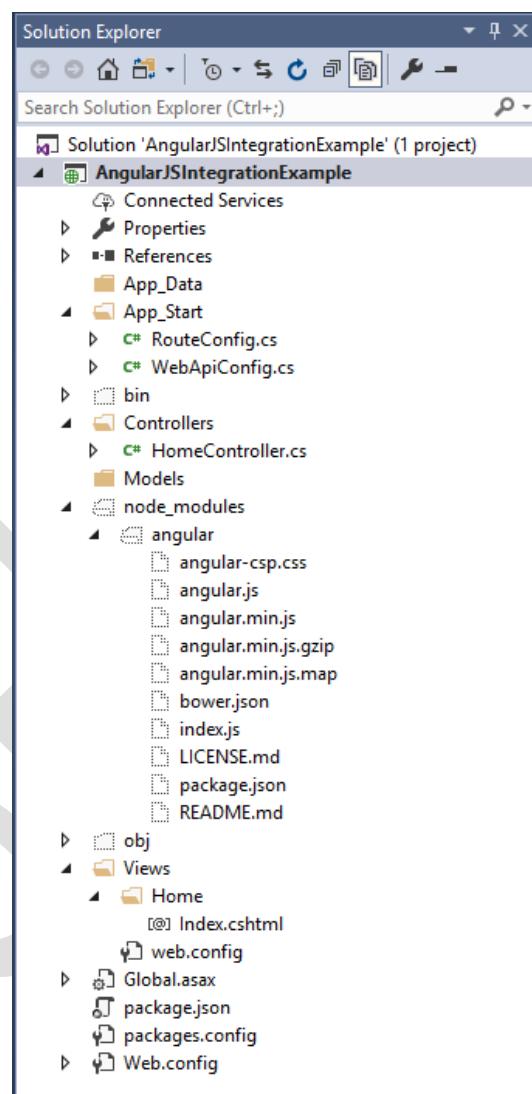
**Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

**Code for "HomeController.cs"**

```
using System;
using System.Web.Mvc;

namespace AngularJSIntegrationExample.Controllers
{
  public class HomeController : Controller
  {
    public ActionResult Index()
    {
      return View();
    }
  }
}
```

**Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Index.cshtml"**

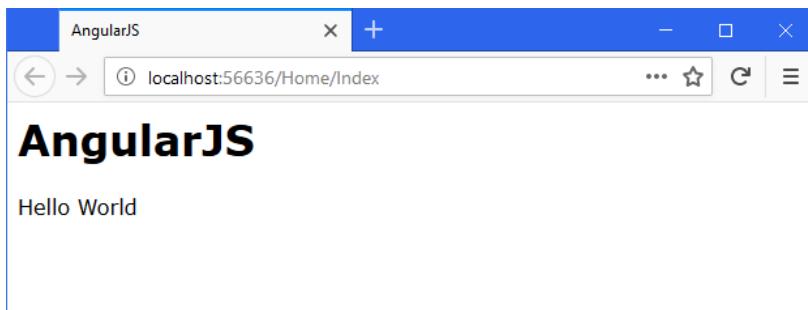
```
<html ng-app="mymodule">
<head>
  <title>AngularJS</title>
  <script src="~/node_modules/angular/angular.js"></script>
  <script>
    var app = angular.module("mymodule", []);
    app.controller("mycontroller", function ($scope)
    {
      $scope.message = "Hello World";
    })
  </script>
</head>
<body>
  <h1>AngularJS</h1>
  <div ng-controller="mycontroller">
    {{message}}
  </div>
</body>
```

</html>

#### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



## Angular 5 Integration

- Angular 5 is a "JavaScript Framework", which is used to create data bindings in web pages, using Component pattern.
- We will see how to integrate AngularJS 5" in this example.

#### How to Work with Angular 5

##### Import Angular 5 in package.json

```
{
  "name": "mypackage",
  "version": "1.0.0",
  "description": "this is my package",
  "license": "ISC",
  "repository": "none",
  "dependencies": {
    "@angular/core": "latest",
    "@angular/common": "latest",
    "@angular/platform-browser": "latest",
    "@angular/compiler": "latest",
    "@angular/platform-browser-dynamic": "latest",
    "systemjs": "latest",
    "core-js": "latest",
    "rxjs": "latest",
    "zone.js": "latest"
  }
}
```

## Angular 5 - Example

#### Creating Project

- Open Visual Studio 2017.
- Go to "File" - "New" - "Project".
- Select ".NET Framework 4.7".
- Select "Visual C#".
- Select "ASP.NET Web Application (.NET Framework)".
- Type the project name "Angular5IntegrationExample".
- Type the location as "C:\Mvc".
- Type the solution name as "Angular5IntegrationExample".

- Click on OK.
- Select "Empty".
- Check the checkbox "MVC".
- Uncheck the checkbox "Enable Docker support".
- Uncheck the checkbox "Add unit tests".
- Click on OK.

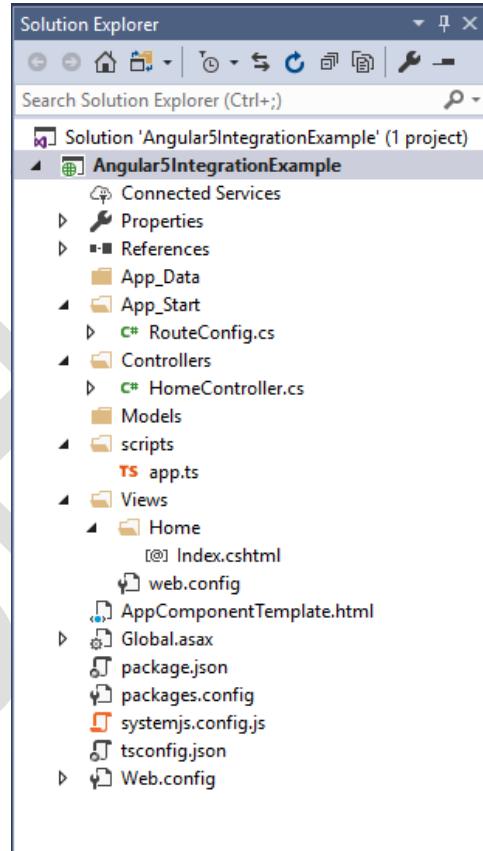
### **Creating package.json**

- Right click on the project (Angular5IntegrationExample) and click on "Add" - "New Item" - "Web" - "Npm Configuration File". Type the filename as "package.json" (default) and click on "Add".

#### **Code for "package.json"**

```
{
  "name": "mypackage",
  "version": "1.0.0",
  "description": "this is my package",
  "license": "ISC",
  "repository": "none",
  "dependencies": {
    "@angular/core": "latest",
    "@angular/common": "latest",
    "@angular/platform-browser": "latest",
    "@angular/compiler": "latest",
    "@angular/platform-browser-dynamic": "latest",
    "systemjs": "latest",
    "core-js": "latest",
    "rxjs": "latest",
    "zone.js": "latest"
  }
}
```

- To install the specified packages (Ex: angular), right click on "package.json" file and click on "Restore Packages".
- To see the installed packages, go to "Project" menu - "Show All Files".
- It shows "node\_modules" folder with the installed packages.



### **Creating tsconfig.json**

- Right click on the project (Angular5IntegrationExample) and click on "Add" - "New Item" - "Web" - "JSON File". Type the filename as "tsconfig.json" and click on "Add".

#### **Code for "tsconfig.json"**

```
{
  "compilerOptions": {
    "target": "es5",
    "sourceMap": false,
    "experimentalDecorators": true,
    "lib": ["es2015", "dom"]
  }
}
```

### **Creating systemjs.config.js**

- Right click on the project (Angular5IntegrationExample) and click on "Add" - "New Item" - "Web" - "JavaScript File". Type the filename as "systemjs.config.js" (default) and click on "Add".

#### **Code for "systemjs.config.js"**

```
SystemJS.config(
```

```
{
  baseURL: "/",
  map: {
    app: "scripts",
    "@angular/core": "node_modules/@angular/core/bundles/core.umd.js",
    "@angular/common": "node_modules/@angular/common/bundles/common.umd.js",
    "@angular/compiler": "node_modules/@angular/compiler/bundles/compiler.umd.js",
    "@angular/platform-browser": "node_modules/@angular/platform-browser/bundles/platform-browser.umd.js",
    "@angular/platform-browser-dynamic": "node_modules/@angular/platform-browser-dynamic/bundles/platform-browser-dynamic.umd.js",
    "rxjs": "node_modules/rxjs"
  },
  packages: {
    app: { main: "./app.js" },
    rxjs: {}
  }
};
```

**Creating AppComponentTemplate.html**

- Right click on the project (Angular5IntegrationExample) and click on "Add" - "New Item" - "Web" - "HTML File". Type the filename as "AppComponentTemplate.html" and click on "Add".

**Code for "AppComponentTemplate.html"**

```
<h1>Hello World</h1>
```

**Creating app.ts**

- Right click on the project (Angular5IntegrationExample) and click on "Add" - "New Folder". Type the folder name as "scripts" and press Enter. Right click on "scripts" folder and click on "Add" - "New Item" - "Web" - "TypeScript File". Type the filename as "app.ts" and click on "Add".

**Code for "app.ts"**

```
import { Component, NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

@Component({ selector: "app", templateUrl: "../AppComponentTemplate.html" })
class AppComponent
{
}

@NgModule({ declarations: [AppComponent], imports: [BrowserModule], bootstrap: [AppComponent] })
class AppModule
{
}
platformBrowserDynamic().bootstrapModule(AppModule);
```

**Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

**Code for "HomeController.cs"**

```
using System;
using System.Web.Mvc;

namespace Angular5IntegrationExample.Controllers
{
  public class HomeController : Controller
  {
    public ActionResult Index()
    {
```

```

        return View();
    }
}
}

```

#### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### Code for "Views\Home\Index.cshtml"

```

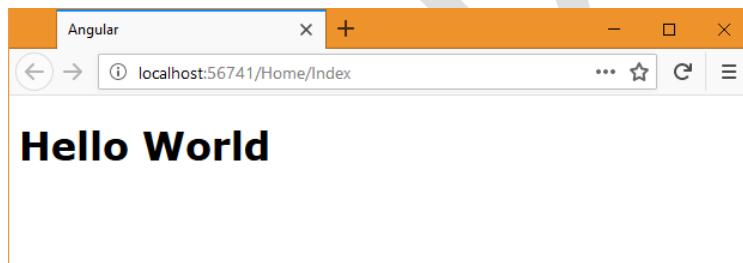
<html>
<head>
    <title>Angular</title>
    <script src="/node_modules/core-js/client/shim.js"></script>
    <script src="/node_modules/zone.js/dist/zone.js"></script>
    <script src="/node_modules/systemjs/dist/system.js"></script>
    <script src="/systemjs.config.js"></script>
    <script>
        System.import("scripts");
    </script>
</head>
<body>
    <app></app>
</body>
</html>

```

#### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



## Gulp Integration

- Gulp is a Task Runner, which minifies / combines javascript files and css files.

#### How to Work with Gulp

##### Import Gulp in package.json

```

{
  "version": "1.0.0",
  "name": "asp.net",
  "private": true,
  "devDependencies":
  {
    "gulp": "latest",
    "gulp-minify-css": "latest",
    "gulp-uglify": "latest",
    "gulp-concat": "latest",
    "del": "latest"
  }
}

```

**JavaScript Minification**

```
var gulp = require("gulp")
,uglify = require("gulp-uglify")
,concat = require("gulp-concat")
,del = require("del");

gulp.task("cleanjs", function ()
{
  return del(["./Scripts/min"]);
});

gulp.task("JavaScript", ["cleanjs"], function ()
{
  gulp.src("./Scripts/**/*.js")
    .pipe(uglify())
    .pipe(gulp.dest("./Scripts/min"))
    .pipe(concat("combined.js"))
    .pipe(gulp.dest("./"));
});
```

**CSS Minification**

```
var gulp = require("gulp")
,uglify = require("gulp-uglify")
,minifyCss = require("gulp-minify-css")
,concat = require("gulp-concat")
,del = require("del");

gulp.task("cleancss", function ()
{
  return del(["./Styles/min"]);
});

gulp.task("CSS", ["cleancss"], function ()
{
  gulp.src("./Styles/**/*.css")
    .pipe(minifyCss())
    .pipe(gulp.dest("./Styles/min"))
    .pipe(concat("combined.css"))
    .pipe(gulp.dest("./"));
});
```

**Gulp - Example****Creating Project**

- Open Visual Studio 2017.
- Go to “File” - “New” - “Project”.
- Select “.NET Framework 4.7”.
- Select “Visual C#”.
- Select “ASP.NET Web Application (.NET Framework)”.
- Type the project name “GulpIntegrationExample”.
- Type the location as “C:\Mvc”.
- Type the solution name as “GulpIntegrationExample”.
- Click on OK.
- Select “Empty”.
- Check the checkbox “MVC”.
- Uncheck the checkbox “Enable Docker support”.
- Uncheck the checkbox “Add unit tests”.
- Click on OK.

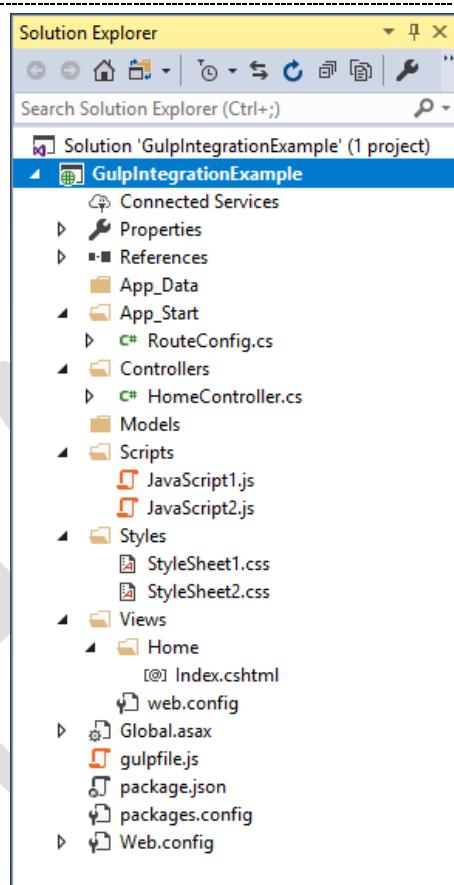
### **Creating package.json**

- Right click on the project (GulpIntegrationExample) and click on "Add" - "New Item" - "Web" - "Npm Configuration File". Type the filename as "package.json" (default) and click on "Add".

### **Code for "package.json"**

```
{
  "version": "1.0.0",
  "name": "asp.net",
  "private": true,
  "devDependencies":
  {
    "gulp": "latest",
    "gulp-minify-css": "latest",
    "gulp-uglify": "latest",
    "gulp-concat": "latest",
    "del": "latest"
  }
}
```

- To install the specified packages (Ex: gulp), right click on "package.json" file and click on "Restore Packages".
- To see the installed packages, go to "Project" menu - "Show All Files".
- It shows "node\_modules" folder with the installed packages.



### **Creating Scripts folder**

- Right click on the project (GulpIntegrationExample) and click on "Add" - "New Folder". Type the folder name as "Scripts" and press Enter. Right click on "Scripts" folder and click on "Add" - "New Item" - "Web" - "JavaScript File". Type the filename as "JavaScript1.js" and click on "Add".

### **Code for "Scripts\JavaScript1.js"**

```
function fun1()
```

- Right click on "Scripts" folder and click on "Add" - "New Item" - "Web" - "JavaScript File". Type the filename as "JavaScript2.js" and click on "Add".

### **Code for "Scripts\JavaScript2.js"**

```
function fun2()
```

### **Creating Styles folder**

- Right click on the project (GulpIntegrationExample) and click on "Add" - "New Folder". Type the folder name as "Styles" and press Enter. Right click on "Styles" folder and click on "Add" - "New Item" - "Web" - "Style Sheet". Type the filename as "StyleSheet1.css" and click on "Add".

### **Code for "Styles\StyleSheet1.css"**

```
body
{
  font-family: "Tahoma";
}
```

- Right click on "Styles" folder and click on "Add" - "New Item" - "Web" - "Style Sheet". Type the filename as "StyleSheet2.css" and click on "Add".

### **Code for "Styles\StyleSheet2.css"**

```
body
{
  font-size: 20px;
```

```
}
```

#### Creating gulpfile.js

- Right click on the project (GulpIntegrationExample) and click on "Add" - "New Item" - "Web" - "Gulp Configuration File". Type the filename as "gulpfile.js" and click on "Add".

#### Code for "gulpfile.js"

```
var gulp = require("gulp")
,uglify = require("gulp-uglify")
,minifyCss = require("gulp-minify-css")
,concat = require("gulp-concat")
,del = require("del");

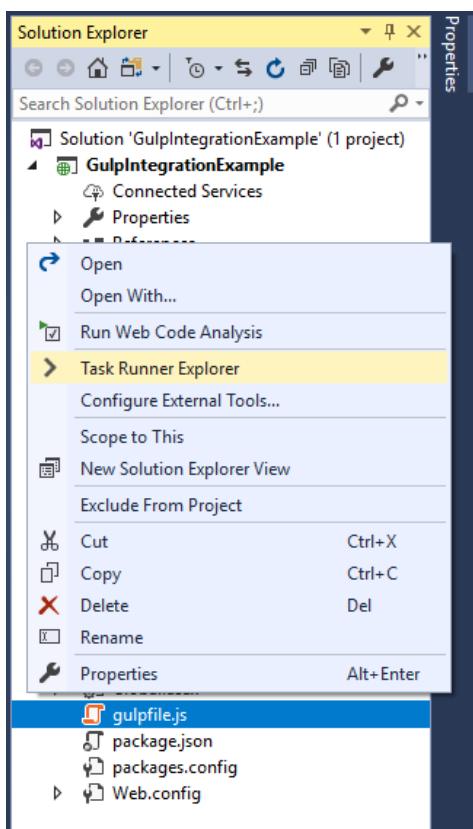
gulp.task("cleancss",function()
{
  return del(["./Styles/min"]);
});

gulp.task("cleanjs",function()
{
  return del(["./Scripts/min"]);
});

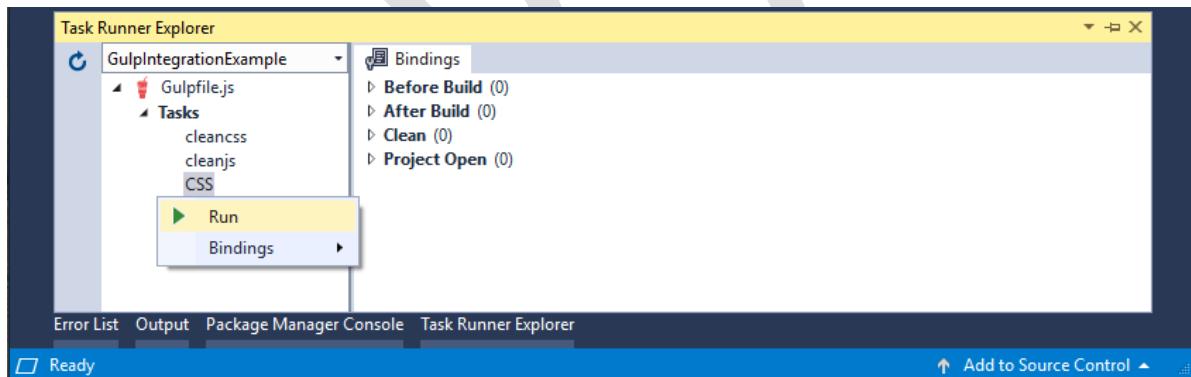
gulp.task("JavaScript",["cleanjs"],function()
{
  gulp.src("./Scripts/**/*.js")
    .pipe(uglify()).pipe(gulp.dest("./Scripts/min"))
    .pipe(concat("combined.js")).pipe(gulp.dest("./"));
});

gulp.task("CSS",["cleancss"],function()
{
  gulp.src("./Styles/**/*.css")
    .pipe(minifyCss()).pipe(gulp.dest("./Styles/min"))
    .pipe(concat("combined.css")).pipe(gulp.dest("./"))
});
```

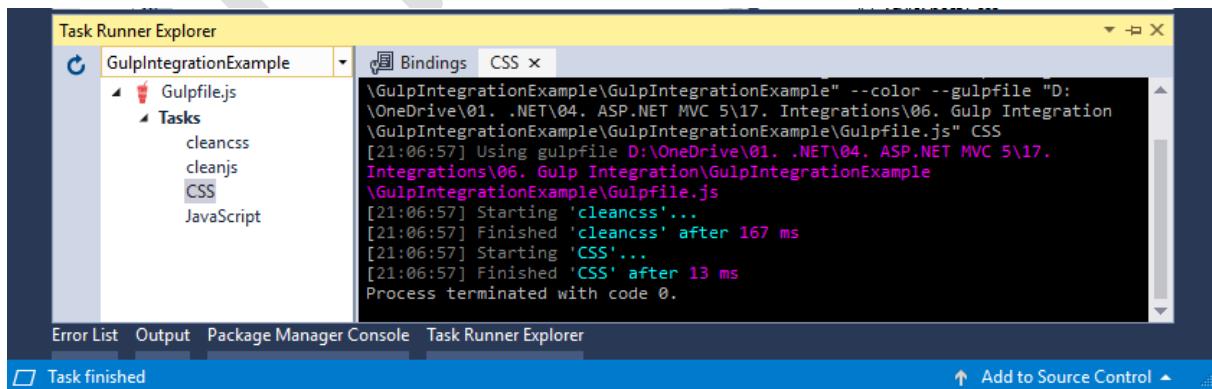
- Right click on "gulpfile.js" and click on "Task Runner Explorer".



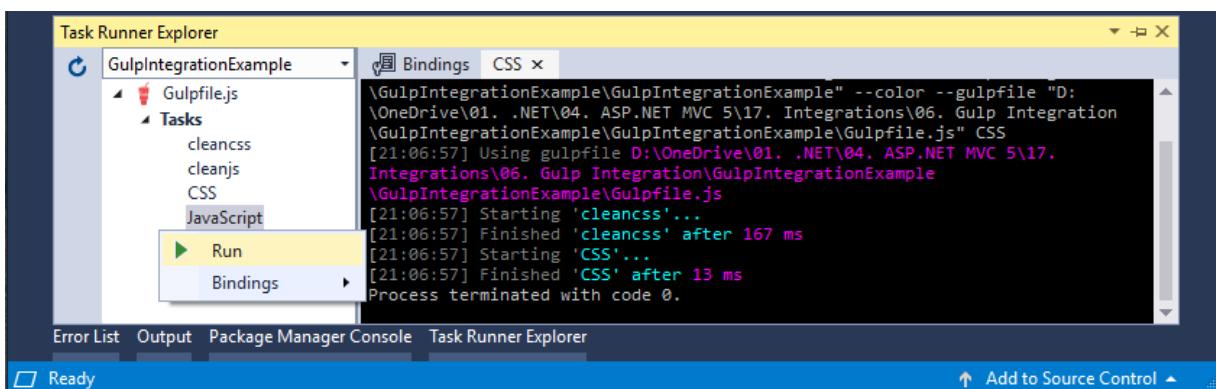
- Right click on "Tasks" - "CSS" and click on "Run".



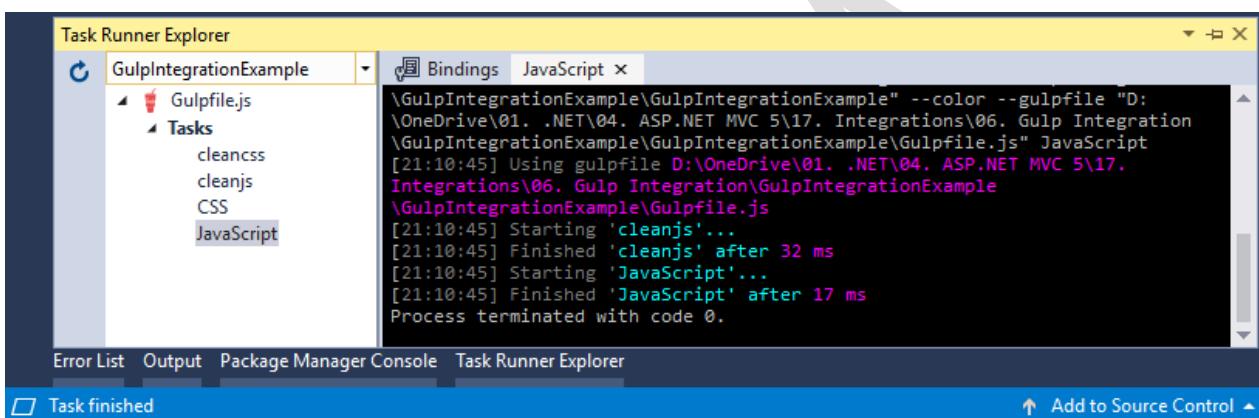
- It automatically generates "combined.css".



- Next, Right click on "Tasks" - "JavaScript" and click on "Run".



- It automatically generates "combined.js".



- Solution Explorer after generating files:

### **Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### **Code for "HomeController.cs"**

```

using System;
using System.Web.Mvc;

namespace GulpIntegrationExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
  
```

### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

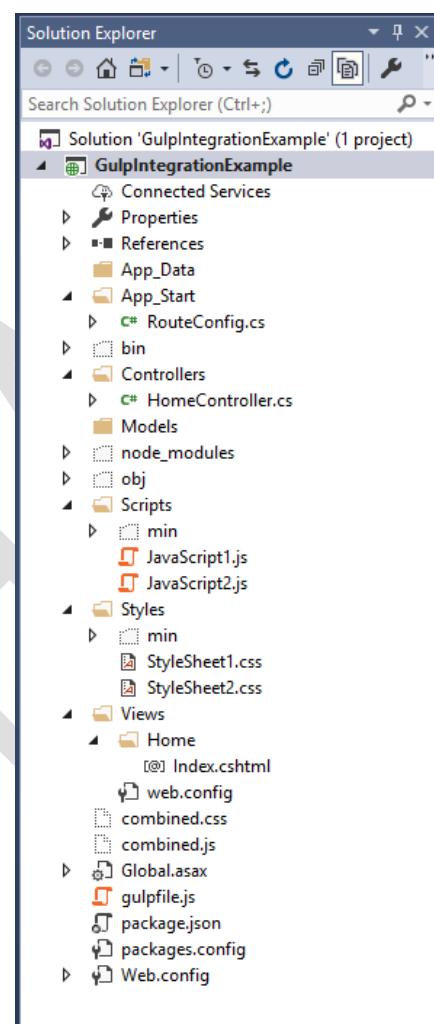
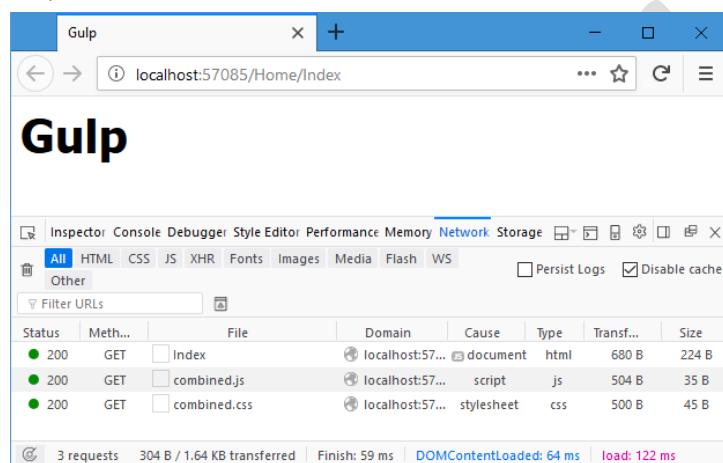
### Code for "Views\Home\Index.cshtml"

```
<html>
  <head>
    <title>Gulp</title>
    <script src="~/combined.js"></script>
    <link href="~/combined.css" rel="stylesheet">
  </head>
  <body>
    <h1>Gulp</h1>
  </body>
</html>
```

### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



## Grunt Integration

- Grunt is a Task Runner, which minifies / combines javascript files and css files.

### How to Work with Grunt

#### Import Gulp in package.json

```
{
  "version": "1.0.0",
  "name": "asp.net",
  "private": true,
  "devDependencies": {
    "grunt": "0.4.5",
    "grunt-contrib-cssmin": "latest",
    "grunt-contrib-uglify": "latest"
  }
}
```

#### JavaScript / CSS Minification

```
module.exports = function(grunt) {
}
```

```
grunt.initConfig({
  uglify: { build: { src: ["Scripts/**/*.js"], dest: "combined.js" } },
  cssmin: { build: { src: "Styles/**/*.css", dest: "combined.css" } }
});
grunt.loadNpmTasks("grunt-contrib-uglify");
grunt.loadNpmTasks("grunt-contrib-cssmin");
grunt.registerTask("JavaScript", ["uglify"]);
grunt.registerTask("CSS", ["cssmin"]);
});
```

### Grunt - Example

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "GruntIntegrationExample". Type the location as "C:\Mvc". Type the solution name as "GruntIntegrationExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

#### Creating package.json

- Right click on the project (GruntIntegrationExample) and click on "Add" - "New Item" - "Web" - "Npm Configuration File". Type the filename as "package.json" (default) and click on "Add".

#### Code for "package.json"

```
{
  "version": "1.0.0",
  "name": "asp.net",
  "private": true,
  "devDependencies": {
    "grunt": "0.4.5",
    "grunt-contrib-cssmin": "latest",
    "grunt-contrib-uglify": "latest"
  }
}
```

- To install the specified packages (Ex: grunt), right click on "package.json" file and click on "Restore Packages". To see the installed packages, go to "Project" menu - "Show All Files". It shows "node\_modules" folder with the installed packages.

#### Creating Scripts folder

- Right click on the project (GruntIntegrationExample) and click on "Add" - "New Folder". Type the folder name as "Scripts" and press Enter. Right click on "Scripts" folder and click on "Add" - "New Item" - "Web" - "JavaScript File". Type the filename as "JavaScript1.js" and click on "Add".

#### Code for "Scripts\JavaScript1.js"

```
function fun1()
{
}
```

- Right click on "Scripts" folder and click on "Add" - "New Item" - "Web" - "JavaScript File". Type the filename as "JavaScript2.js" and click on "Add".

#### Code for "Scripts\JavaScript2.js"

```
function fun2()
{
}
```

#### Creating Styles folder

- Right click on the project (GruntIntegrationExample) and click on "Add" - "New Folder". Type the folder name as "Styles" and press Enter. Right click on "Styles" folder and click on "Add" - "New Item" - "Web" - "Style Sheet". Type the filename as "StyleSheet1.css" and click on "Add".

#### Code for "Styles\StyleSheet1.css"

```
body
```

```
{
    font-family: "Tahoma";
}
```

- Right click on "Styles" folder and click on "Add" - "New Item" - "Web" - "Style Sheet". Type the filename as "StyleSheet2.css" and click on "Add".

### Code for "Styles\StyleSheet2.css"

```
body
{
    font-size: 20px;
}
```

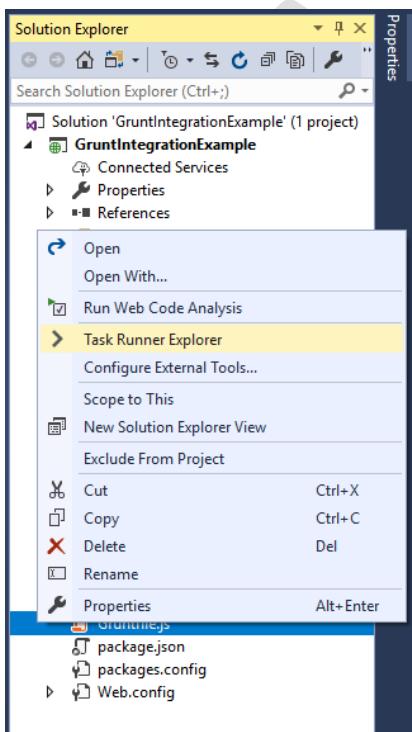
### Creating Gruntfile.js

- Right click on the project (GruntIntegrationExample) and click on "Add" - "New Item" - "Web" - "Grunt Configuration File". Type the filename as "Gruntfile.js" and click on "Add".

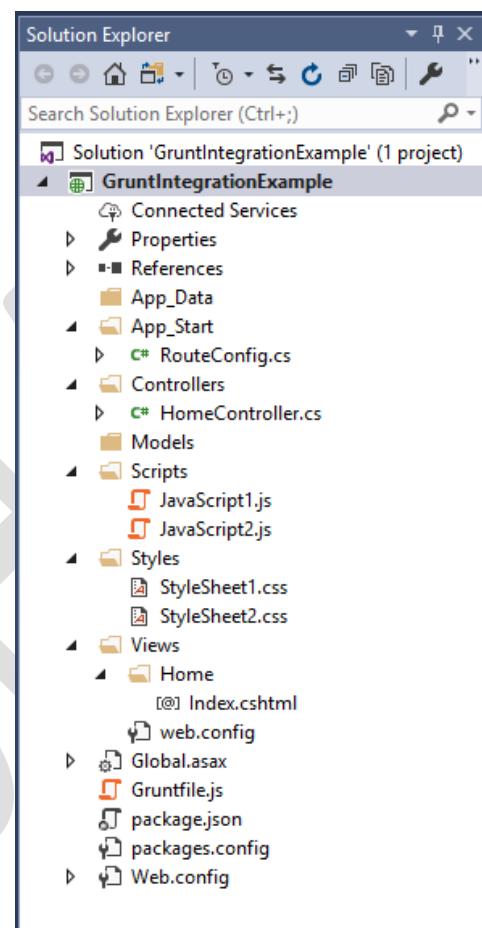
### Code for "Gruntfile.js"

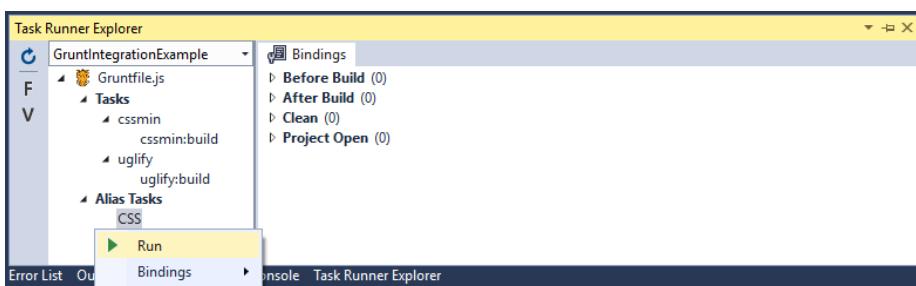
```
module.exports = function (grunt)
{
    grunt.initConfig({
        uglify: { build: { src: ["Scripts/**/*.js"], dest: "combined.js" } },
        cssmin: { build: { src: "Styles/**/*.css", dest: "combined.css" } }
    });
    grunt.loadNpmTasks("grunt-contrib-uglify");
    grunt.loadNpmTasks("grunt-contrib-cssmin");
    grunt.registerTask("JavaScript", ["uglify"]);
    grunt.registerTask("CSS", ["cssmin"]);
};
```

- Right click on "Gruntfile.js" and click on "Task Runner Explorer".

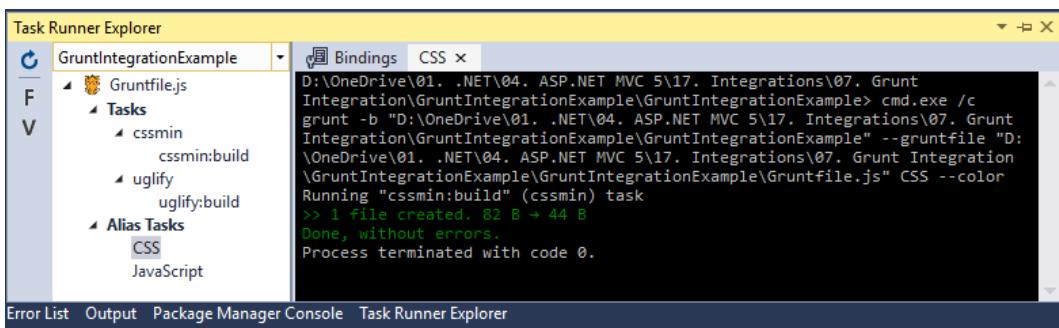


- Right click on "Alias Tasks" - "CSS" and click on "Run".

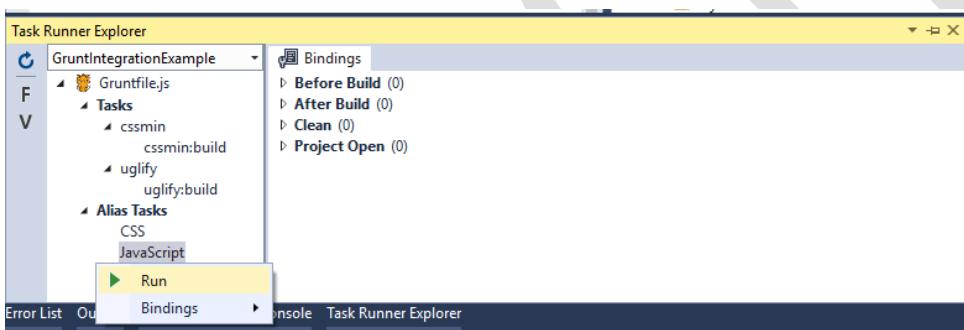




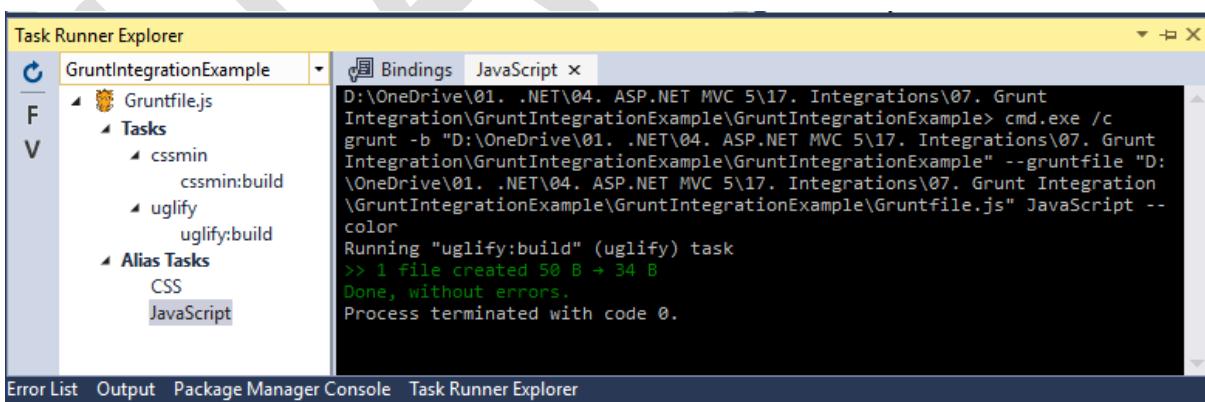
- It automatically generates "combined.css".



- Next, Right click on "Alias Tasks" - "JavaScript" and click on "Run".



- It automatically generates "combined.js".



- Solution Explorer after generating files:

### **Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

**Code for "HomeController.cs"**

```
using System;
using System.Web.Mvc;

namespace GruntIntegrationExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

**Creating Index.cshtml**

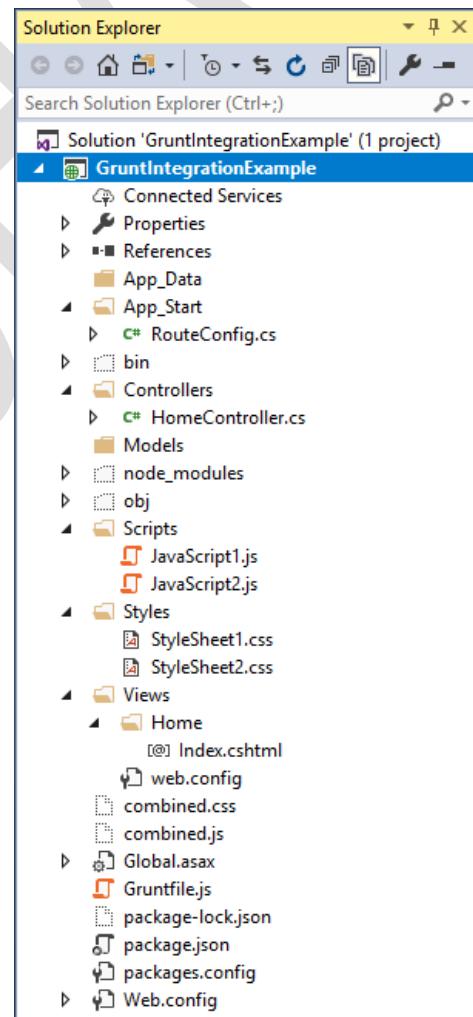
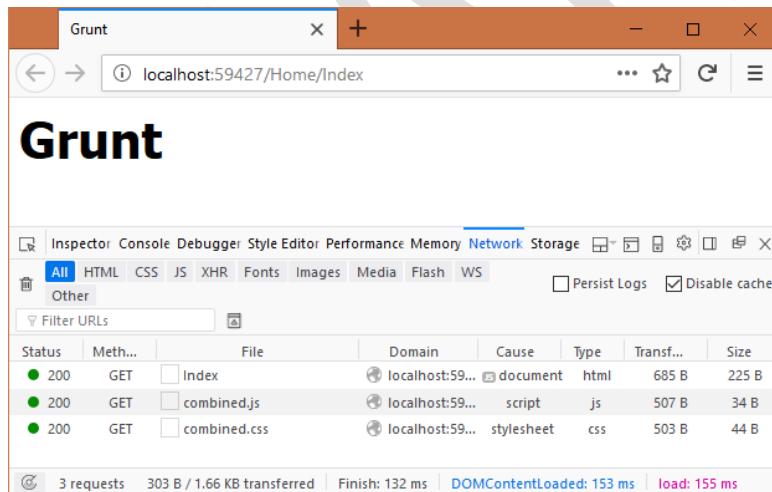
- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Index.cshtml"**

```
<html>
<head>
<title>Grunt</title>
<script src="~/combined.js"></script>
<link href="~/combined.css" rel="stylesheet">
</head>
<body>
<h1>Grunt</h1>
</body>
</html>
```

**Running the application**

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

**Output:****Bower Integration**

- Bower is a Package Repository, which contains many packages, similar to "npm".

**How to Work with Bower**

### Import Packages in bower.json

```
{  
  "name": "asp.net",  
  "private": true,  
  "dependencies":  
  {  
    "packagename": "version"  
  }  
}
```

## Bower - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "BowerIntegrationExample". Type the location as "C:\Mvc". Type the solution name as "BowerIntegrationExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating bower.json

- Right click on the project (BowerIntegrationExample) and click on "Add" - "New Item" - "Web" - "Bower Configuration File". Type the filename as "bower.json" (default) and click on "Add".

### Code for "package.json"

```
{  
  "name": "asp.net",  
  "private": true,  
  "dependencies":  
  {  
    "jquery": "latest"  
  }  
}
```

- To install the specified packages (Ex: jquery), right click on "bowser.json" file and click on "Restore Packages".
- To see the installed packages, go to "Project" menu - "Show All Files".
- It shows "bower\_components" folder with the installed packages.

### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```
using System;  
using System.Web.Mvc;  
  
namespace BowerIntegrationExample.Controllers  
{  
  public class HomeController : Controller  
  {
```

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

### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

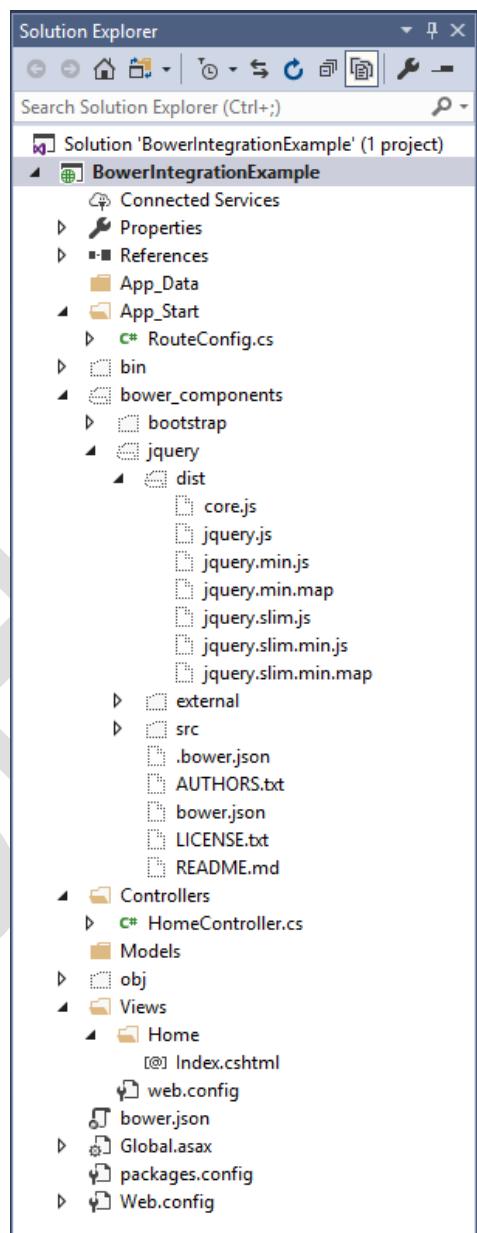
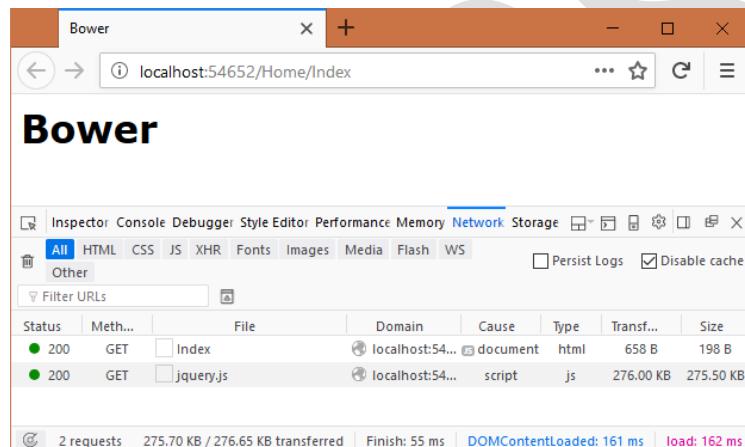
### Code for "Views\Home\Index.cshtml"

```
<html>
<head>
<title>Bower</title>
<script src="~/bower_components/jquery/dist/jquery.js"></script>
</head>
<body>
<h1>Bower</h1>
</body>
</html>
```

### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

#### Output:



## WCF Integration

- WCF stands for "Windows Communication Foundation", which is used to create services that can be called in any type of .net applications using any protocol, and also from non-.net applications using "HTTP" protocol.
- Service contains re-usable code. In large applications, it acts as a mediator between business logic and presentation logic.

### How to Work with WCF

#### Create Service Contract

```
using System;
using System.ServiceModel;

namespace namespacename
{
    [ServiceContract]
    public interface interfacename
    {
```

```

    [OperationContract]
    returntype methodname(datatype argument, ...);
}
}

```

### Create Service

```

using System;

namespace namespace
{
    public class Serviceclassname : interfacename
    {
        public returntype Methodname(datatype argument, ...)
        {
            code here
        }
    }
}

```

### Add Service Reference

- Right click on the project and click on "Add" - "Service Reference". Enter the URL of the service and click on OK button.

### Call the Service

```

ServiceReference1.Service1Client client = new ServiceReference1.Service1Client();
client.Methodname(arg1, arg2, ...);

```

## WCF - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "WcfIntegrationExample". Type the location as "C:\Mvc". Type the solution name as "WcfIntegrationExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating WCF Service

- Right click on the project (WcfIntegrationExample) and click on "Add" - "New Folder". Type the folder name as "Services". Right click on "Services" and click on "Add" - "New Item" - "Web" - "WCF Service". Type the filename as "Service1.svc" and click on "Add".

### Code for "Services\IService1.cs"

```

using System;
using System.ServiceModel;

namespace WCFIntegrationExample.Services
{
    [ServiceContract]
    public interface IService1
    {
        [OperationContract]
        int Add(int a, int b);
    }
}

```

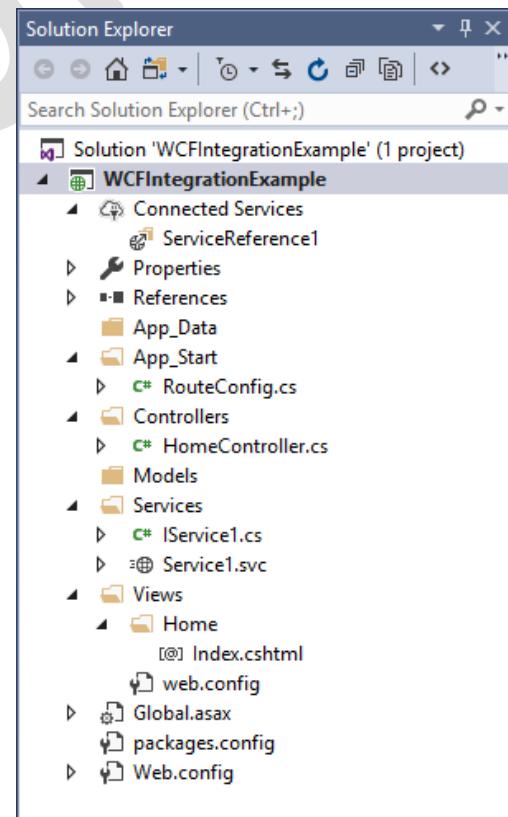
### Code for "Services\Service1.svc"

```

using System;

namespace WCFIntegrationExample.Services

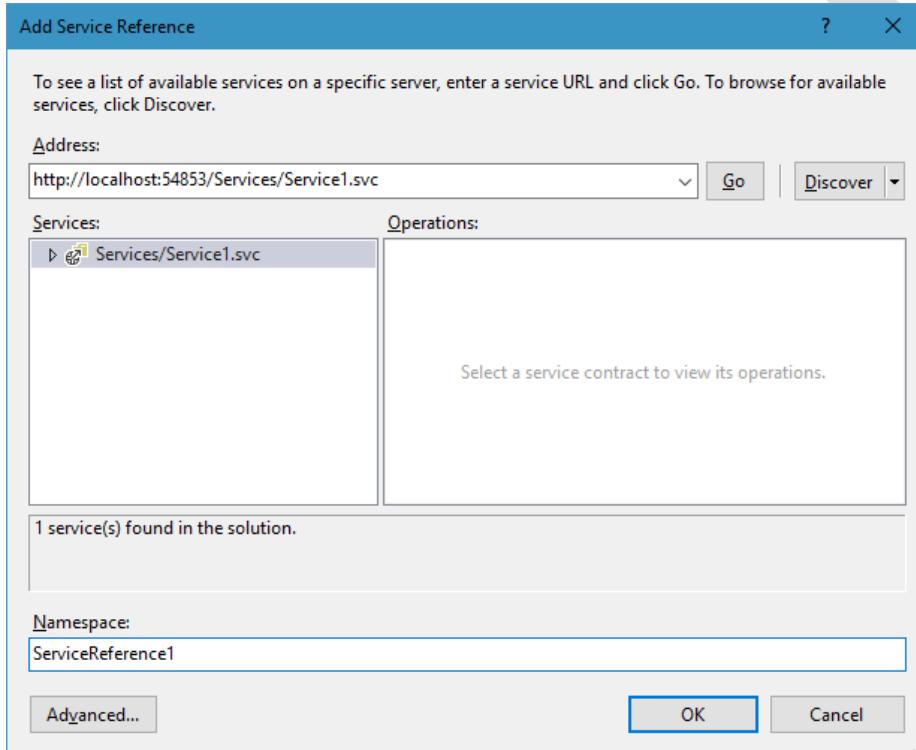
```



```
{
    public class Service1 : IService1
    {
        public int Add(int a, int b)
        {
            int c = a + b;
            return c;
        }
    }
}
```

### **Adding Service Reference**

- Right click on the project (WcfIntegrationExample) and click on "Add" - "Service Reference". Click on "Discover". It shows the url as "http://localhost:54853/Services/Service1.svc". Enter the namespace name as "ServiceReference1". Click on OK.



### **Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### **Code for "HomeController.cs"**

```
using System;
using System.Web.Mvc;

namespace WCFIntegrationExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            ServiceReference1.Service1Client client = new ServiceReference1.Service1Client();
            int result = client.Add(10, 20);
            ViewBag.result = result;
            return View();
        }
    }
}
```

```
}
```

### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

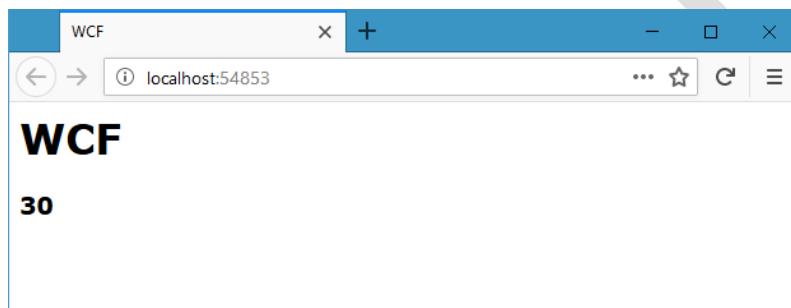
### Code for "Views\Home\Index.cshtml"

```
<html>
  <head>
    <title>WCF</title>
  </head>
  <body>
    <h1>WCF</h1>
    <h3>@ViewBag.result</h3>
  </body>
</html>
```

### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



## AutoMapper Integration

- AutoMapper is one of the most popular packages in MVC projects.
- It is used to convert a model object to another model object. That means it copies data from one model class's object to another model class's object, where the property names are matching. It skips the properties that are not matching.
- It is useful in realtime applications, to migrate data from "Domain Model" to "View Model" and vice versa.

### How to Work with AutoMapper

#### Install AutoMapper

```
install-package AutoMapper;
```

#### Import AutoMapper Namespace

```
using AutoMapper;
```

#### Initialize AutoMapper

```
var config = new MapperConfiguration(cfg => { cfg.CreateMap<Modelclass1, Modelclass2>(); });
var mapper = config.CreateMapper();
```

#### Convert data from Modelclass1 to Modelclass2

```
Modelclass2 destination = mapper.Map<Modelclass1, Modelclass2>(source);
```

## AutoMapper - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "AutoMapperExample". Type the location as "C:\Mvc". Type the solution name as "AutoMapperExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Installing Packages

- Open Solution Explorer.
- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Type the following command in "Package Manager Console" and press Enter.  
`install-package AutoMapper`

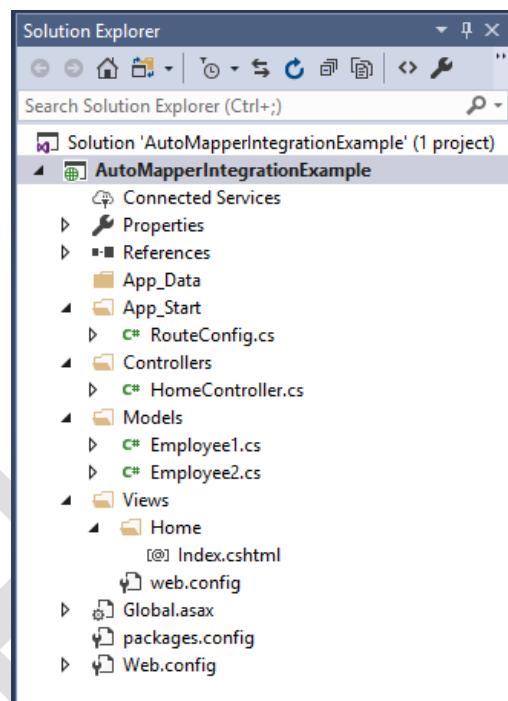
### Creating Employee1.cs

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee1.cs". Click on "Add".

### Code for "Employee1.cs"

```
using System;

namespace AutoMapperIntegrationExample.Models
{
    public class Employee1
    {
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public double Salary { get; set; }
    }
}
```



### Creating Employee2.cs

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee2.cs". Click on "Add".

### Code for "Employee2.cs"

```
using System;

namespace AutoMapperIntegrationExample.Models
{
    public class Employee2
    {
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public string Email { get; set; }
    }
}
```

### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;
using AutoMapperIntegrationExample.Models;
using AutoMapper;

namespace AutoMapperIntegrationExample.Controllers
```

```
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            Employee1 emp1 = new Employee1() { EmpID=101, EmpName = "Scott", Salary = 8000 };
            var config = new MapperConfiguration(cfg => { cfg.CreateMap<Employee1, Employee2>(); });
            var mapper = config.CreateMapper();
            Employee2 emp2 = mapper.Map<Employee1, Employee2>(emp1);
            ViewBag.emp2 = emp2;
            return View();
        }
    }
}
```

**Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

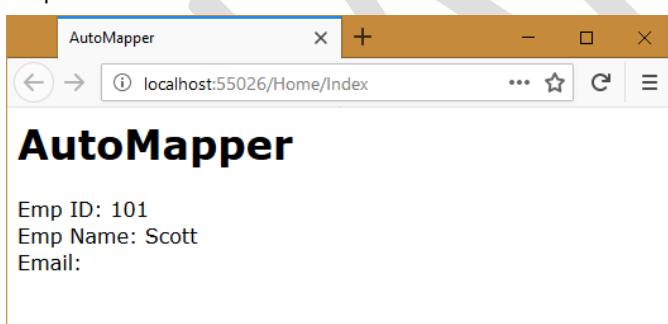
**Code for "Views\Home\Index.cshtml"**

```
<html>
    <head>
        <title>AutoMapper</title>
    </head>
    <body>
        <h1>AutoMapper</h1>
        Emp ID: @ViewBag.emp2.EmpID<br />
        Emp Name: @ViewBag.emp2.EmpName<br />
        Email: @ViewBag.emp2.Email<br />
    </body>
</html>
```

**Running the application**

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:

**WORKING WITH FORMS****Forms**

- Form is a collection of form elements such as textboxes, checkboxes, radio buttons, dropdownlists etc. Form can be submitted to controller; then the controller can receive values of the form and do some manipulations

**How to Work with Form****Create Form**

```
<form action="/controllername/actionname" method="get/post">
    <input type="text" name="fieldname1"/>
    <input type="text" name="fieldname2"/>
```

```
...
<input type="submit" value="Submit" />
</form>
```

### Receive Form Parameters in Controller

```
public class Controllerclassname : Controller
{
    [HttpGet] or [HttpPost]
    public returnType Methodname()
    {
        Request.Form["parameter name"]
    }
}
```

## Forms - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "FormsExample". Type the location as "C:\Mvc". Type the solution name as "FormsExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating HomeController.cs

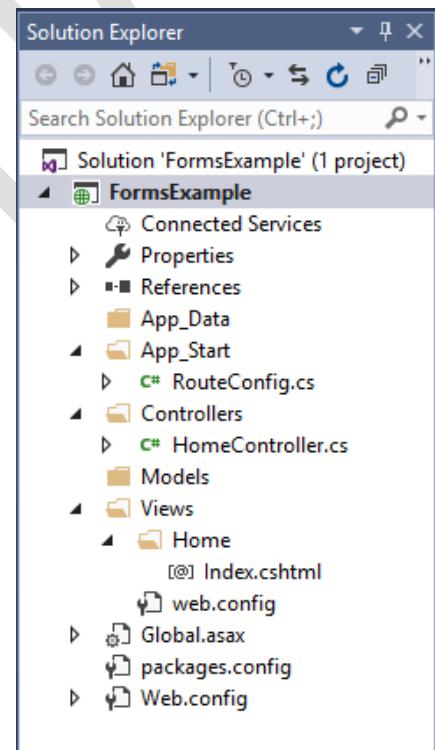
- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;

namespace FormsExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        [HttpPost]
        public ActionResult ProcessData()
        {
            int a = Convert.ToInt32(Request.Params["EmpID"]);
            string b = Request.Params["EmpName"];
            double c = Convert.ToDouble(Request.Params["Salary"]);
            string s = "Emp ID: " + a + "<br>Emp Name: " + b + "<br>Salary: " + c;
            return Content(s);
        }
    }
}
```



### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### Code for "Views\Home\Index.cshtml"

```
<html>
<head>
    <title>Form</title>
</head>
```

```

<body>
    <h1>Form</h1>
    <form action="/Home/ProcessData" method="post">
        Emp ID: <input type="text" name="EmpID" /><br />
        Emp Name: <input type="text" name="EmpName" /><br />
        Salary: <input type="text" name="Salary" /><br />
        <input type="submit" value="Submit" />
    </form>
</body>
</html>

```

#### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:

Form

Emp ID:

Emp Name:

Salary:

Enter some details.

Form

Emp ID:

Emp Name:

Salary:

Click on "Submit".

localhost:55759/Home/ProcessData

Emp ID: 1  
Emp Name: abc  
Salary: 5000

#### Action Parameters

- Action method can receive form values as arguments.
- We can give any data type.

## How to Work with Action Parameters

### Create Form

```
<form action="/controllername/actionname" method="get/post">
<input type="text" name="fieldnamer1"/>
<input type="text" name="fieldname2"/>
...
<input type="submit" value="Submit" />
</form>
```

### Receive Action Parameters in Controller

```
public class Controllerclassname : Controller
{
    [HttpPost]
    public returntype Methodname(datatype fieldname1, datatype fieldname2, ...)
    {
        code here
    }
}
```

## Action Parameters - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "ActionParametersExample". Type the location as "C:\Mvc". Type the solution name as "ActionParametersExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating HomeController.cs

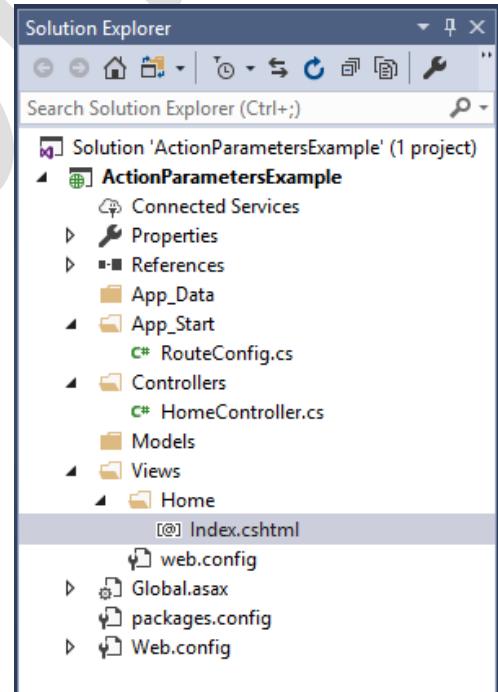
- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;

namespace ActionParametersExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        [HttpPost]
        public ActionResult ProcessData(int EmpID, string EmpName, double Salary)
        {
            string s = "Emp ID: " + EmpID + "<br>Emp Name: " + EmpName + "<br>Salary: " + Salary;
            return Content(s);
        }
    }
}
```



### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Index.cshtml"**

```

<html>
<head>
<title>Action Parameters</title>
</head>
<body>
<h1>Action Parameters</h1>
<form action="/Home/ProcessData" method="post">
Emp ID: <input type="text" name="EmpID" /><br />
Emp Name: <input type="text" name="EmpName" /><br />
Salary: <input type="text" name="Salary" /><br />
<input type="submit" value="Submit"/>
</form>
</body>
</html>

```

**Running the application**

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:

Action Parameters

Emp ID:

Emp Name:

Salary:

Enter some details.

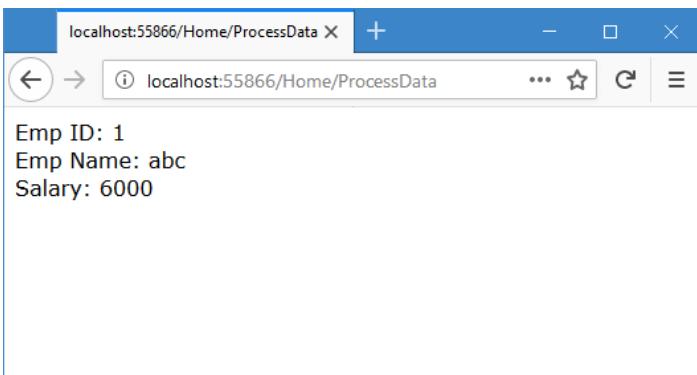
Action Parameters

Emp ID:

Emp Name:

Salary:

Click on "Submit".



### Automatic Model Binding

- "Automatic Model Binding" is one of the nice features of ASP.NET MVC.
- Process of Automatic Model Binding:
  - The user submits the form with a set of fields (parameters).
  - MVC identifies the appropriate route.
  - MVC identifies the appropriate controller.
  - MVC identifies the appropriate action.
  - MVC automatically creates an object for the model class and copies parameter values into respective properties of model object, where the names are matching. It skips the properties that names doesn't match.
  - The model object will be received as argument in the action method.

#### How to Work with Automatic Model Binding

##### Create Form

```
<form action="/controllername/actionname" method="get / post">
<input type="text" name="fieldname1" />
<input type="text" name="fieldname2" />
...
<input type="submit" value="Submit" />
</form>
```

##### Receive Model object in Controller

```
public class Controllerclassname : Controller
{
    [HttpPost]
    public returntype Methodname(Modelclassname argumentname)
    {
        code here
    }
}
```

### Automatic Model Binding - Example

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "AutomaticModelBindingExample". Type the location as "C:\Mvc". Type the solution name as "AutomaticModelBindingExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

#### Creating Employee.cs

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

### Code for "Employee.cs"

```
using System;

namespace AutomaticModelBindingExample.Models
{
    public class Employee
    {
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public double Salary { get; set; }
    }
}
```

### Creating HomeController.cs

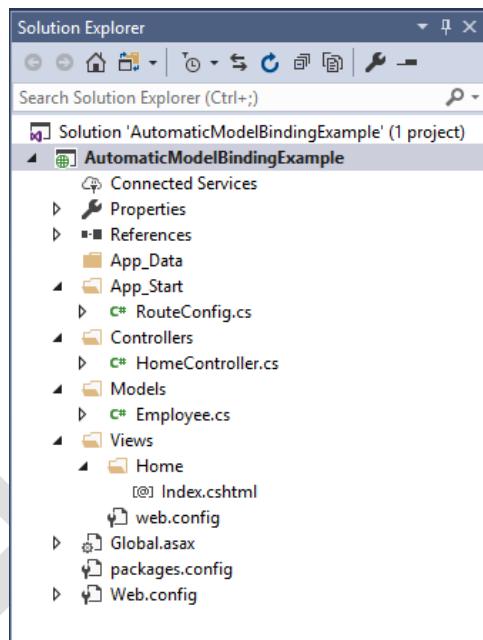
- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;
using AutomaticModelBindingExample.Models;

namespace AutomaticModelBindingExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        public ActionResult ProcessData(Employee emp)
        {
            string s = "Emp ID: " + emp.EmpID + "<br>Emp Name: " + emp.EmpName + "<br>Salary: " + emp.Salary;
            return Content(s);
        }
    }
}
```



### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### Code for "Views\Home\Index.cshtml"

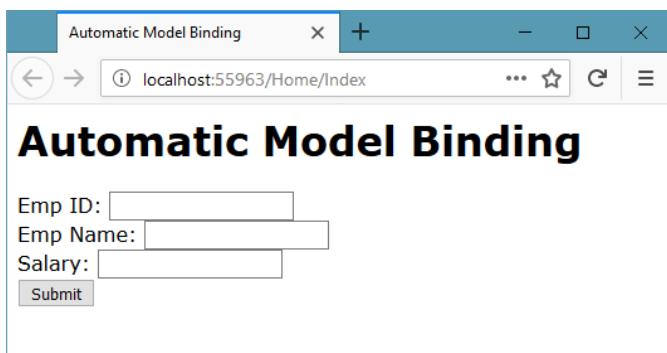
```
<html>
<head>
    <title>Automatic Model Binding</title>
</head>
<body>
    <h1>Automatic Model Binding</h1>
    <form action="/Home/ProcessData" method="post">
        Emp ID: <input type="text" name="EmpID" /><br />
        Emp Name: <input type="text" name="EmpName" /><br />
        Salary: <input type="text" name="Salary" /><br />
        <input type="submit" value="Submit" />
    </form>
</body>
</html>
```

### Running the application

- Press "F5" to run the application.

- Type "http://localhost:portnumber/Home/Index".

Output:



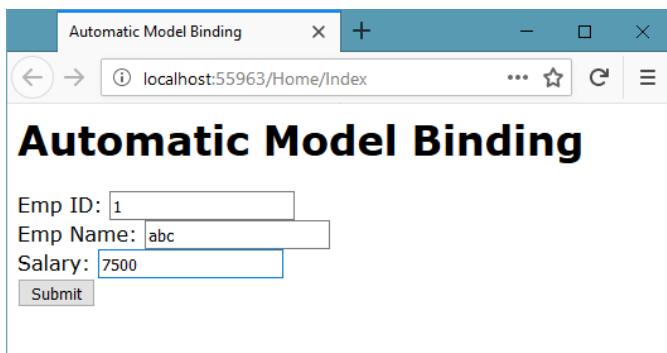
Automatic Model Binding

Emp ID:

Emp Name:

Salary:

Enter some details.



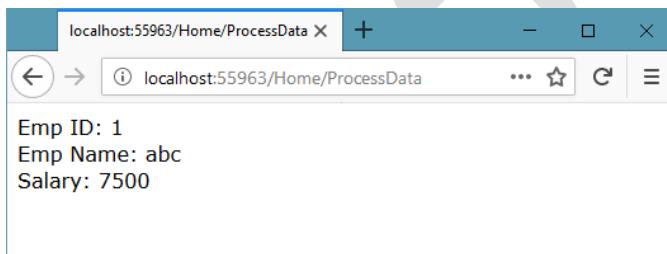
Automatic Model Binding

Emp ID:

Emp Name:

Salary:

Click on "Submit".



localhost:55963/Home/ProcessData

Emp ID: 1  
Emp Name: abc  
Salary: 7500

## STRONGLY TYPED VIEWS

### What is Strongly Typed View

- ViewBag does not support "IntelliSense". That is why "strongly typed views" are introduced. "Strongly-typed view" is a view that is bounded with a model class. Strongly typed view can receive a model object of the corresponding model class. Strongly typed views support intellisense.
- It is recommended to use "strongly typed views", if it is doing something with model. Ex: "Login.cshtml" is binded to "LoginModel". To make a view as "strongly-typed view", add the following statement in the view, at the top of the view.

#### Syntax of Strongly Typed View

```
@model modelclassname
```

#### "Model" property in strongly-typed view

- The "Model" property represents the model object that is passed by the controller.

Syntax: Model.propertyname

#### Passing Model Object from Controller to View

- To pass a model object to the strongly typed view, use the following syntax:

```
return View(model object);
```

## Strongly Typed Views - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "StronglyTypedViewsExample". Type the location as "C:\Mvc". Type the solution name as "StronglyTypedViewsExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

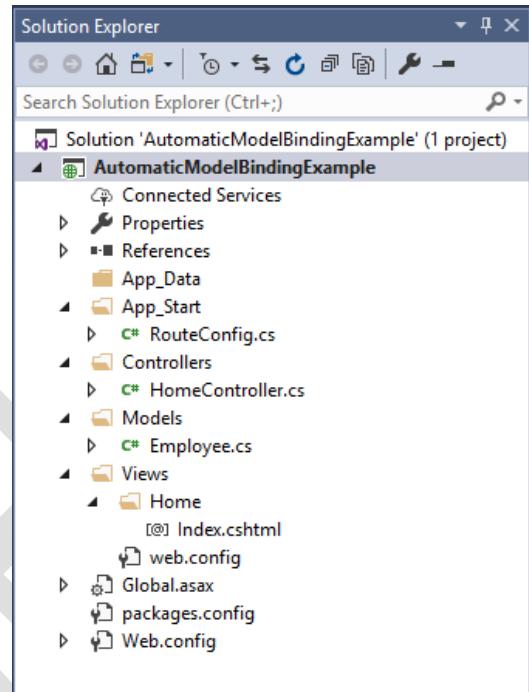
### Creating Employee.cs

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

### Code for "Employee.cs"

```
using System;
```

```
namespace StronglyTypedViewsExample.Models
{
    public class Employee
    {
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public double Salary { get; set; }
    }
}
```



### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```
using System;
using StronglyTypedViewsExample.Models;
using System.Web.Mvc;

namespace StronglyTypedViewsExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            Employee emp = new Employee() { EmpID = 101, EmpName = "Scott", Salary = 8000 };
            return View(emp);
        }
    }
}
```

### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### Code for "Views\Home\Index.cshtml"

```
@model StronglyTypedViewsExample.Models.Employee
<html>
<head>
    <title>Strongly Typed Views</title>
</head>
<body>
    <h1>Strongly Typed Views</h1>
    <p>Emp ID: @Model.EmpID</p>

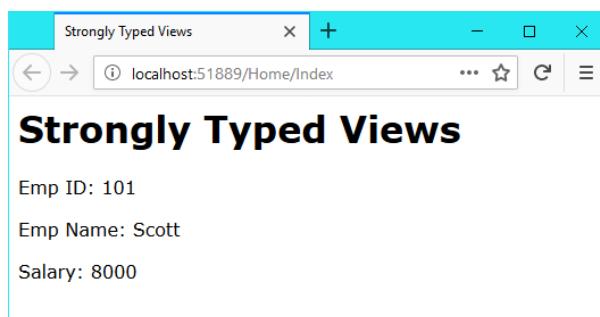
```

```
<p>Emp Name: @Model.EmpName</p>
<p>Salary: @Model.Salary</p>
</body>
</html>
```

#### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



## Strongly Typed Views With Collections

- Strongly Typed View can be binded with collection of model also. Then it can receive a collection of model objects from the controller.

#### Syntax of Strongly Typed View With Collections

```
@model List<modelclassname>
```

## Strongly Typed Views With Collections - Example

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "StronglyTypedViewsWithCollectionsExample". Type the location as "C:\Mvc". Type the solution name as "StronglyTypedViewsWithCollectionsExample". Click on OK.
- Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

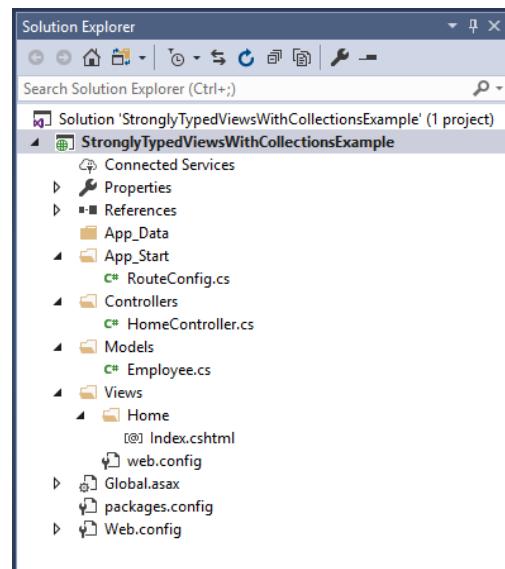
#### Creating Employee.cs

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

#### Code for "Employee.cs"

```
using System;

namespace StronglyTypedViewsWithCollectionsExample.Models
{
    public class Employee
    {
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public double Salary { get; set; }
    }
}
```



#### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

**Code for "HomeController.cs"**

```

using System;
using System.Web.Mvc;
using System.Collections.Generic;
using StronglyTypedViewsWithCollectionsExample.Models;

namespace StronglyTypedViewsWithCollectionsExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            List<Employee> emps = new List<Employee>()
            {
                new Employee() { EmpID = 1, EmpName = "Scott", Salary = 5000 },
                new Employee() { EmpID = 2, EmpName = "Allen", Salary = 6000 },
                new Employee() { EmpID = 3, EmpName = "John", Salary = 7000 }
            };
            return View(emps);
        }
    }
}

```

**Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Index.cshtml"**

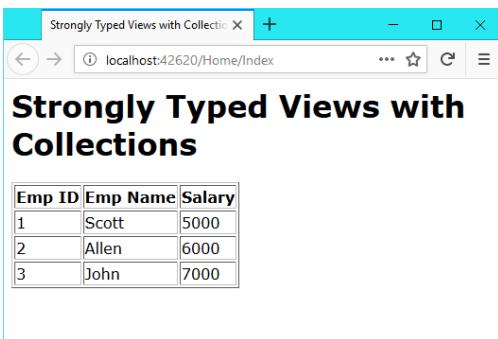
```

@model List<StronglyTypedViewsWithCollectionsExample.Models.Employee>
<html>
<head>
    <title>Strongly Typed Views with Collections</title>
</head>
<body>
    <h1>Strongly Typed Views with Collections</h1>
    <table border="1">
        <tr>
            <th>Emp ID</th>
            <th>Emp Name</th>
            <th>Salary</th>
        </tr>
        @foreach (var item in Model)
        {
            <tr>
                <td>@item.EmpID</td>
                <td>@item.EmpName</td>
                <td>@item.Salary</td>
            </tr>
        }
    </table>
</body>
</html>

```

**Running the application**

- Press "F5" to run the application.
  - Type "http://localhost:portnumber/Home/Index".
- Output:



## View Model

- View-specific model is called as "ViewModel". Use view model in following cases:
  - To pass specific set of properties to view.
  - To pass multiple model's data to the view.

### Syntax of View Model

```
class.viewmodelclassname
{
  public modelclassname1 property1 { get; set; }
  public modelclassname2 property2 { get; set; }
  ...
}
```

### Syntax of Strongly Typed View With View Model

```
@model.viewmodelclassname
```

## View Models - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7".
- Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "ViewModelExample". Type the location as "C:\Mvc". Type the solution name as "ViewModelExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

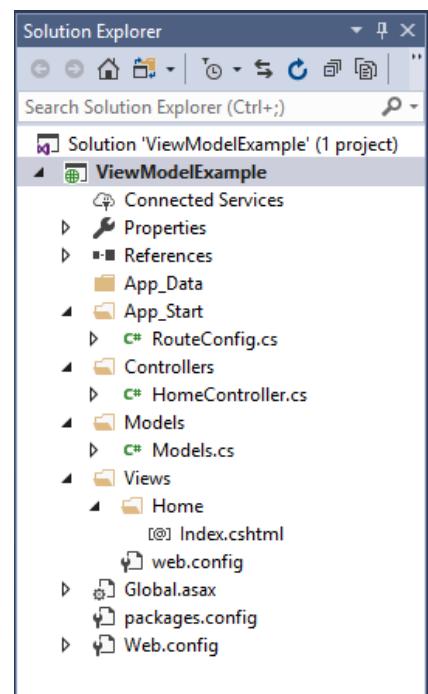
### Creating Models.cs

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Models.cs". Click on "Add".

### Code for "Models.cs"

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

namespace ViewModelExample.Models
{
  public class Employee
  {
    public int EmpID { get; set; }
    public string EmpName { get; set; }
    public double Salary { get; set; }
  }
  public class Product
  {
    public int ProductID { get; set; }
    public string ProductName { get; set; }
    public double Cost { get; set; }
  }
}
```



```

}
public class EmployeeAndProductViewModel
{
    public List<Employee> emps { get; set; }
    public List<Product> prods { get; set; }
}

```

#### **Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### **Code for "HomeController.cs"**

```

using System;
using System.Collections.Generic;
using System.Web.Mvc;
using ViewModelExample.Models;

namespace ViewModelExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            List<Employee> myemployees = new List<Employee>()
            {
                new Employee() { EmpID = 1, EmpName = "Scott", Salary = 5000 },
                new Employee() { EmpID = 2, EmpName = "Smith", Salary = 6000 },
                new Employee() { EmpID = 3, EmpName = "Jones", Salary = 7000 }
            };

            List<Product> myproducts = new List<Product>()
            {
                new Product() { ProductID = 1, ProductName = "Mobile", Cost = 20000 },
                new Product() { ProductID = 2, ProductName = "Laptop", Cost = 40000 },
                new Product() { ProductID = 3, ProductName = "TV", Cost = 60000 },
                new Product() { ProductID = 4, ProductName = "Tablet", Cost = 25000 }
            };
            EmployeeAndProductViewModel vm;
            vm = new EmployeeAndProductViewModel();
            vm.emps = myemployees;
            vm.prods = myproducts;
            return View(vm);
        }
    }
}

```

#### **Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### **Code for "Views\Home\Index.cshtml"**

```

@model ViewModelExample.Models.EmployeeAndProductViewModel
<html>
<head>
    <title>View Model</title>
</head>
<body>
    <h1>ViewModel</h1>
    <table border="1">
        <caption>Employees</caption>
        <tr>

```

```

<th>Emp ID</th>
<th>Emp Name</th>
<th>Salary</th>
</tr>
@foreach (var item in Model.emps)
{
<tr>
<td>@item.EmpID</td>
<td>@item.EmpName</td>
<td>@item.Salary</td>
</tr>
}
</table>



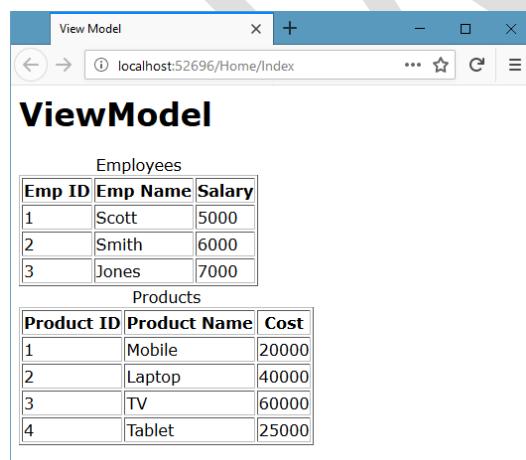


```

#### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

#### Output:



**Employees**

Emp ID	Emp Name	Salary
1	Scott	5000
2	Smith	6000
3	Jones	7000

**Products**

Product ID	Product Name	Cost
1	Mobile	20000
2	Laptop	40000
3	TV	60000
4	Tablet	25000

## ENTITY FRAMEWORK

### Introduction to Entity Framework

- "ADO.NET Entity Framework" is the Microsoft's most popular database framework in recent times. "ADO.NET Entity Framework (EF)" is used to interact with database. "EF (ADO.NET Entity Framework) uses ADO.NET internally. ADO.NET EF is the "easy version of ADO.NET". In ADO.NET EF, we have to use LINQ queries (instead of SQL). However, the LINQ queries will be converted into SQL queries internally and those SQL queries will be executed at database server.

- ADO.NET EF converts the “result data” into a collection format automatically. Collections are easy to handle and easy to apply data bindings. ADO.NET EF supports all database operations such as INSERT, UPDATE, DELETE, SELECT, calling stored procedures etc. ADO.NET EF is implemented using “EDMX” file (Entity Data Model Extended) or with manual coding. For easy modifications, manual coding approach is recommended.

### Advantages of EF

- Better OOP.
- Easy & less amount of code.
- Easy to combine with other frameworks such as “ASP.NET MVC” and “WPF” etc.

### Basic Concepts of Entity Framework

- **Model class:**
  - A table is represented as a model class. All the columns become as properties.
- **DbSet:**
  - A DbSet represents a table. A DbSet is a collection of objects of model class.
- **DbContext:**
  - DbContext is like “DataSet” in ado.net. DbContext is a collection of DbSet's.

### Syntax of Model Class

```
using System;
namespace Projectname.Models
{
    public class Modelclassname
    {
        public Datatype Property1 { get; set; }
        public Datatype Property2 { get; set; }
        ...
    }
}
```

### Syntax of DbContext Class

```
using System;
using System.Data.Entity;

namespace Projectname.Models
{
    public class DbContextclassname : DbContext
    {
        public virtual DbSet<Modelclassname> Collectionname { get; set; }
        ...
    }
}
```

## Entity Framework - Example

### Creating Project

- Open Visual Studio 2017. Go to “File” – “New” – “Project”. Select “.NET Framework 4.7”. Select “Visual C#”. Select “ASP.NET Web Application (.NET Framework)”. Type the project name “EntityFrameworkExample”. Type the location as “C:\Mvc”. Type the solution name as “EntityFrameworkExample”. Click on OK. Select “Empty”. Check the checkbox “MVC”. Uncheck the checkbox “Enable Docker support”. Uncheck the checkbox “Add unit tests”. Click on OK.

### Creating Database

- Open SQL Server Management Studio. Select “Windows Authentication”.
- Click on “Connect”.

- Ignore this step if "company" database already exists.
- Click on "New Query".
- Type the following code:

```

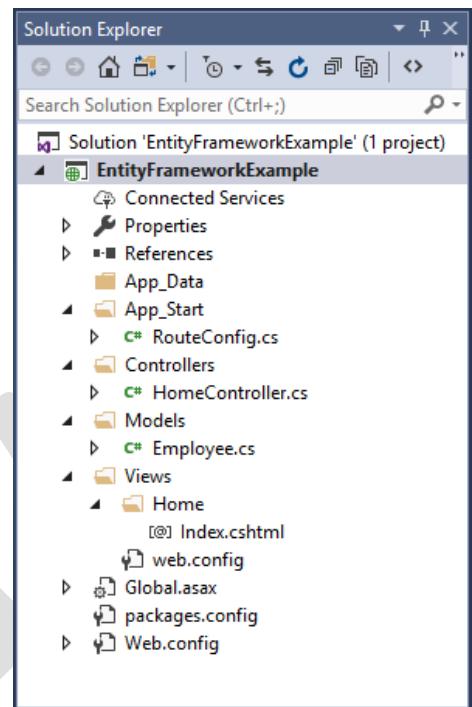
create database company
go
use company
go
create table Employees(EmpID int primary key, EmpName nvarchar(max),
Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go

```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### **Installing Packages**

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Type the following command in "Package Manager Console" and press Enter.  
`install-package EntityFramework`



### **Creating Employee.cs**

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

#### **Code for "Employee.cs"**

```

using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;

namespace EntityFrameworkExample.Models
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }

        public DbSet<Employee> Employees { get; set; }
    }
}

```

### **Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### **Code for "HomeController.cs"**

```

using System;
using System.Collections.Generic;

```

```

using System.Linq;
using System.Web.Mvc;
using EntityFrameworkExample.Models;

namespace EntityFrameworkExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            CompanyDbContext db = new CompanyDbContext();
            List<Employee> emps = db.Employees.ToList();
            return View(emps);
        }
    }
}

```

#### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### Code for "Views\Home\Index.cshtml"

```

@model List<EntityFrameworkExample.Models.Employee>
<html>
    <head>
        <title>Entity Framework</title>
    </head>
    <body>
        <h1>Entity Framework</h1>
        <table border="1">
            <tr>
                <th>Emp ID</th>
                <th>Emp Name</th>
                <th>Salary</th>
            </tr>
            @foreach (var emp in Model)
            {
                <tr>
                    <td>@emp.EmpID</td>
                    <td>@emp.EmpName</td>
                    <td>@emp.Salary</td>
                </tr>
            }
        </table>
    </body>
</html>

```

#### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:

Emp ID	Emp Name	Salary
1	Scott	4000
2	Allen	2500
3	Jones	5200
4	James	4400
5	Smith	7600

## Entity Framework – Search - Example

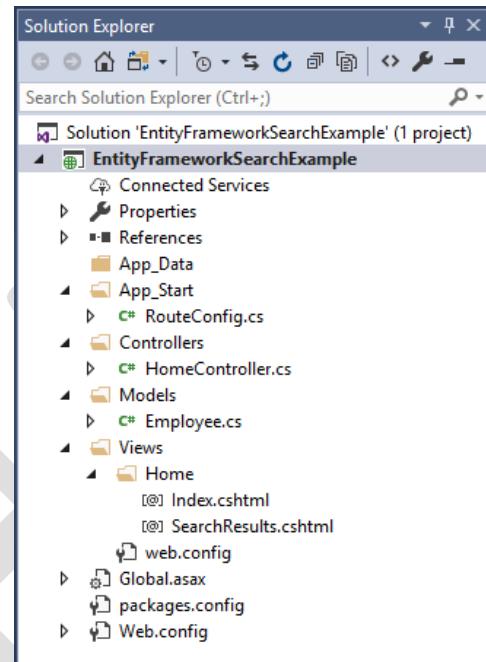
### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "EntityFrameworkSearchExample". Type the location as "C:\Mvc". Type the solution name as "EntityFrameworkSearchExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```
create database company
go
use company
go
create table Employees(
EmpID int primary key,
EmpName nvarchar(max),
Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```



- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
```

### Creating Employee.cs

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

### Code for "Employee.cs"

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;

namespace EntityFrameworkSearchExample.Models
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }
    }
}
```

```
public DbSet<Employee> Employees { get; set; }
}
```

### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### Code for "HomeController.cs"

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

namespace EntityFrameworkSearchExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        [HttpGet]
        public ActionResult SearchResults(string str)
        {
            CompanyDbContext db = new CompanyDbContext();
            List<Employee> emps = db.Employees.Where(temp => temp.EmpName.Contains(str)).ToList();
            return View(emps);
        }
    }
}
```

### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### Code for "Views\Home\Index.cshtml"

```
<html>
  <head>
    <title>Search</title>
  </head>
  <body>
    <h1>Employees - Search</h1>
    <form action="/Home/SearchResults" method="get">
      Search: <input type="text" name="str"/>
      <input type="submit" value="Search" />
    </form>
  </body>
</html>
```

### Creating SearchResults.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "SearchResults". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### Code for "Views\Home\SearchResults.cshtml"

```
@model List<EntityFrameworkSearchExample.Models.Employee>
<html>
  <head>
    <title>Search Results</title>
  </head>
  <body>
```

```

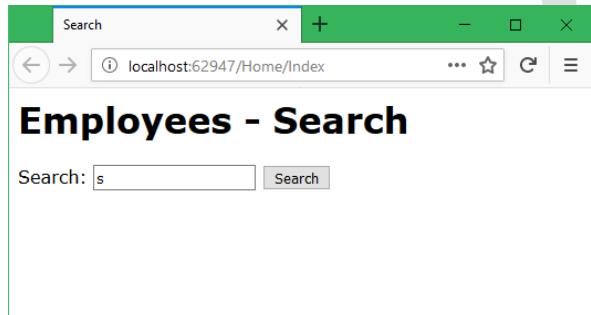
<h1>Search Results</h1>
<table border="1">
<tr>
<th>Emp ID</th>
<th>Emp Name</th>
<th>Salary</th>
</tr>
@foreach (var emp in Model)
{
<tr>
<td>@emp.EmpID</td>
<td>@emp.EmpName</td>
<td>@emp.Salary</td>
</tr>
}
</table>
Back to search
</body>
</html>

```

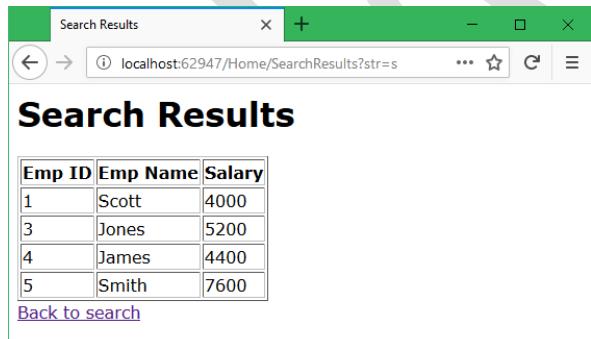
#### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



Type some text and click on "Search".



## Entity Framework – Stored Procedure - Example

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "EntityFrameworkSearchSPExample". Type the location as "C:\Mvc". Type the solution name as "EntityFrameworkSearchSPExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

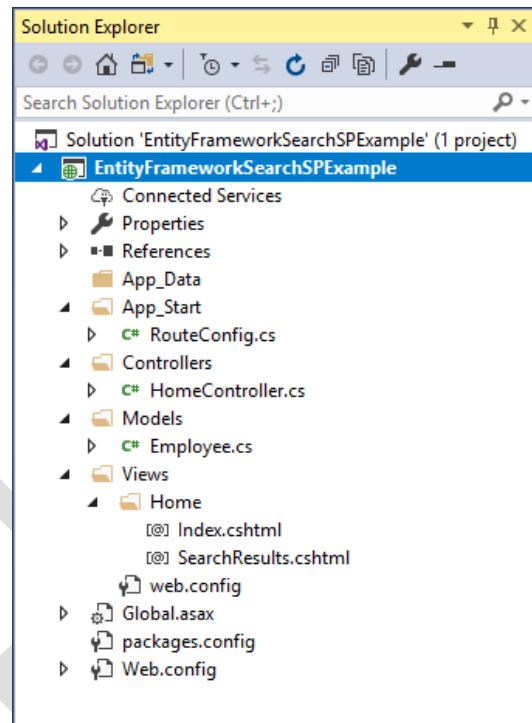
### **Creating Database**

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```

create database company
go
use company
go
create table Employees(
EmpID int primary key,
EmpName nvarchar(max),
Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
create procedure sp_searchemployees
(@str nvarchar(max))
as begin
    select * from Employees where EmpName like '%' + @str + '%'
end
go

```



- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### **Installing Packages**

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Type the following command in "Package Manager Console" and press Enter.  
`install-package EntityFramework`

### **Creating Employee.cs**

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

#### **Code for "Employee.cs"**

```

using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
using System.Data.SqlClient;
using System.Collections.Generic;
using System.Linq;

namespace EntityFrameworkSearchSPEExample.Models
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
    }
}

```

```

    }

    public DbSet<Employee> Employees { get; set; }

    public List<Employee> GetEmpsStoredProcedure(string str)
    {
        SqlParameter p1 = new SqlParameter("@str", str);
        return this.Database.SqlQuery<Employee>("sp_searchemployees @str", p1).ToList();
    }
}
}
}

```

**Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

**Code for "HomeController.cs"**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Mvc;
using EntityFrameworkSearchSPEExample.Models;

namespace EntityFrameworkSearchSPEExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        [HttpGet]
        public ActionResult SearchResults(string str)
        {
            CompanyDbContext db = new CompanyDbContext();
            var emps = db.GetEmpsStoredProcedure(str);
            return View(emps);
        }
    }
}

```

**Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Index.cshtml"**

```

<html>
    <head>
        <title>Search</title>
    </head>
    <body>
        <h1>Employees - Search</h1>
        <form action="/Home/SearchResults" method="get">
            Search: <input type="text" name="str" />
            <input type="submit" value="Search" />
        </form>
    </body>
</html>

```

**Creating SearchResults.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "SearchResults". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

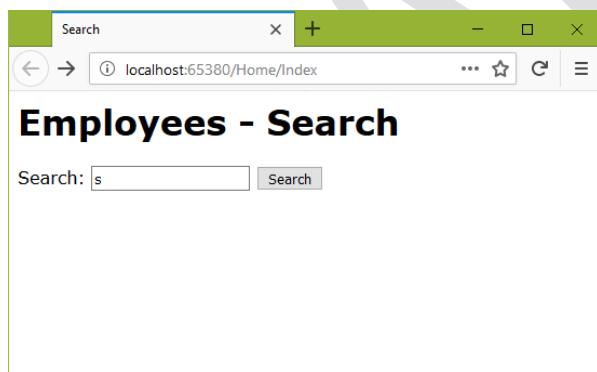
**Code for "Views\Home\SearchResults.cshtml"**

```
@model List<EntityFrameworkSearchSPEExample.Models.Employee>
<html>
<head>
    <title>Search Results</title>
</head>
<body>
    <h1>Search Results</h1>
    <table border="1">
        <tr>
            <th>Emp ID</th>
            <th>Emp Name</th>
            <th>Salary</th>
        </tr>
        @foreach (var emp in Model)
        {
            <tr>
                <td>@emp.EmpID</td>
                <td>@emp.EmpName</td>
                <td>@emp.Salary</td>
            </tr>
        }
    </table>
    <a href="/Home/Index">Back to search</a>
</body>
</html>
```

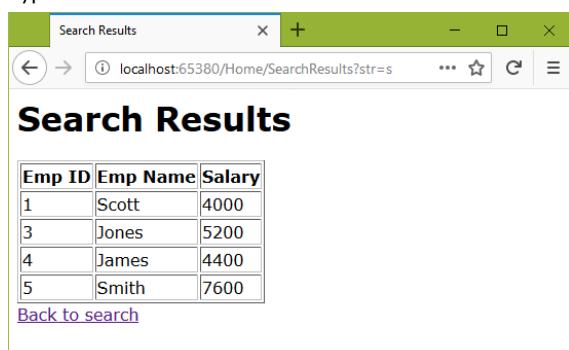
**Running the application**

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



Type some text and click on "Search".



## Entity Framework – Insert - Example

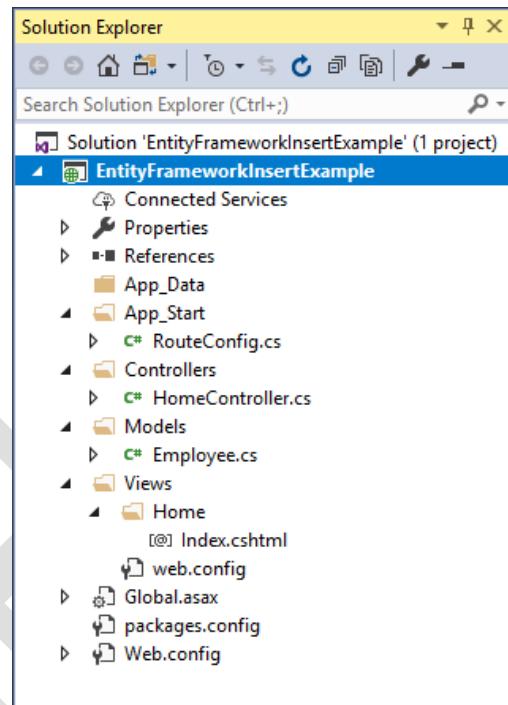
### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "EntityFrameworkInsertExample". Type the location as "C:\Mvc". Type the solution name as "EntityFrameworkInsertExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```
create database company
go
use company
go
create table Employees(EmpID int primary key, EmpName
nvarchar(max), Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```



- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
```

### Creating Employee.cs

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

#### Code for "Employee.cs"

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Data.Entity;

namespace EntityFrameworkInsertExample.Models
{
    public class Employee
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
    }
}
```

```

    }
    public DbSet<Employee> Employees { get; set; }
}
}

```

#### **Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### **Code for "HomeController.cs"**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Mvc;
using EntityFrameworkInsertExample.Models;

namespace EntityFrameworkInsertExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        [HttpPost]
        public ActionResult InsertEmp(Employee emp)
        {
            CompanyDbContext db = new CompanyDbContext();
            db.Employees.Add(emp);
            db.SaveChanges();
            return Content("Successfully Inserted");
        }
    }
}

```

#### **Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### **Code for "Views\Home\Index.cshtml"**

```

<html>
  <head>
    <title>Entity Framework - Insert</title>
  </head>
  <body>
    <h1>Entity Framework - Insert</h1>
    <form action="/Home/InsertEmp" method="post">
      Emp ID: <input type="text" name="EmpID" /><br />
      Emp Name: <input type="text" name="EmpName" /><br />
      Salary: <input type="text" name="Salary" /><br />
      <input type="submit" value="Insert"/>
    </form>
  </body>
</html>

```

#### **Running the application**

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:

Entity Framework - Insert

localhost:65469/Home/Index

**Entity Framework - Insert**

Emp ID:

Emp Name:

Salary:

**Insert**

Type some details and click on "Insert".

localhost:65469/Home/InsertEmp

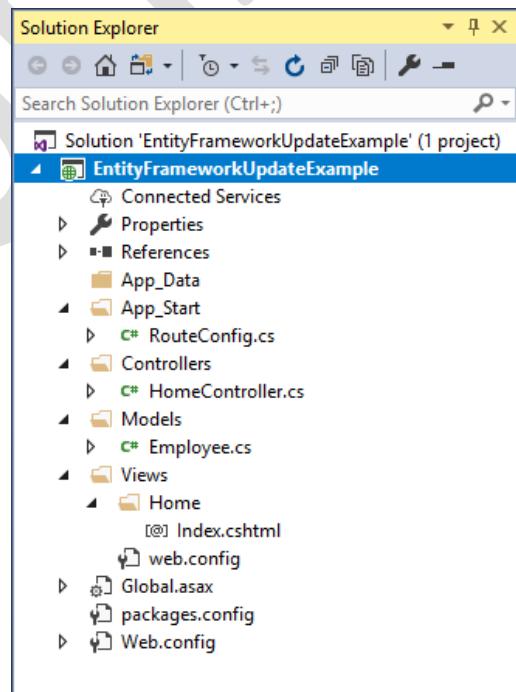
localhost:65469/Home/InsertEmp

**Successfully Inserted**

## Entity Framework – Update – Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "EntityFrameworkUpdateExample". Type the location as "C:\Mvc". Type the solution name as "EntityFrameworkUpdateExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.



### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```
create database company
go
use company
go
create table Employees(
    EmpID int primary key,
    EmpName nvarchar(max),
    Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Type the following command in "Package Manager Console" and press Enter.  
`install-package EntityFramework`

### Creating Employee.cs

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

### Code for "Employee.cs"

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;

namespace EntityFrameworkUpdateExample.Models
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }

        public DbSet<Employee> Employees { get; set; }
    }
}
```

### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

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

namespace EntityFrameworkUpdateExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        [HttpPost]
        public ActionResult UpdateEmp(Employee emp)
        {
            CompanyDbContext db = new CompanyDbContext();
            Employee emp2 = db.Employees.Where(temp => temp.EmpID == emp.EmpID).FirstOrDefault();
            emp2.EmpName = emp.EmpName;
            emp2.Salary = emp.Salary;
            db.SaveChanges();
            return Content("Successfully Updated");
        }
    }
}
```

### **Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### **Code for "Views\Home\Index.cshtml"**

```

<html>
  <head>
    <title>Entity Framework - Update</title>
  </head>
  <body>
    <h1>Entity Framework - Update</h1>
    <form action="/Home/UpdateEmp" method="post">
      Existing Emp ID: <input type="text" name="EmpID"/><br />
      Emp Name: <input type="text" name="EmpName"/><br />
      Salary: <input type="text" name="Salary"/><br />
      <input type="submit" value="Update"/>
    </form>
  </body>
</html>

```

### **Running the application**

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:

Entity Framework - Update

Existing Emp ID: 20  
Emp Name: pqr  
Salary: 8000  
**Update**

Type some details and click on "Update".

localhost:54700/Home/UpdateEmp

Successfully Updated

## Entity Framework – Delete - Example

### **Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "EntityFrameworkDeleteExample". Type the location as "C:\Mvc". Type the solution name as "EntityFrameworkDeleteExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### **Creating Database**

- Open SQL Server Management Studio. Select "Windows Authentication".
- Click on "Connect".
- Ignore this step if "company" database already exists.
- Click on "New Query".
- Type the following code:

```
create database company
```

```

go
use company
go
create table Employees(
EmpID int primary key,
EmpName nvarchar(max),
Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go

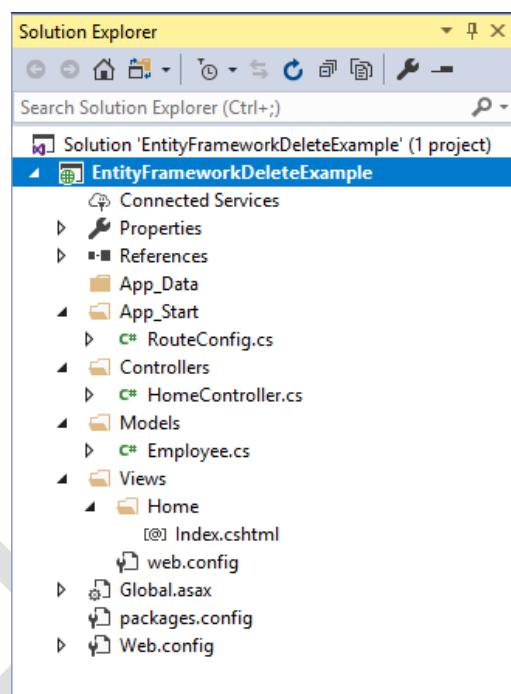
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

#### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
```



#### Creating Employee.cs

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

#### Code for "Employee.cs"

```

using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;

namespace EntityFrameworkDeleteExample.Models
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }
        public DbSet<Employee> Employees { get; set; }
    }
}

```

#### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### Code for "HomeController.cs"

```

using System;
using System.Collections.Generic;
using System.Linq;

```

```

using System.Web.Mvc;
using EntityFrameworkDeleteExample.Models;

namespace EntityFrameworkDeleteExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        [HttpPost]
        public ActionResult DeleteEmp(int n)
        {
            CompanyDbContext db = new CompanyDbContext();
            Employee emp = db.Employees.Where(temp => temp.EmpID == n).FirstOrDefault();
            db.Employees.Remove(emp);
            db.SaveChanges();
            return Content("Successfully Deleted");
        }
    }
}

```

#### **[Creating Index.cshtml]**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### **Code for "Views\Home\Index.cshtml"**

```

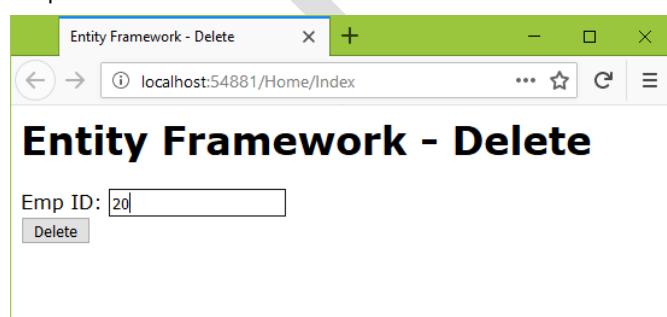
<html>
  <head>
    <title>Entity Framework - Delete</title>
  </head>
  <body>
    <h1>Entity Framework - Delete</h1>
    <form action="/Home/DeleteEmp" method="post">
      Emp ID: <input type="text" name="n" /><br />
      <input type="submit" value="Delete" />
    </form>
  </body>
</html>

```

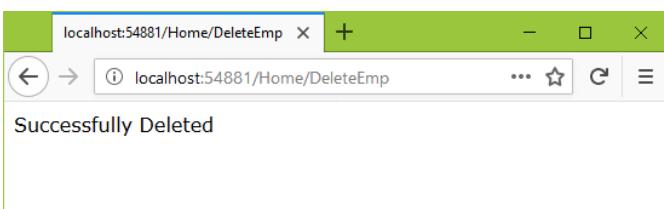
#### **[Running the application]**

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



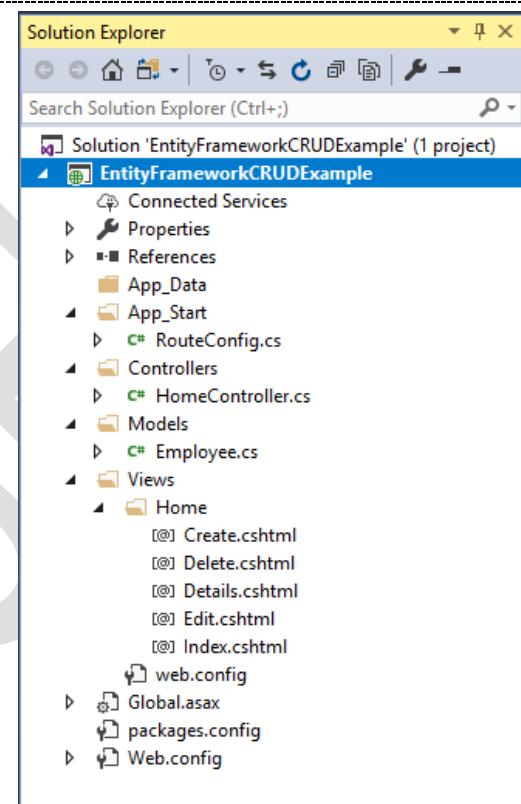
Type some details and click on "Delete".



### Entity Framework – CRUD - Example

#### **Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "EntityFrameworkCRUDExample". Type the location as "C:\Mvc". Type the solution name as "EntityFrameworkCRUDExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.



#### **Creating Database**

- Open SQL Server Management Studio. Select "Windows Authentication".
- Click on "Connect".
- Ignore this step if "company" database already exists.
- Click on "New Query".
- Type the following code:

```
create database company
go
use company
go
create table Employees(EmpID int primary key, EmpName
nvarchar(max), Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

#### **Installing Packages**

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Type the following command in "Package Manager Console" and press Enter.  
`install-package EntityFramework`

#### **Creating Employee.cs**

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

#### **Code for "Employee.cs"**

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Data.Entity;

namespace EntityFrameworkCRUDExample.Models
{
    public class Employee
```

```

{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.None)]
    public int EmpID { get; set; }
    public string EmpName { get; set; }
    public decimal Salary { get; set; }
}

public class CompanyDbContext : DbContext
{
    public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
    {
    }

    public DbSet<Employee> Employees { get; set; }
}

```

#### **Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### **Code for "HomeController.cs"**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Mvc;
using EntityFrameworkCRUDExample.Models;

namespace EntityFrameworkCRUDExample.Controllers
{
    public class HomeController : Controller
    {
        CompanyDbContext db = new CompanyDbContext();

        public ActionResult Index()
        {
            List<Employee> emps = db.Employees.ToList();
            return View(emps);
        }

        public ActionResult Create()
        {
            return View();
        }

        [HttpPost]
        public ActionResult Create(Employee emp)
        {
            db.Employees.Add(emp);
            db.SaveChanges();
            return RedirectToAction("Index");
        }

        public ActionResult Details(int id)
        {
            Employee emp = db.Employees.Find(id);
            return View(emp);
        }

        public ActionResult Edit(int id)
        {
            Employee emp = db.Employees.Find(id);

```

```

        return View(emp);
    }

    [HttpPost]
    public ActionResult Edit(Employee e)
    {
        Employee emp = db.Employees.Where(temp => temp.EmpID == e.EmpID).FirstOrDefault();
        emp.EmpName = e.EmpName;
        emp.Salary = e.Salary;
        db.SaveChanges();
        return RedirectToAction("Index");
    }

    public ActionResult Delete(int id)
    {
        Employee emp = db.Employees.Where(temp => temp.EmpID == id).FirstOrDefault();
        return View(emp);
    }

    [HttpPost]
    public ActionResult Delete(int id, Employee e)
    {
        Employee emp = db.Employees.Where(temp => temp.EmpID == id).FirstOrDefault();
        db.Employees.Remove(emp);
        db.SaveChanges();
        return RedirectToAction("Index");
    }
}
}
}

```

**Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Index.cshtml"**

```

@model List<EntityFrameworkCRUDExample.Models.Employee>
<html>
<head>
    <title>Entity Framework</title>
</head>
<body>
    <h1>Entity Framework</h1>
    <table border="1">
        <tr>
            <th>Emp ID</th>
            <th>Emp Name</th>
            <th>Salary</th>
            <th>Options</th>
        </tr>
        @foreach (var item in Model)
        {
            <tr>
                <td>@item.EmpID</td>
                <td>@item.EmpName</td>
                <td>@item.Salary</td>
                <td>
                    <a href="/Home/Details/@item.EmpID">Details</a>
                    <a href="/Home/Edit/@item.EmpID">Edit</a>
                    <a href="/Home/Delete/@item.EmpID">Delete</a>
                </td>
            </tr>
        }
    
```

```

</table>
<a href="/Home/Create">New Employee</a>
</body>
</html>

```

#### **Creating Details.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Details". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### **Code for "Views\Home\Details.cshtml"**

```

@model EntityFrameworkCRUDExample.Models.Employee
<html>
<head>
<title>Details</title>
</head>
<body>
<h1>Employees - Details</h1>
<table>
<tr>
<td>Emp ID:</td>
<td>@Model.EmpID</td>
</tr>
<tr>
<td>Emp Name:</td>
<td>@Model.EmpName</td>
</tr>
<tr>
<td>Salary:</td>
<td>@Model.Salary</td>
</tr>
</table>
<a href="/Home/Index">Back to list</a>
</body>
</html>

```

#### **Creating Create.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Create". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### **Code for "Views\Home\Create.cshtml"**

```

<html>
<head>
<title>Create</title>
</head>
<body>
<h1>Employees - Create</h1>
<form action="/Home/Create" method="post">
<table>
<tr>
<td>Emp ID:</td>
<td><input type="text" name="EmpID" /></td>
</tr>
<tr>
<td>Emp Name:</td>
<td><input type="text" name="EmpName" /></td>
</tr>
<tr>
<td>Salary:</td>
<td><input type="text" name="Salary" /></td>
</tr>
<tr>
<td></td>

```

```

<td><input type="submit" value="Create" /></td>
</tr>
</table>
<a href="/Home/Index">Back to list</a>
</form>
</body>
</html>

```

#### **Creating Edit.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Edit". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### **Code for "Views\Home>Edit.cshtml"**

```

@model EntityFrameworkCRUDEExample.Models.Employee
<html>
<head>
<title>Edit</title>
</head>
<body>
<h1>Employees - Edit</h1>
<form action="/Home/Edit" method="post">
<table>
<tr>
<td>Emp ID:</td>
<td><input type="text" name="EmpID" value="@Model.EmpID" readonly="readonly" /></td>
</tr>
<tr>
<td>Emp Name:</td>
<td><input type="text" name="EmpName" value="@Model.EmpName" /></td>
</tr>
<tr>
<td>Salary:</td>
<td><input type="text" name="Salary" value="@Model.Salary" /></td>
</tr>
<tr>
<td></td>
<td><input type="submit" value="Update" /></td>
</tr>
</table>
<a href="/Home/Index">Back to list</a>
</form>
</body>
</html>

```

#### **Creating Delete.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Delete". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### **Code for "Views\Home>Delete.cshtml"**

```

@model EntityFrameworkCRUDEExample.Models.Employee
<html>
<head>
<title>Delete</title>
</head>
<body>
<h1>Employees - Delete</h1>
<form action="/Home/Delete/@Model.EmpID" method="post">
<h2>Are you sure to delete this employee?</h2>
<table>
<tr>
<td>Emp ID:</td>
<td>@Model.EmpID</td>

```

```

</tr>
<tr>
<td>Emp Name:</td>
<td>@Model.EmpName</td>
</tr>
<tr>
<td>Salary:</td>
<td>@Model.Salary</td>
</tr>
<tr>
<td></td>
<td><input type="submit" value="Delete" /></td>
</tr>
</table>
<a href="/Home/Index">Back to list</a>
</form>
</body>
</html>

```

#### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:

Emp ID	Emp Name	Salary	Options
1	Scott	4000	<a href="#">Details</a> <a href="#">Edit</a> <a href="#">Delete</a>
2	Allen	2500	<a href="#">Details</a> <a href="#">Edit</a> <a href="#">Delete</a>
3	Jones	5200	<a href="#">Details</a> <a href="#">Edit</a> <a href="#">Delete</a>
4	James	4400	<a href="#">Details</a> <a href="#">Edit</a> <a href="#">Delete</a>
5	Smith	7600	<a href="#">Details</a> <a href="#">Edit</a> <a href="#">Delete</a>

## Code First Approach

- "ADO.NET Entity Framework" supports two types of database approaches:
  1. **Database First Approach:**
    - Database created first; model class next.
    - Useful if you have existing database.
  2. **Code First Approach:**
    - Model class created first; Database will be auto-generated.
    - Useful if you want to create database automatically on production server.

## Code First Approach - Example

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "EntityFrameworkInsertExample". Type the location as "C:\Mvc". Type the solution name as "EntityFrameworkInsertExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

#### Deleting old Database

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Delete the "Company2" database, if already exists.

## Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Type the following command in "Package Manager Console" and press Enter.  
**install-package EntityFramework**

## Creating Employee.cs

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

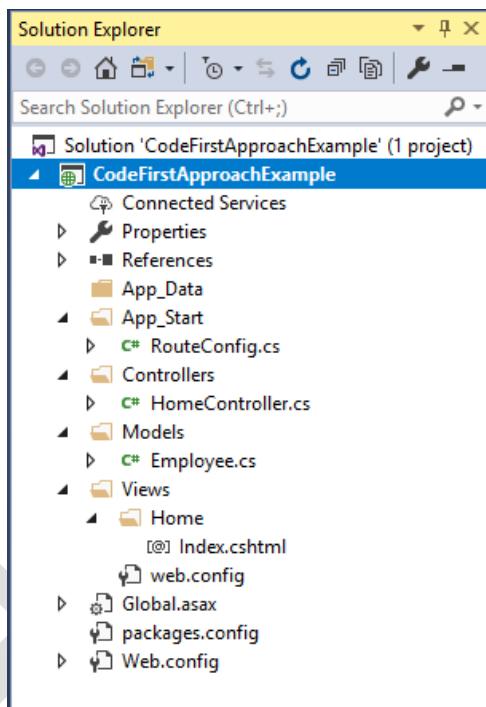
### Code for "Employee.cs"

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Data.Entity;

namespace CodeFirstApproachExample.Models
{
    public class Employee
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class Company2DbContext : DbContext
    {
        public Company2DbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company2")
        {
        }

        public DbSet<Employee> Employees { get; set; }
    }
}
```



## Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

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

namespace CodeFirstApproachExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        [HttpPost]
        public ActionResult InsertEmp(Employee emp)
        {
            Company2DbContext db = new Company2DbContext();
            db.Employees.Add(emp);
        }
    }
}
```

```

        db.SaveChanges();
        return Content("Successfully Inserted");
    }
}
}

```

#### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### Code for "Views\Home\Index.cshtml"

```

<html>
<head>
    <title>Code First Approach</title>
</head>
<body>
    <h1>Code First Approach</h1>
    <form action="/Home/InsertEmp" method="post">
        Emp ID: <input type="text" name="EmpID" /><br />
        Emp Name: <input type="text" name="EmpName" /><br />
        Salary: <input type="text" name="Salary" /><br />
        <input type="submit" value="Insert" />
    </form>
</body>
</html>

```

#### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

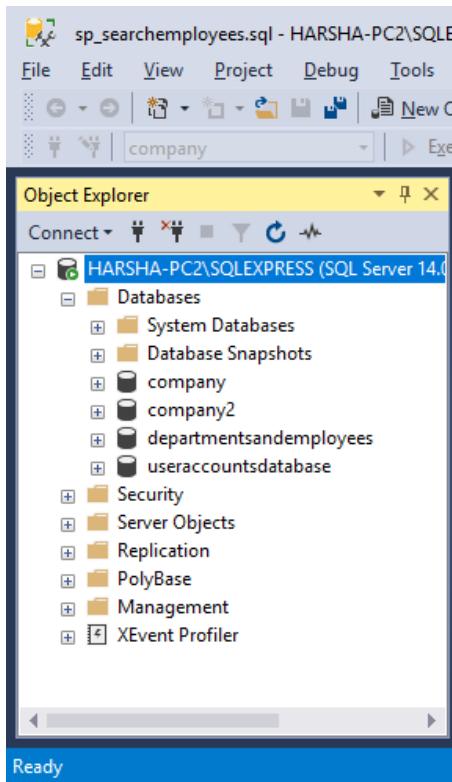
Output:

The screenshot shows a browser window with the title "Code First Approach". Inside the window, there is a form with three text input fields: "Emp ID" containing "67", "Emp Name" containing "asdf", and "Salary" containing "8500". Below the form is a single button labeled "Insert".

Type some details and click on "Insert".

The screenshot shows a browser window with the URL "localhost:51971/Home/InsertEmp". The page displays the message "Successfully Inserted" in a large font.

It creates the "company2" database in SQL Server automatically. Check in "SQL Server Management Studio" for "company2" database.



## SCAFFOLDING TEMPLATES

### What are Scaffolding Templates

- Scaffolding Templates are the pre-defined templates to easily generate the code for CRUD operations.
- There are two types of Scaffolding Templates:
  - Controller Scaffolding Templates
    - 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
  - View Scaffolding Templates
    - Empty (without model)
    - Empty
    - List
    - Details
    - Create
    - Edit
    - Delete

### Scaffolding Templates - Example

#### Creating Project

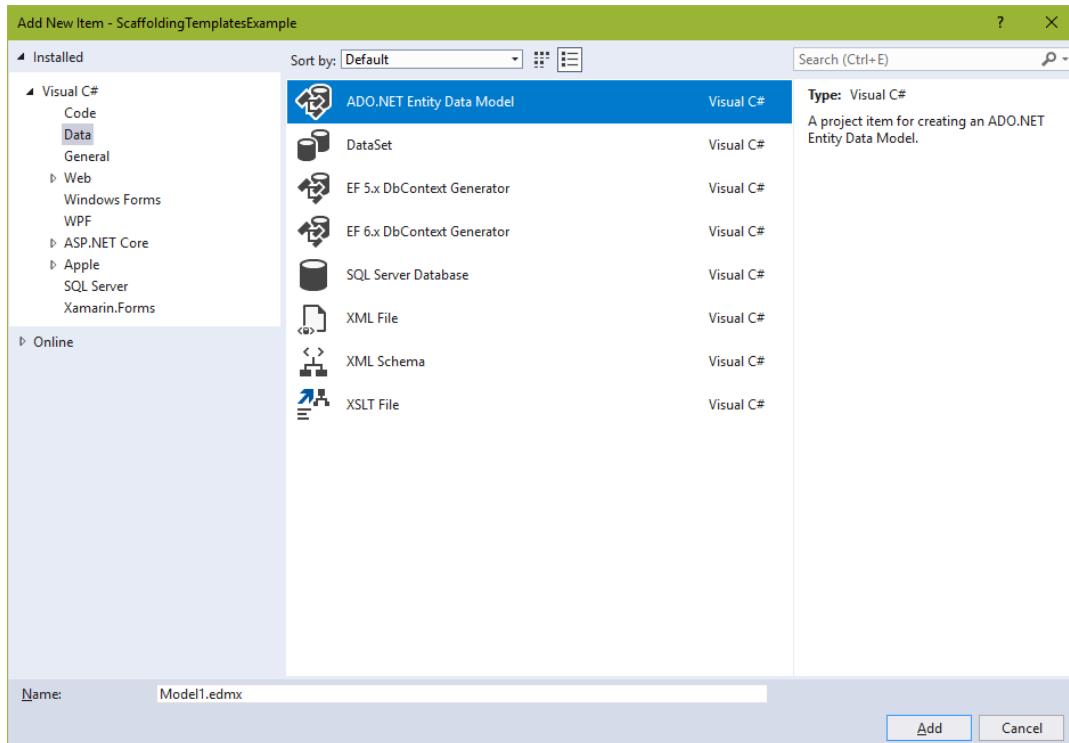
- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "ScaffoldingTemplatesExample". Type the location as "C:\Mvc". Type the solution name as "ScaffoldingTemplatesExample". Click on OK.

## Asp.Net Mvc 5

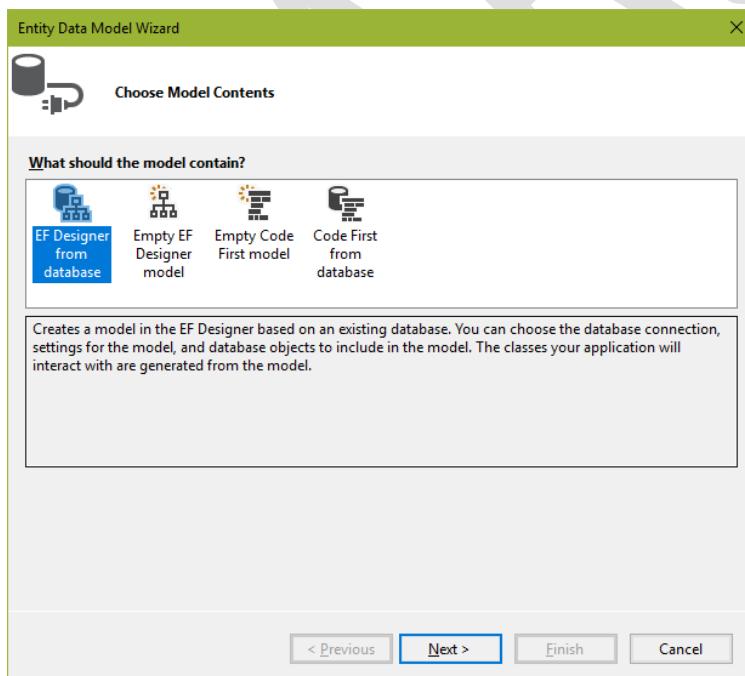
- Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating Models

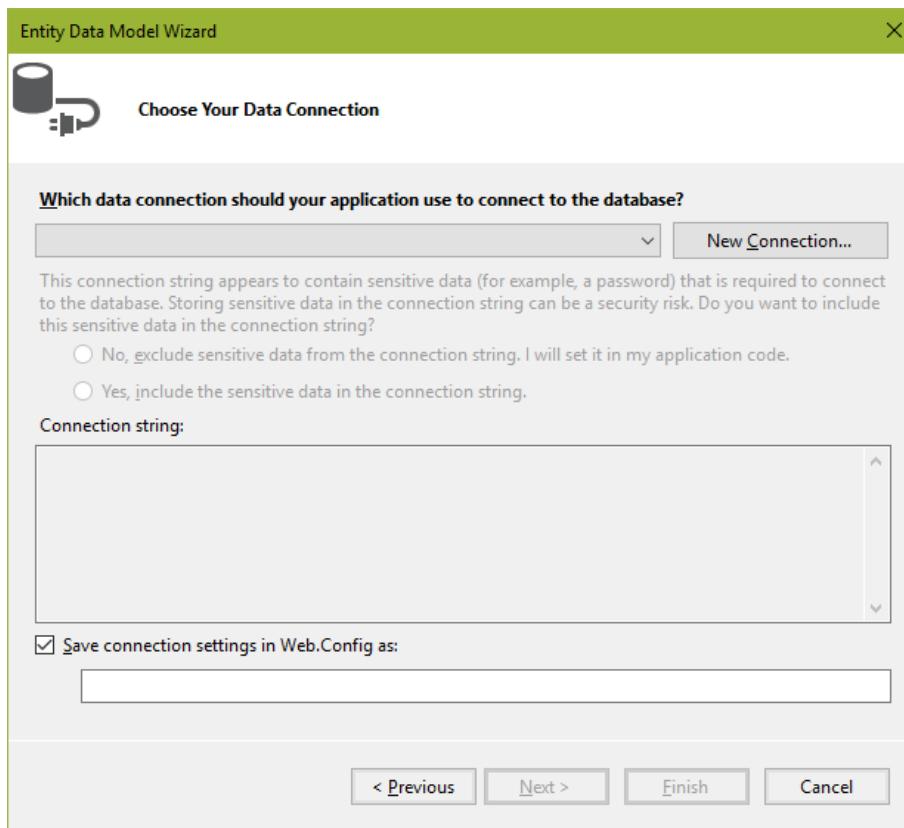
- Right click on "Models" folder and click on "Add" - "New Item". Click on "Data" - "ADO.NET Entity Data Model". Type the file name as "Modell.edmx". Click on "Add".



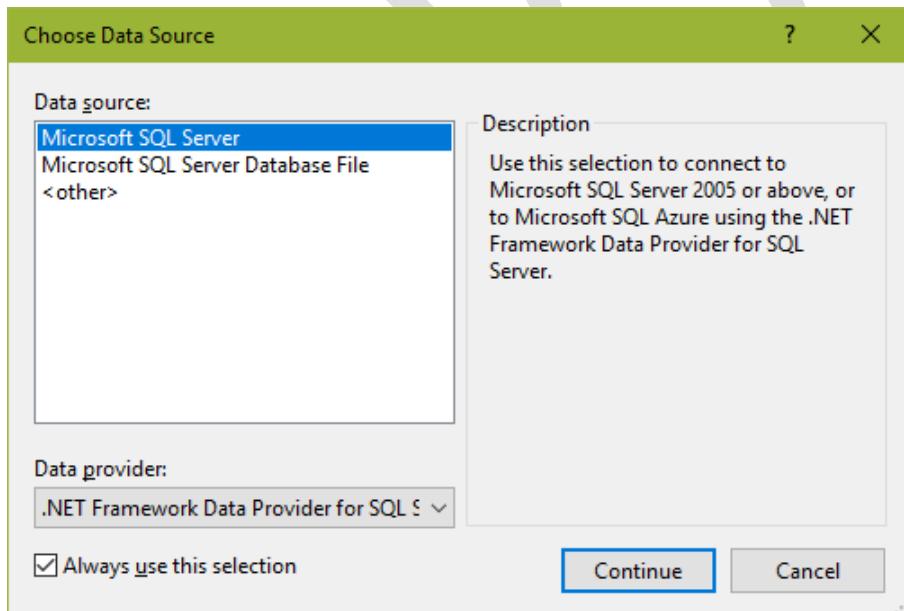
- Select "EF Designer from database".



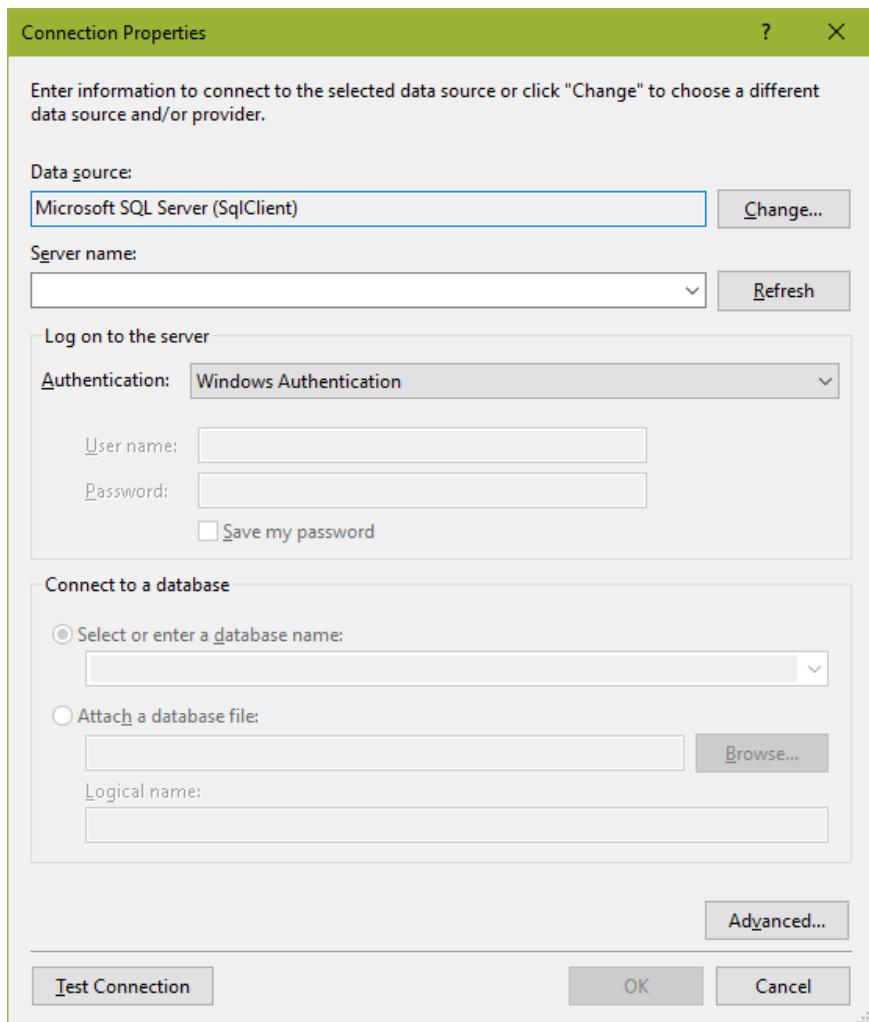
- Click on "Next".



- Click on "New Connection".



- Click on "Microsoft SQL Server".
- Select data provider as ".NET Framework Data Provider for SQL Server".
- Click on "Continue".



- Enter the server name as "localhost\sqlexpress".
- Select the Authentication as "Windows Authentication".
- Select the database name as "departmentsandemployees".

The screenshot shows the 'Connection Properties' dialog box on the left and the 'Solution Explorer' window on the right.

**Connection Properties Dialog:**

- Data source:** Microsoft SQL Server (SqlClient)
- Server name:** localhost\sqlexpress
- Authentication:** Windows Authentication
- User name:** (empty)
- Password:** (empty)
- Save my password:**
- Select or enter a database name:** departmentsandemployees
- Attach a database file:** (empty text box)
- Logical name:** (empty text box)
- Advanced...** button
- Test Connection** button
- OK** button (highlighted in blue)
- Cancel** button

**Solution Explorer:**

- Solution 'CodeFirstApproachExample' (1 project)
- CodeFirstApproachExample
  - Connected Services
  - Properties
  - References
  - App\_Data
  - App\_Start
    - RouteConfig.cs
  - Controllers
    - HomeController.cs
  - Models
    - Employee.cs
  - Views
    - Home
      - Index.cshtml
      - web.config
    - Global.asax
    - packages.config
    - Web.config

- Click on OK.

The screenshot shows the 'Entity Data Model Wizard' dialog box.

**Choose Your Data Connection**

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

harsha-pc2\sqlexpress.departmentsandemployees.dbo

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.

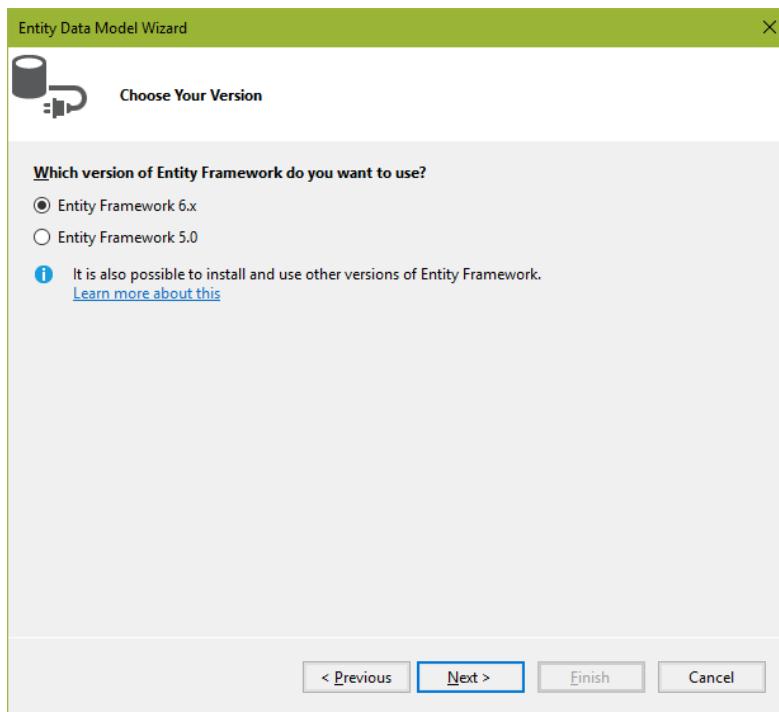
**Connection string:**

```
metadata=res://"/Models/Model1.csdl|res://"/Models/Model1.ssdl|res://"/Models/Model1.msl;provider=System.Data.SqlClient;provider connection string="data source=localhost\sqlexpress;initial catalog=departmentsandemployees;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework"
```

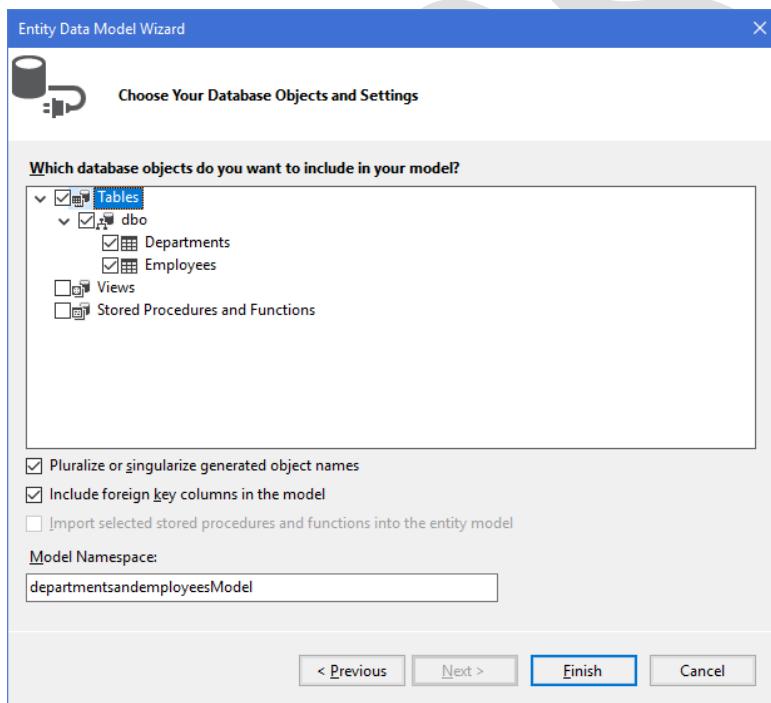
Save connection settings in Web.Config as:  
 departmentsandemployeesEntities

**Buttons:** < Previous, Next >, Finish, Cancel

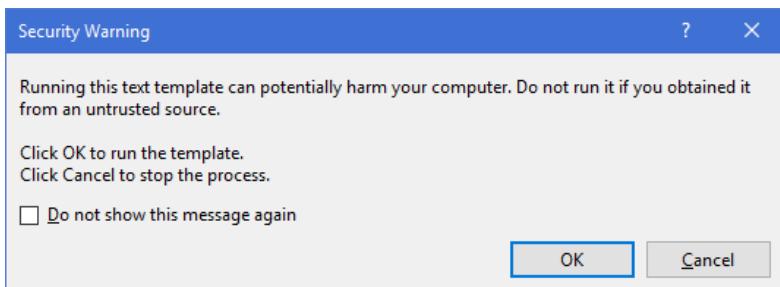
- Click on "Next".



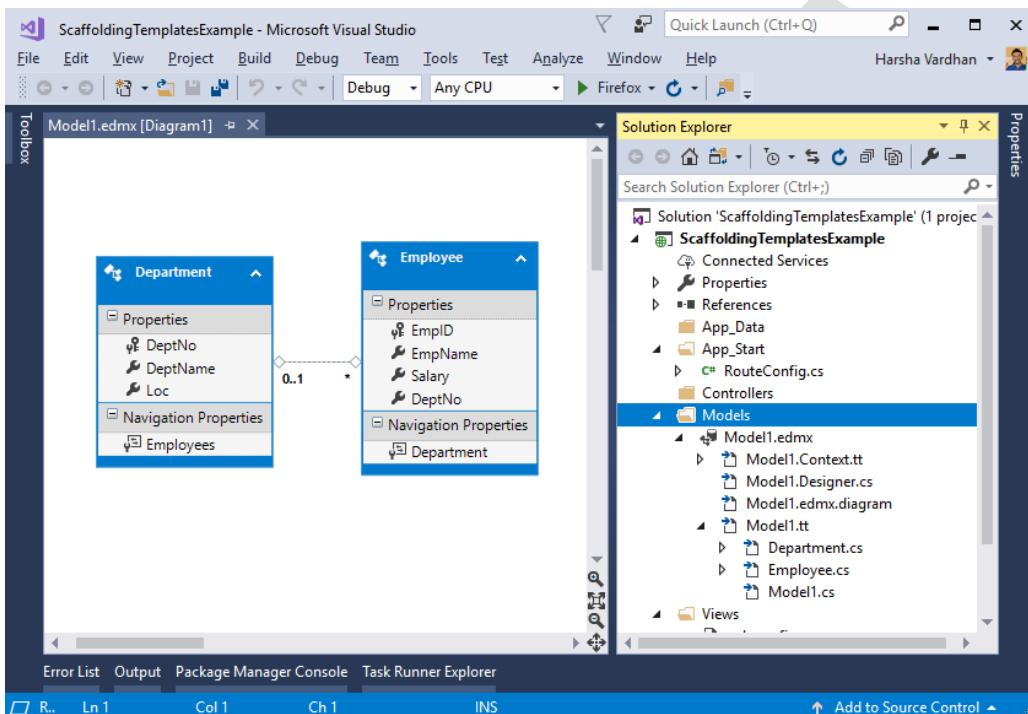
- Click on "Entity Framework 6.x".
- Click on "Next".



- Select "Departments" and "Employees" table.
- Click on "Finish".



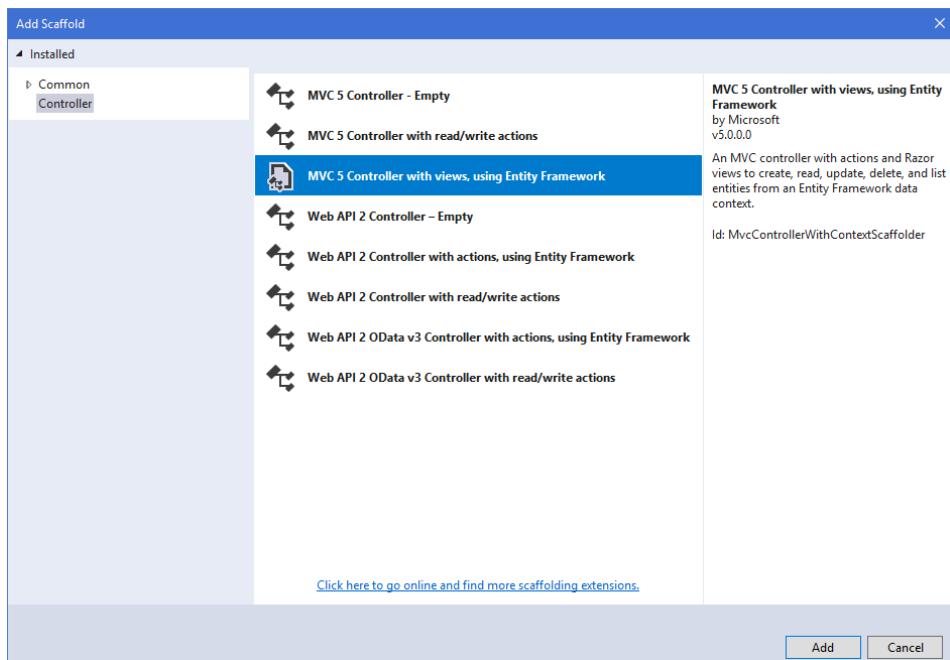
- Click on OK.
- Click on OK again.
- Click on OK once again.



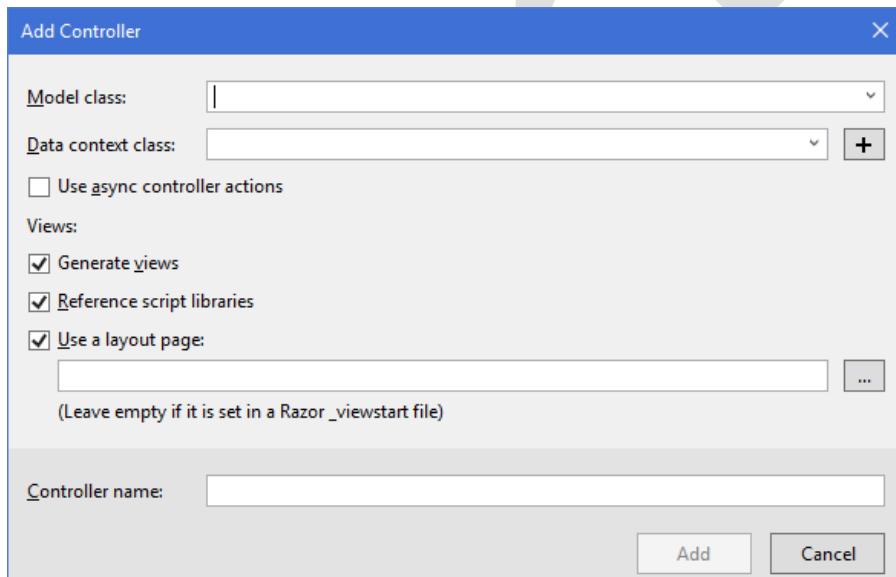
- Go to "Build" menu - click on "Build Solution".

### Creating DepartmentsController.cs

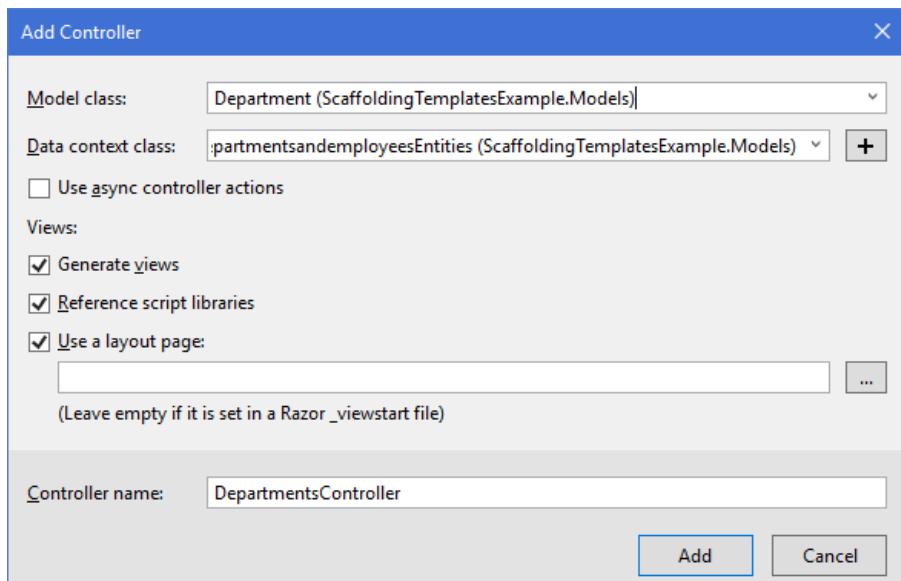
- Right click on "Controllers" folder and click on "Add" - "Controller".



- Select "MVC 5 Controller with views, using Entity Framework". Click on "Add".

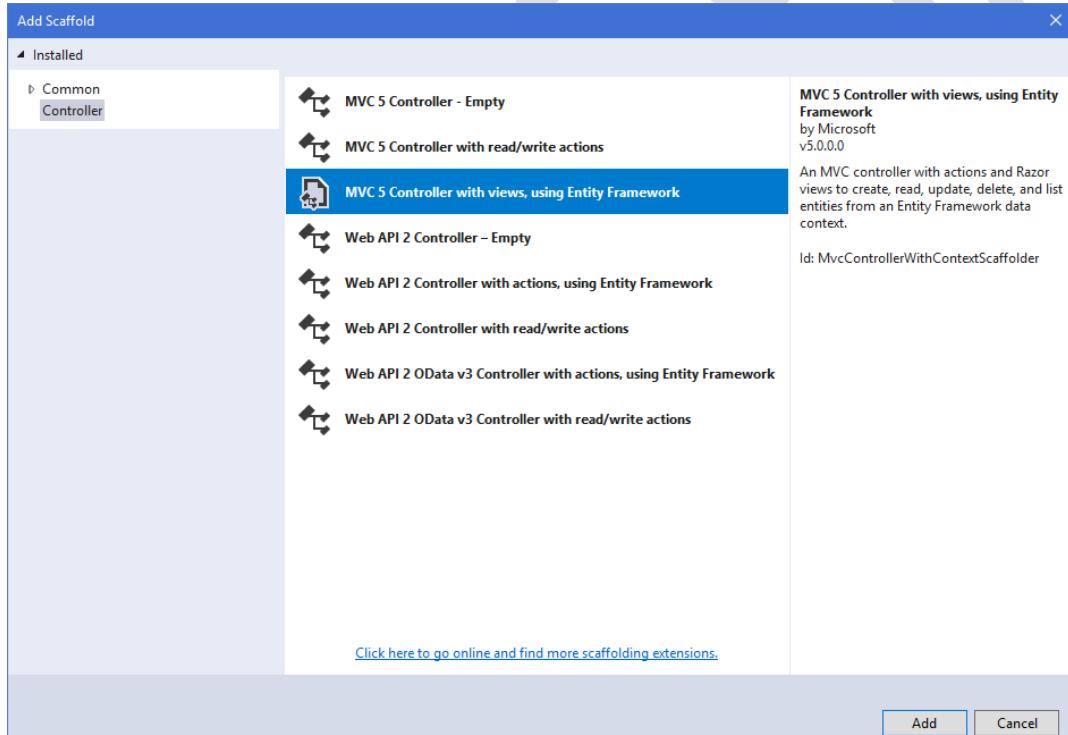


- Select the Model class as "Department (ScaffoldingTemplatesExample.Models)".
- Select the Data context class as "departmentsandemployeesEntities (ScaffoldingTemplatesExample.Models)".
- Check the checkbox "Generate views".
- Check the checkbox "Reference script libraries".
- Check the checkbox "Use a layout page".
- Enter the Controller name as "DepartmentsController".
- Click on "Add".

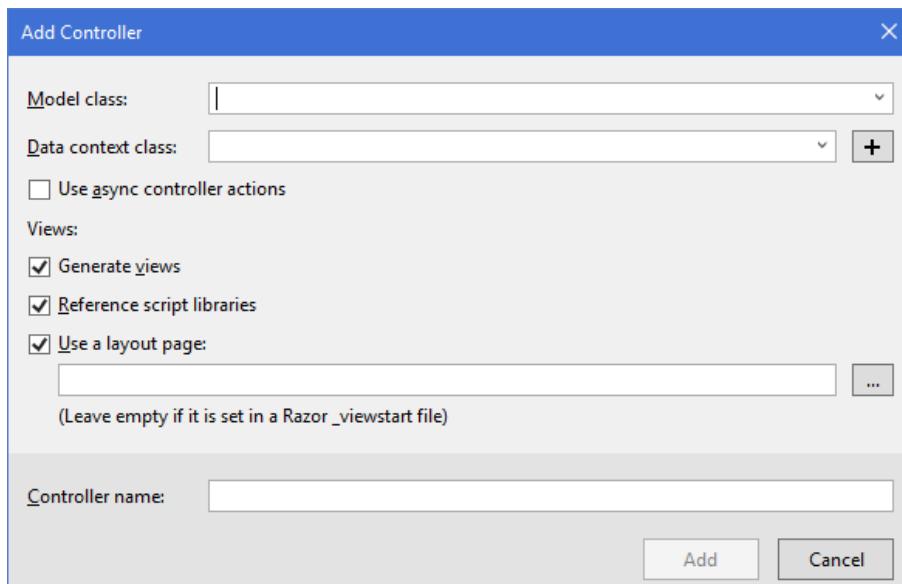


### Creating EmployeesController.cs

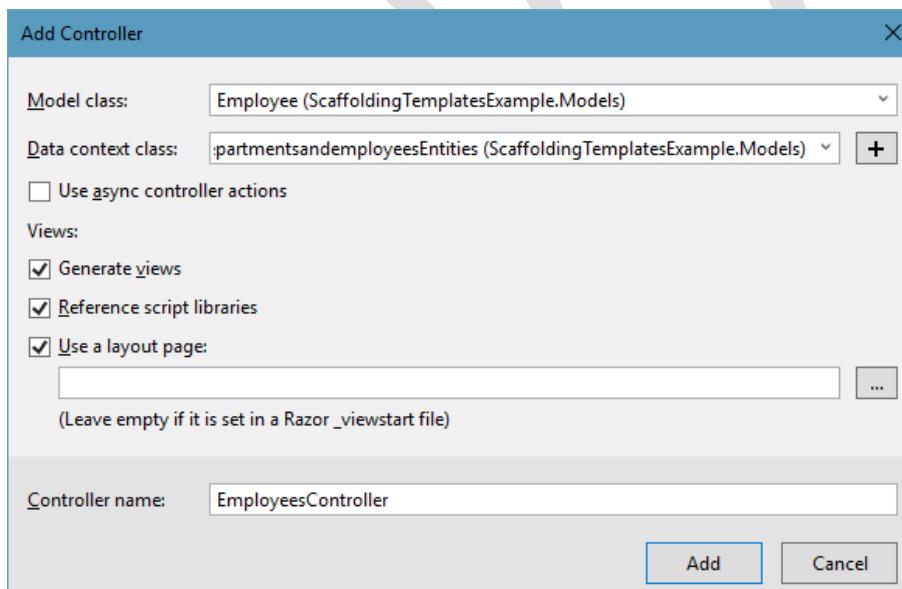
- Right click on "Controllers" folder and click on "Add" - "Controller".



- Select "MVC 5 Controller with views, using Entity Framework". Click on "Add".



- Select the Model class as "Employee (ScaffoldingTemplatesExample.Models)".
- Select the Data context class as "departmentsandemployeesEntities (ScaffoldingTemplatesExample.Models)".
- Check the checkbox "Generate views".
- Check the checkbox "Reference script libraries".
- Check the checkbox "Use a layout page".
- Enter the Controller name as "EmployeesController".
- Click on "Add".



### **Running the application**

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Departments/Index".

Output:

http://localhost:portnumber/Departments/Index

The screenshot shows a browser window titled "Index - My ASP.NET Application". The address bar displays "localhost:50151/Departments/Index". The page has a dark header bar with the text "Application name". Below it, the word "Index" is prominently displayed in large white letters. A "Create New" link is visible. A table lists three department entries:

DeptName	Loc	
Acouting	New York	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Operations	New Delhi	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Sales	New Jersy	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

At the bottom of the page, a copyright notice reads "© 2018 - My ASP.NET Application".

http://localhost:portnumber/Employees/Index

The screenshot shows a browser window titled "Index - My ASP.NET Application". The address bar displays "localhost:50151/Employees/Index". The page has a dark header bar with the text "Application name". Below it, the word "Index" is prominently displayed in large white letters. A "Create New" link is visible. A table lists six employee entries:

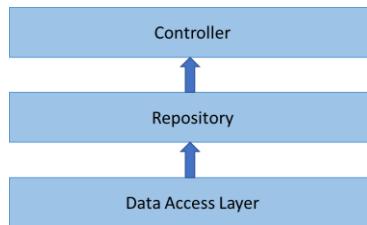
EmpName	Salary	DeptName	
Scott	2000.00	Acouting	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Allen	6500.00	Acouting	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Jones	4577.00	Operations	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
James	2500.00	Operations	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Smith	3345.00	Sales	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Harry	3600.00	Sales	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

At the bottom of the page, a copyright notice reads "© 2018 - My ASP.NET Application".

## REPOSITORY PATTERN

### What is Repository Pattern

- Repository sits between "Controller" and "DbContext". Controller calls Repository; Repository calls DbContext. Repository contains methods for CRUD operations. Repository makes it easy to create unit testing for controllers.



#### **Creating Repository Interface**

```

public interface IRepository
{
    List<Employee> GetAll();
    Employee GetById(int id);
    void Insert(Employee entity);
    void Update(Employee entity);
    void Delete(Employee entity);
}
  
```

#### **Creating Repository Class**

```

public class Repository : IRepository
{
    private DbContextClassname context;

    public Repository()
    {
        this.context = new DbContextClassname();
    }

    public List<Modelclass> GetAll()
    {
        return this.context.Dbsetname.ToList();
    }

    public Modelclass GetById(int id)
    {
        return this.context.Dbsetname.Find(id);
    }

    public void Insert(Modelclass entity)
    {
        this.context.Dbsetname.Add(entity);
        this.context.SaveChanges();
    }

    public void Update(Modelclass entity)
    {
        this.context.SaveChanges();
    }

    public void Delete(Modelclass entity)
    {
        this.context.Dbsetname.Remove(entity);
        this.context.SaveChanges();
    }
}
  
```

```

    }
}

```

### Repository - Example

#### **Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "RepositoryPatternExample". Type the location as "C:\Mvc". Type the solution name as "RepositoryPatternExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

#### **Creating Projects**

- Right click on the solution (RepositoryPatternExample) and click on "Add" - "New Project". Select ".NET Framework 4.7" - "Visual C#". Select "Class Library (.NET Framework)". Type the name as "DataAccessLayer". Click on "OK".
- Right click on the solution (RepositoryPatternExample) and click on "Add" - "New Project". Select ".NET Framework 4.7" - "Visual C#". Select "Class Library (.NET Framework)". Type the name as "RepositoryLayer". Click on "OK".

#### **Creating Database**

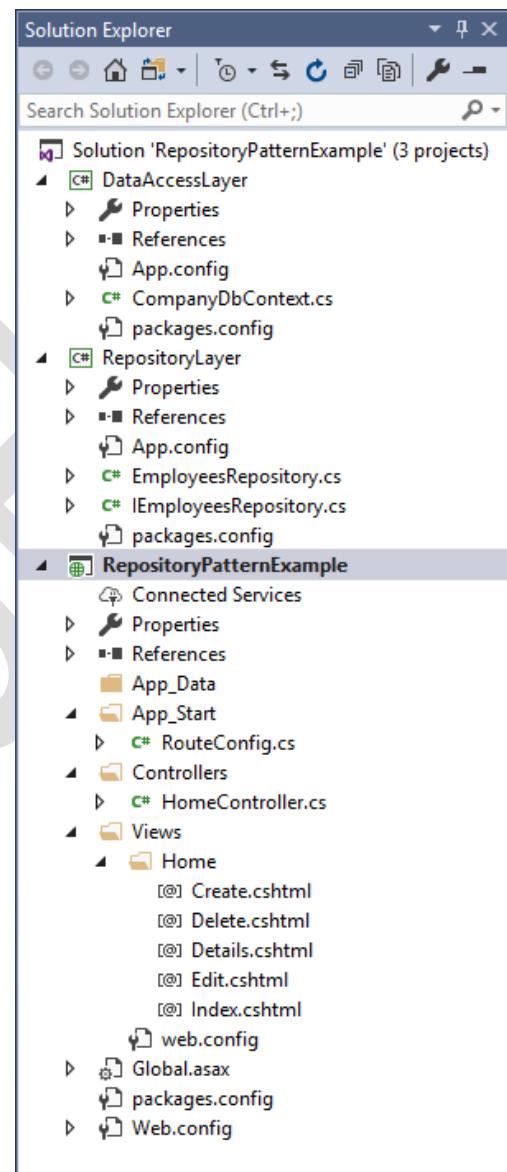
- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```

create database company
go
use company
go
create table Employees(EmpID int primary key, EmpName
nvarchar(max), Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go

```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.



#### **Installing Packages**

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Select "RepositoryPatternExample" project. Type the following command in "Package Manager Console" and press Enter.  
**install-package EntityFramework**
- Select "DataAccessLayer" project. Type the following command in "Package Manager Console" and press Enter.  
**install-package EntityFramework**
- Select "RepositoryLayer" project. Type the following command in "Package Manager Console" and press Enter.  
**install-package EntityFramework**

#### **Creating CompanyDbContext.cs**

- Right click on "DataAccessLayer" project and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "CompanyDbContext.cs". Click on "Add".

**Code for "CompanyDbContext.cs"**

```

using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Data.Entity;

namespace DataAccessLayer
{
    public class Employee
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }

        public DbSet<Employee> Employees { get; set; }
    }
}

```

**Creating IEmployeesRepository.cs**

- Right click on "DataAccessLayer" project and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "IEmployeesRepository.cs". Click on "Add".

**Code for "IEmployeesRepository.cs"**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using DataAccessLayer;

namespace RepositoryLayer
{
    public interface IEmployeesRepository
    {
        List<Employee> GetAll();
        Employee GetById(int id);
        void Insert(Employee entity);
        void Update(Employee entity);
        void Delete(Employee entity);
    }
}

```

**Creating EmployeesRepository.cs**

- Right click on "DataAccessLayer" project and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "EmployeesRepository.cs". Click on "Add".

**Code for "EmployeesRepository.cs"**

```

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

namespace RepositoryLayer
{

```

```

public class EmployeesRepository : IEmployeesRepository
{
    private CompanyDbContext context;

    public EmployeesRepository()
    {
        this.context = new CompanyDbContext();
    }

    public List<Employee> GetAll()
    {
        return this.context.Employees.ToList();
    }

    public Employee GetById(int id)
    {
        return this.context.Employees.Find(id);
    }

    public void Insert(Employee entity)
    {
        this.context.Employees.Add(entity);
        this.context.SaveChanges();
    }

    public void Update(Employee entity)
    {
        this.context.SaveChanges();
    }

    public void Delete(Employee entity)
    {
        this.context.Employees.Remove(entity);
        this.context.SaveChanges();
    }
}

```

#### **Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### **Code for "HomeController.cs"**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using DataAccessLayer;
using RepositoryLayer;

namespace RepositoryPatternExample.Controllers
{
    public class HomeController : Controller
    {
        IEmployeesRepository repository;

        public HomeController()
        {
            repository = new EmployeesRepository();
        }
    }
}

```

```

public ActionResult Index()
{
    List<Employee> emps = repository.GetAll();
    return View(emps);
}

public ActionResult Create()
{
    return View();
}

[HttpPost]
public ActionResult Create(Employee emp)
{
    repository.Insert(emp);
    return RedirectToAction("Index");
}

public ActionResult Details(int id)
{
    Employee emp = repository.GetById(id);
    return View(emp);
}

public ActionResult Edit(int id)
{
    Employee emp = repository.GetById(id);
    return View(emp);
}

[HttpPost]
public ActionResult Edit(Employee e)
{
    Employee emp = repository.GetById(e.EmpID);
    emp.EmpName = e.EmpName;
    emp.Salary = e.Salary;
    repository.Update(emp);
    return RedirectToAction("Index");
}

public ActionResult Delete(int id)
{
    Employee emp = repository.GetById(id);
    return View(emp);
}

[HttpPost]
public ActionResult Delete(int id, Employee e)
{
    Employee emp = repository.GetById(id);
    repository.Delete(emp);
    return RedirectToAction("Index");
}
}

```

#### **Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### **Code for "Views\Home\Index.cshtml"**

```
@model List<DataAccessLayer.Employee>
```

```

<html>
  <head>
    <title>Repository</title>
  </head>
  <body>
    <h1>Repository</h1>
    <table border="1">
      <tr>
        <th>Emp ID</th>
        <th>Emp Name</th>
        <th>Salary</th>
        <th>Options</th>
      </tr>
      @foreach (var item in Model)
      {
        <tr>
          <td>@item.EmpID</td>
          <td>@item.EmpName</td>
          <td>@item.Salary</td>
          <td>
            <a href="/Home/Details/@item.EmpID">Details</a>
            <a href="/Home/Edit/@item.EmpID">Edit</a>
            <a href="/Home/Delete/@item.EmpID">Delete</a>
          </td>
        </tr>
      }
    </table>
    <a href="/Home/Create">New Employee</a>
  </body>
</html>

```

#### Creating Details.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Details". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### Code for "Views\Home\Details.cshtml"

```

@model DataAccessLayer.Employee
<html>
  <head>
    <title>Details</title>
  </head>
  <body>
    <h1>Employees - Details</h1>
    <table>
      <tr>
        <td>Emp ID:</td>
        <td>@Model.EmpID</td>
      </tr>
      <tr>
        <td>Emp Name:</td>
        <td>@Model.EmpName</td>
      </tr>
      <tr>
        <td>Salary:</td>
        <td>@Model.Salary</td>
      </tr>
    </table>
    <a href="/Home/Index">Back to list</a>
  </body>
</html>

```

**[Creating Create.cshtml]**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Create". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Create.cshtml"**

```
<html>
<head>
<title>Create</title>
</head>
<body>
<h1>Employees - Create</h1>
<form action="/Home/Create" method="post">
<table>
<tr>
<td>Emp ID:</td>
<td><input type="text" name="EmpID" /></td>
</tr>
<tr>
<td>Emp Name:</td>
<td><input type="text" name="EmpName" /></td>
</tr>
<tr>
<td>Salary:</td>
<td><input type="text" name="Salary" /></td>
</tr>
<tr>
<td></td>
<td><input type="submit" value="Create" /></td>
</tr>
</table>
<a href="/Home/Index">Back to list</a>
</form>
</body>
</html>
```

**[Creating Edit.cshtml]**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Edit". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home>Edit.cshtml"**

```
@model DataAccessLayer.Employee
<html>
<head>
<title>Edit</title>
</head>
<body>
<h1>Employees - Edit</h1>
<form action="/Home/Edit" method="post">
<table>
<tr>
<td>Emp ID:</td>
<td><input type="text" name="EmpID" value="@Model.EmpID" readonly="readonly" /></td>
</tr>
<tr>
<td>Emp Name:</td>
<td><input type="text" name="EmpName" value="@Model.EmpName" /></td>
</tr>
<tr>
<td>Salary:</td>
<td><input type="text" name="Salary" value="@Model.Salary" /></td>
</tr>
<tr>
```

```

<td></td>
<td><input type="submit" value="Update" /></td>
</tr>
</table>
<a href="/Home/Index">Back to list</a>
</form>
</body>
</html>

```

#### Creating Delete.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Delete". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### Code for "Views\Home\Delete.cshtml"

```

@model DataAccessLayer.Employee
<html>
<head>
<title>Delete</title>
</head>
<body>
<h1>Employees - Delete</h1>
<form action="/Home/Delete/@Model.EmpID" method="post">
<h2>Are you sure to delete this employee?</h2>
<table>
<tr>
<td>Emp ID:</td>
<td>@Model.EmpID</td>
</tr>
<tr>
<td>Emp Name:</td>
<td>@Model.EmpName</td>
</tr>
<tr>
<td>Salary:</td>
<td>@Model.Salary</td>
</tr>
<tr>
<td></td>
<td><input type="submit" value="Delete" /></td>
</tr>
</table>
<a href="/Home/Index">Back to list</a>
</form>
</body>
</html>

```

#### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:

Emp ID	Emp Name	Salary	Options
1	Scott	4000	<a href="#">Details</a> <a href="#">Edit</a> <a href="#">Delete</a>
2	Allen	2500	<a href="#">Details</a> <a href="#">Edit</a> <a href="#">Delete</a>
3	Jones	5200	<a href="#">Details</a> <a href="#">Edit</a> <a href="#">Delete</a>
4	James	4400	<a href="#">Details</a> <a href="#">Edit</a> <a href="#">Delete</a>
5	Smith	7600	<a href="#">Details</a> <a href="#">Edit</a> <a href="#">Delete</a>

New Employee

## Generic Repository

- Generic Repository is a repository, which is common for all tables in the project.
- **Advantage:** We need not create one repository for one table. We can use same repository for all tables.
- **Drawback:** It is difficult to add methods that are specific to particular table.

### **Creating Generic Repository Interface**

```
public interface IRepository<T>
{
    List<T> GetAll();
    T GetById(int id);
    void Insert(T entity);
    void Update(T entity);
    void Delete(T entity);
}
```

### **Creating Generic Repository Class**

```
public class Repository<T> : IRepository<T> where T : class
{
    private IDbContext context;

    public Repository(IDbContext ctx)
    {
        this.context = ctx;
    }

    public List<T> GetAll()
    {
        return this.context.Set<T>().ToList();
    }

    public T GetById(int id)
    {
        return this.context.Set<T>().Find(id);
    }

    public void Insert(T entity)
    {
        this.context.Set<T>().Add(entity);
        this.context.SaveChanges();
    }

    public void Update(T entity)
    {
        this.context.SaveChanges();
    }

    public void Delete(T entity)
    {
        this.context.Set<T>().Remove(entity);
        this.context.SaveChanges();
    }
}
```

## Repository - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "RepositoryPatternExample". Type the location as "C:\Mvc". Type the solution name as "RepositoryPatternExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating Projects

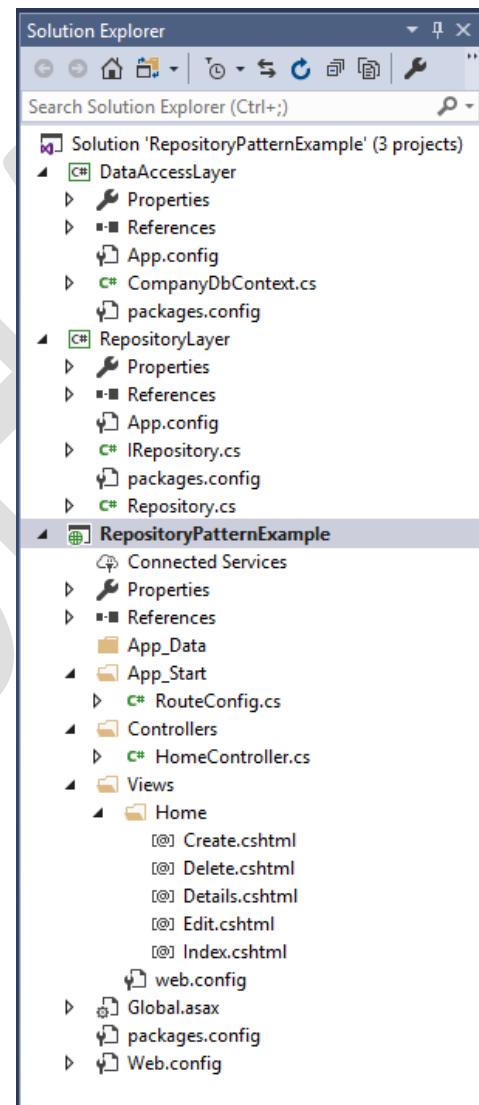
- Right click on the solution (RepositoryPatternExample) and click on "Add" - "New Project". Select ".NET Framework 4.7" - "Visual C#". Select "Class Library (.NET Framework)". Type the name as "DataAccessLayer". Click on "OK".
- Right click on the solution (RepositoryPatternExample) and click on "Add" - "New Project". Select ".NET Framework 4.7" - "Visual C#". Select "Class Library (.NET Framework)". Type the name as "RepositoryLayer". Click on "OK".

### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication".
- Click on "Connect".
- Ignore this step if "company" database already exists.
- Click on "New Query".
- Type the following code:

```
create database company
go
use company
go
create table Employees(
EmpID int primary key,
EmpName nvarchar(max),
Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.



### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Select "RepositoryPatternExample" project. Type the following command in "Package Manager Console" and press Enter.  
`install-package EntityFramework`
- Select "DataAccessLayer" project. Type the following command in "Package Manager Console" and press Enter.  
`install-package EntityFramework`
- Select "RepositoryLayer" project. Type the following command in "Package Manager Console" and press Enter.  
`install-package EntityFramework`

### Creating CompanyDbContext.cs

- Right click on "DataAccessLayer" project and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "CompanyDbContext.cs". Click on "Add".

#### Code for "CompanyDbContext.cs"

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
```

```

using System.Data.Entity;

namespace DataAccessLayer
{
    public class Employee
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public interface IDbContext
    {
        IDbSet<TEntity> Set<TEntity>() where TEntity : class;
        int SaveChanges();
    }

    public class CompanyDbContext : DbContext, IDbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }

        public DbSet<Employee> Employees { get; set; }

        public new IDbSet<TEntity> Set<TEntity>() where TEntity : class
        {
            return base.Set<TEntity>();
        }
    }
}

```

**Creating IRepository.cs**

- Right click on "DataAccessLayer" project and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "IRepository.cs". Click on "Add".

**Code for "IRepository.cs"**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using DataAccessLayer;

namespace RepositoryLayer
{
    public interface IRepository<T>
    {
        List<T> GetAll();
        T GetByid(int id);
        void Insert(T entity);
        void Update(T entity);
        void Delete(T entity);
    }
}

```

**Creating Repository.cs**

- Right click on "DataAccessLayer" project and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Repository.cs". Click on "Add".

**Code for "Repository.cs"**

```
using System;
```

```

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

namespace RepositoryLayer
{
    public class Repository<T> : IRepository<T> where T : class
    {
        private IDbContext context;

        public Repository(IDbContext ctx)
        {
            this.context = ctx;
        }

        public List<T> GetAll()
        {
            return this.context.Set<T>().ToList();
        }

        public T GetById(int id)
        {
            return this.context.Set<T>().Find(id);
        }

        public void Insert(T entity)
        {
            this.context.Set<T>().Add(entity);
            this.context.SaveChanges();
        }

        public void Update(T entity)
        {
            this.context.SaveChanges();
        }

        public void Delete(T entity)
        {
            this.context.Set<T>().Remove(entity);
            this.context.SaveChanges();
        }
    }
}

```

#### **Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### **Code for "HomeController.cs"**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using DataAccessLayer;
using RepositoryLayer;

namespace RepositoryPatternExample.Controllers
{
    public class HomeController : Controller
    {
}

```

```
IRepository<Employee> repository;

public HomeController()
{
    repository = new Repository<Employee>(new CompanyDbContext());
}

public ActionResult Index()
{
    List<Employee> emps = repository.GetAll();
    return View(emps);
}

public ActionResult Create()
{
    return View();
}

[HttpPost]
public ActionResult Create(Employee emp)
{
    repository.Insert(emp);
    return RedirectToAction("Index");
}

public ActionResult Details(int id)
{
    Employee emp = repository.GetById(id);
    return View(emp);
}

public ActionResult Edit(int id)
{
    Employee emp = repository.GetById(id);
    return View(emp);
}

[HttpPost]
public ActionResult Edit(Employee e)
{
    Employee emp = repository.GetById(e.EmpID);
    emp.EmpName = e.EmpName;
    emp.Salary = e.Salary;
    repository.Update(emp);
    return RedirectToAction("Index");
}

public ActionResult Delete(int id)
{
    Employee emp = repository.GetById(id);
    return View(emp);
}

[HttpPost]
public ActionResult Delete(int id, Employee e)
{
    Employee emp = repository.GetById(id);
    repository.Delete(emp);
    return RedirectToAction("Index");
}
```

```
}
```

#### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### Code for "Views\Home\Index.cshtml"

```
@model List<DataAccessLayer.Employee>
<html>
  <head>
    <title>Repository</title>
  </head>
  <body>
    <h1>Repository</h1>
    <table border="1">
      <tr>
        <th>Emp ID</th>
        <th>Emp Name</th>
        <th>Salary</th>
        <th>Options</th>
      </tr>
      @foreach (var item in Model)
      {
        <tr>
          <td>@item.EmpID</td>
          <td>@item.EmpName</td>
          <td>@item.Salary</td>
          <td>
            <a href="/Home/Details/@item.EmpID">Details</a>
            <a href="/Home/Edit/@item.EmpID">Edit</a>
            <a href="/Home/Delete/@item.EmpID">Delete</a>
          </td>
        </tr>
      }
    </table>
    <a href="/Home/Create">New Employee</a>
  </body>
</html>
```

#### Creating Details.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Details". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### Code for "Views\Home\Details.cshtml"

```
@model DataAccessLayer.Employee
<html>
  <head>
    <title>Details</title>
  </head>
  <body>
    <h1>Employees - Details</h1>
    <table>
      <tr>
        <td>Emp ID:</td>
        <td>@Model.EmpID</td>
      </tr>
      <tr>
        <td>Emp Name:</td>
        <td>@Model.EmpName</td>
      </tr>
      <tr>
        <td>Salary:</td>
      </tr>
    </table>
  </body>
</html>
```

```

<td>@Model.Salary</td>
</tr>
</table>
<a href="/Home/Index">Back to list</a>
</body>
</html>

```

#### Creating Create.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Create". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### Code for "Views\Home\Create.cshtml"

```

<html>
<head>
<title>Create</title>
</head>
<body>
<h1>Employees - Create</h1>
<form action="/Home/Create" method="post">
<table>
<tr>
<td>Emp ID:</td>
<td><input type="text" name="EmpID" /></td>
</tr>
<tr>
<td>Emp Name:</td>
<td><input type="text" name="EmpName" /></td>
</tr>
<tr>
<td>Salary:</td>
<td><input type="text" name="Salary" /></td>
</tr>
<tr>
<td></td>
<td><input type="submit" value="Create" /></td>
</tr>
</table>
<a href="/Home/Index">Back to list</a>
</form>
</body>
</html>

```

#### Creating Edit.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Edit". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### Code for "Views\Home>Edit.cshtml"

```

@model DataAccessLayer.Employee
<html>
<head>
<title>Edit</title>
</head>
<body>
<h1>Employees - Edit</h1>
<form action="/Home/Edit" method="post">
<table>
<tr>
<td>Emp ID:</td>
<td><input type="text" name="EmpID" value="@Model.EmpID" readonly="readonly" /></td>
</tr>
<tr>
<td>Emp Name:</td>

```

```

<td><input type="text" name="EmpName" value="@Model.EmpName" /></td>
</tr>
<tr>
<td>Salary:</td>
<td><input type="text" name="Salary" value="@Model.Salary" /></td>
</tr>
<tr>
<td></td>
<td><input type="submit" value="Update" /></td>
</tr>
</table>
<a href="/Home/Index">Back to list</a>
</form>
</body>
</html>

```

#### Creating Delete.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Delete". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### Code for "Views\Home\Delete.cshtml"

```

@model DataAccessLayer.Employee
<html>
<head>
<title>Delete</title>
</head>
<body>
<h1>Employees - Delete</h1>
<form action="/Home/Delete/@Model.EmpID" method="post">
<h2>Are you sure to delete this employee?</h2>
<table>
<tr>
<td>Emp ID:</td>
<td>@Model.EmpID</td>
</tr>
<tr>
<td>Emp Name:</td>
<td>@Model.EmpName</td>
</tr>
<tr>
<td>Salary:</td>
<td>@Model.Salary</td>
</tr>
<tr>
<td></td>
<td><input type="submit" value="Delete" /></td>
</tr>
</table>
<a href="/Home/Index">Back to list</a>
</form>
</body>
</html>

```

#### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:

Emp ID	Emp Name	Salary	Options
1	Scott	4000	<a href="#">Details</a> <a href="#">Edit</a> <a href="#">Delete</a>
2	Allen	2500	<a href="#">Details</a> <a href="#">Edit</a> <a href="#">Delete</a>
3	Jones	5200	<a href="#">Details</a> <a href="#">Edit</a> <a href="#">Delete</a>
4	James	4400	<a href="#">Details</a> <a href="#">Edit</a> <a href="#">Delete</a>
5	Smith	7600	<a href="#">Details</a> <a href="#">Edit</a> <a href="#">Delete</a>

New Employee

## DEPENDENCY INJECTION

### What is Dependency Injection

- Dependency Injection is a concept of creating object of repository and injecting the same into the controller, through constructor.
- Advantage:** The controller becomes independent of the repository, so we can perform unit testing on controller without repository also.



#### Syntax for Dependency Injection in "Unity.Mvc" Package

```
container.RegisterType< IRepository, Repository>();
```

#### Syntax of Dependency Injection in Controller

```
public class Controllername : Controller
{
    IRepository repository;

    public Controllername(IRepository repo)
    {
        this.repository = repo;
    }
}
```

## Dependency Injection - Example

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "RepositoryPatternExample". Type the location as "C:\Mvc". Type the solution name as "RepositoryPatternExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

#### Creating Projects

- Right click on the solution (RepositoryPatternExample) and click on "Add" - "New Project". Select ".NET Framework 4.7" - "Visual C#". Select "Class Library (.NET Framework)". Type the name as "DataAccessLayer". Click on "OK".

- Right click on the solution (RepositoryPatternExample) and click on "Add" - "New Project". Select ".NET Framework 4.7" - "Visual C#". Select "Class Library (.NET Framework)". Type the name as "RepositoryLayer". Click on "OK".

### **Creating Database**

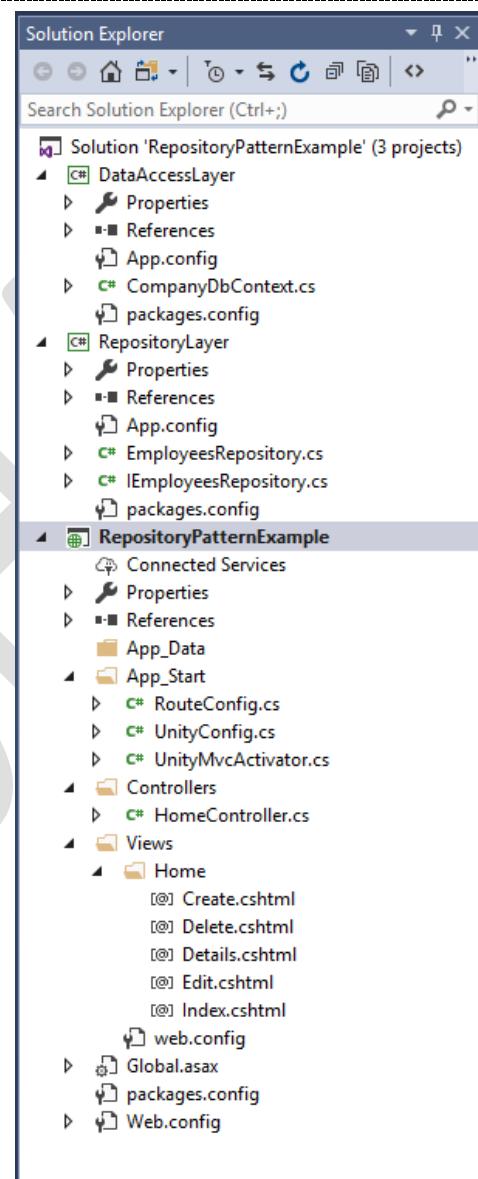
- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```

create database company
go
use company
go
create table Employees(
EmpID int primary key,
EmpName nvarchar(max),
Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go

```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.



### **Installing Packages**

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Select "RepositoryPatternExample" project. Type the following command in "Package Manager Console" and press Enter.  
`install-package EntityFramework`  
`install-package Unity.Mvc`
- Select "DataAccessLayer" project. Type the following command in "Package Manager Console" and press Enter.  
`install-package EntityFramework`
- Select "RepositoryLayer" project. Type the following command in "Package Manager Console" and press Enter.  
`install-package EntityFramework`

### **Creating CompanyDbContext.cs**

- Right click on "DataAccessLayer" project and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "CompanyDbContext.cs". Click on "Add".

### **Code for "CompanyDbContext.cs"**

```

using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Data.Entity;

namespace DataAccessLayer
{
    public class Employee
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }
}

```

```
public class CompanyDbContext : DbContext
{
    public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
    {
    }
    public DbSet<Employee> Employees { get; set; }

    public new IDbSet< TEntity > Set< TEntity >() where TEntity : class
    {
        return base.Set< TEntity >();
    }
}
```

**Creating IEmployeesRepository.cs**

- Right click on "DataAccessLayer" project and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "IEmployeesRepository.cs". Click on "Add".

**Code for "IEmployeesRepository.cs"**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using DataAccessLayer;

namespace RepositoryLayer
{
    public interface IEmployeesRepository
    {
        List<Employee> GetAll();
        Employee GetById(int id);
        void Insert(Employee entity);
        void Update(Employee entity);
        void Delete(Employee entity);
    }
}
```

**Creating EmployeesRepository.cs**

- Right click on "DataAccessLayer" project and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "EmployeesRepository.cs". Click on "Add".

**Code for "EmployeesRepository.cs"**

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

namespace RepositoryLayer
{
    public class EmployeesRepository : IEmployeesRepository
    {
        private CompanyDbContext context;

        public EmployeesRepository()
        {
            this.context = new CompanyDbContext();
        }

        public List<Employee> GetAll()
        {
```

```

        return this.context.Employees.ToList();
    }

    public Employee GetById(int id)
    {
        return this.context.Employees.Find(id);
    }

    public void Insert(Employee entity)
    {
        this.context.Employees.Add(entity);
        this.context.SaveChanges();
    }

    public void Update(Employee entity)
    {
        this.context.SaveChanges();
    }

    public void Delete(Employee entity)
    {
        this.context.Employees.Remove(entity);
        this.context.SaveChanges();
    }
}
}
}

```

#### **Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### **Code for "HomeController.cs"**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using DataAccessLayer;
using RepositoryLayer;

namespace RepositoryPatternExample.Controllers
{
    public class HomeController : Controller
    {
        IEmployeesRepository repository;

        public HomeController(IEmployeesRepository repo)
        {
            this.repository = repo;
        }

        public ActionResult Index()
        {
            List<Employee> emps = repository.GetAll();
            return View(emps);
        }

        public ActionResult Create()
        {
            return View();
        }
    }
}

```

```

[HttpPost]
public ActionResult Create(Employee emp)
{
    repository.Insert(emp);
    return RedirectToAction("Index");
}

public ActionResult Details(int id)
{
    Employee emp = repository.GetById(id);
    return View(emp);
}

public ActionResult Edit(int id)
{
    Employee emp = repository.GetById(id);
    return View(emp);
}

[HttpPost]
public ActionResult Edit(Employee e)
{
    Employee emp = repository.GetById(e.EmpID);
    emp.EmpName = e.EmpName;
    emp.Salary = e.Salary;
    repository.Update(emp);
    return RedirectToAction("Index");
}

public ActionResult Delete(int id)
{
    Employee emp = repository.GetById(id);
    return View(emp);
}

[HttpPost]
public ActionResult Delete(int id, Employee e)
{
    Employee emp = repository.GetById(id);
    repository.Delete(emp);
    return RedirectToAction("Index");
}
}

```

**Code for UnityConfig.cs**

```

using System;
using RepositoryLayer;
using Unity;

namespace RepositoryPatternExample
{

    public static class UnityConfig
    {
        private static Lazy<IUnityContainer> container =
            new Lazy<IUnityContainer>(() =>
        {
            var container = new UnityContainer();
            RegisterTypes(container);
            return container;
        });
    }
}

```

```

    });
    public static IUnityContainer Container => container.Value;

    public static void RegisterTypes(IUnityContainer container)
    {
        container.RegisterType<IEmployeesRepository, EmployeesRepository>();
    }
}
}

```

#### **Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### **Code for "Views\Home\Index.cshtml"**

```

@model List<DataAccessLayer.Employee>
<html>
<head>
    <title>Repository</title>
</head>
<body>
    <h1>Repository</h1>
    <table border="1">
        <tr>
            <th>Emp ID</th>
            <th>Emp Name</th>
            <th>Salary</th>
            <th>Options</th>
        </tr>
        @foreach (var item in Model)
        {
            <tr>
                <td>@item.EmpID</td>
                <td>@item.EmpName</td>
                <td>@item.Salary</td>
                <td>
                    <a href="/Home/Details/@item.EmpID">Details</a>
                    <a href="/Home/Edit/@item.EmpID">Edit</a>
                    <a href="/Home/Delete/@item.EmpID">Delete</a>
                </td>
            </tr>
        }
    </table>
    <a href="/Home/Create">New Employee</a>
</body>
</html>

```

#### **Creating Details.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Details". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### **Code for "Views\Home\Details.cshtml"**

```

@model DataAccessLayer.Employee
<html>
<head>
    <title>Details</title>
</head>
<body>
    <h1>Employees - Details</h1>
    <table>
        <tr>
            <td>Emp ID:</td>

```

```

<td>@Model.EmpID</td>
</tr>
<tr>
<td>Emp Name:</td>
<td>@Model.EmpName</td>
</tr>
<tr>
<td>Salary:</td>
<td>@Model.Salary</td>
</tr>
</table>
<a href="/Home/Index">Back to list</a>
</body>
</html>

```

#### Creating Create.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Create". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### Code for "Views\Home\Create.cshtml"

```

<html>
<head>
<title>Create</title>
</head>
<body>
<h1>Employees - Create</h1>
<form action="/Home/Create" method="post">
<table>
<tr>
<td>Emp ID:</td>
<td><input type="text" name="EmpID" /></td>
</tr>
<tr>
<td>Emp Name:</td>
<td><input type="text" name="EmpName" /></td>
</tr>
<tr>
<td>Salary:</td>
<td><input type="text" name="Salary" /></td>
</tr>
<tr>
<td></td>
<td><input type="submit" value="Create" /></td>
</tr>
</table>
<a href="/Home/Index">Back to list</a>
</form>
</body>
</html>

```

#### Creating Edit.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Edit". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### Code for "Views\Home>Edit.cshtml"

```

@model DataAccessLayer.Employee
<html>
<head>
<title>Edit</title>
</head>
<body>
<h1>Employees - Edit</h1>

```

```

<form action="/Home/Edit" method="post">
<table>
<tr>
<td>Emp ID:</td>
<td><input type="text" name="EmpID" value="@Model.EmpID" readonly="readonly" /></td>
</tr>
<tr>
<td>Emp Name:</td>
<td><input type="text" name="EmpName" value="@Model.EmpName" /></td>
</tr>
<tr>
<td>Salary:</td>
<td><input type="text" name="Salary" value="@Model.Salary" /></td>
</tr>
<tr>
<td></td>
<td><input type="submit" value="Update" /></td>
</tr>
</table>
<a href="/Home/Index">Back to list</a>
</form>
</body>
</html>

```

#### **Creating Delete.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Delete". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### **Code for "Views\Home\Delete.cshtml"**

```

@model DataAccessLayer.Employee
<html>
<head>
<title>Delete</title>
</head>
<body>
<h1>Employees - Delete</h1>
<form action="/Home/Delete/@Model.EmpID" method="post">
<h2>Are you sure to delete this employee?</h2>
<table>
<tr>
<td>Emp ID:</td>
<td>@Model.EmpID</td>
</tr>
<tr>
<td>Emp Name:</td>
<td>@Model.EmpName</td>
</tr>
<tr>
<td>Salary:</td>
<td>@Model.Salary</td>
</tr>
<tr>
<td></td>
<td><input type="submit" value="Delete" /></td>
</tr>
</table>
<a href="/Home/Index">Back to list</a>
</form>
</body>
</html>

```

### **Running the application**

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:

Emp ID	Emp Name	Salary	Options
1	Scott	4000	<a href="#">Details</a> <a href="#">Edit</a> <a href="#">Delete</a>
2	Allen	2500	<a href="#">Details</a> <a href="#">Edit</a> <a href="#">Delete</a>
3	Jones	5200	<a href="#">Details</a> <a href="#">Edit</a> <a href="#">Delete</a>
4	James	4400	<a href="#">Details</a> <a href="#">Edit</a> <a href="#">Delete</a>
5	Smith	7600	<a href="#">Details</a> <a href="#">Edit</a> <a href="#">Delete</a>

[New Employee](#)

## AJAX

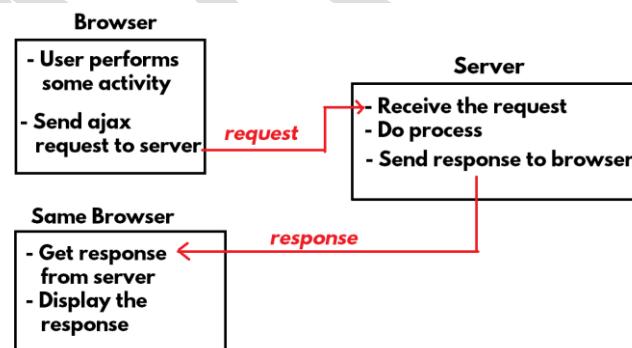
### Introduction to AJAX

- AJAX stands for "Asynchronous JavaScript And Xml".
- AJAX is not a language; but AJAX is a "concept", which is used to "send background request from browser to server" and also "get background response from server to browser", without refreshing the web page in the browser.
- Examples of AJAX usage: Google search, Gmail username checking, Facebook like button, Facebook share button, Facebook comments, IRCTC search trains etc.

Asynchronous = Background

JavaScript = Using JavaScript only, we can implement AJAX

XML = Data exchange format



### Advantages of AJAX

- Executes faster
- Less burden on browser (client) and server
- Better user experience.

### Types of AJAX Request

- Get : Used to retrieve / search data from server
- Post : Used to insert data to server.
- Put : Used to update data on server.
- Delete : Used to delete data from server

### jQuery AJAX

- "jQuery" is a "JavaScript library", which is a collection of pre-defined functions to perform DOM manipulations easily. "jQuery" provides a set of methods to perform "AJAX" easily. "jQuery AJAX" internally uses "JavaScript AJAX".
- In order to use "jQuery AJAX", we have to import the jquery library file. Ex: jquery-3.3.1.js

#### Syntax of jQuery AJAX

```
$ajax({ url: "server url", method: "GET | POST | PUT | DELETE", success: functionname, error: functionname });
```

- **url:** address of server program, to which you want to send request
- **method:** GET | POST
  - GET: Retrieving / searching
  - POST: Inserting
  - PUT: Updating
  - Delete: Deleting
- **Success:** Represents the function, which will be called automatically when the browser receives response successfully.
- **Error:** Represents the function, which will be called automatically when the browser receives some error as response.

### jQuery AJAX - Simple - Example

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "jQueryAJAXSimpleExample". Type the location as "C:\Mvc". Type the solution name as "jQueryAJAXSimpleExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

#### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Type the following command in "Package Manager Console" and press Enter.  
`install-package jQuery`

#### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;

namespace jQueryAJAXSimpleExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        public string Sample()
```

```

    {
        return "Hello";
    }
}
}

```

#### Creating Index.cshtml

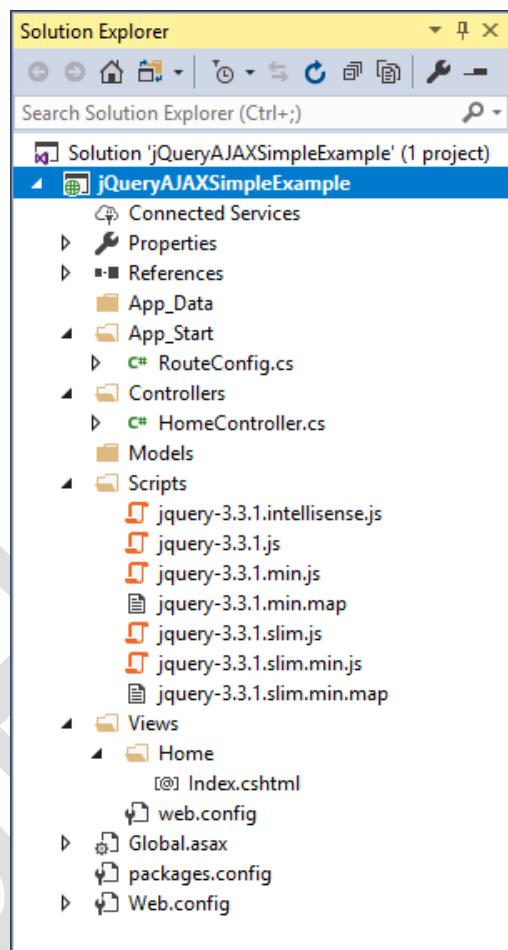
- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### Code for "Views\Home\Index.cshtml"

```

<html>
    <head>
        <title>jQuery AJAX - Simple</title>
    </head>
    <body>
        <h1>jQuery AJAX - Simple</h1>
        <input type="submit" id="btn1" value="Send AJAX Request" />
        <div id="div1"></div>
        <script src="~/Scripts/jquery-3.3.1.js"></script>
        <script>
            $("#btn1").click(fun1);
            function fun1(event)
            {
                event.preventDefault();
                $.ajax({url: "/Home/Sample", type: "GET", success: fun2, error: fun3 });
            }
            function fun2(response)
            {
                $("#div1").append("<p>" + response + "</p>");
            }
            function fun3(xhr)
            {
                alert(xhr.responseText);
            }
        </script>
    </body>
</html>

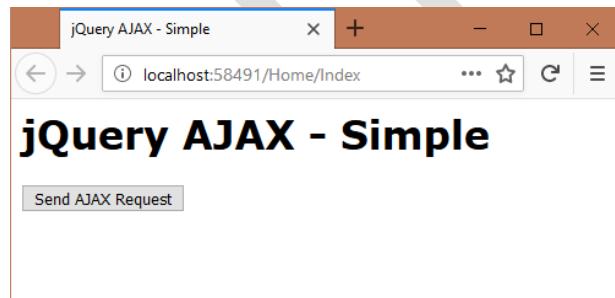
```



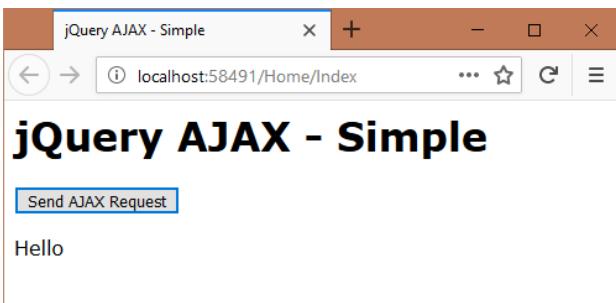
#### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



Click on "Send AJAX Request".



## jQuery AJAX - Get - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "jQueryAJAXGetExample". Type the location as "C:\Mvc". Type the solution name as "jQueryAJAXGetExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

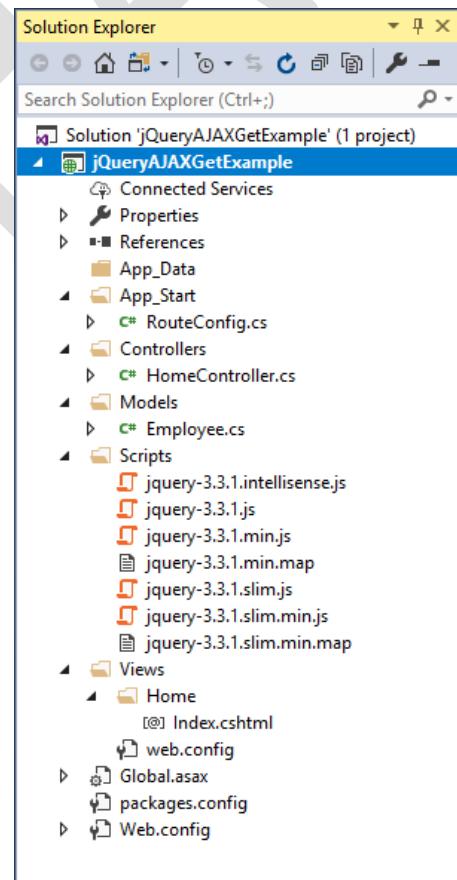
### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query".

- Type the following code:

```
create database company
go
use company
go
create table Employees(
EmpID int primary key,
EmpName nvarchar(max),
Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.



### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
install-package jQuery
```

### Creating Employee.cs

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

#### Code for "Employee.cs"

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;

namespace jQueryAJAXGetExample.Models
{
```

```

public class Employee
{
    [Key]
    public int EmpID { get; set; }
    public string EmpName { get; set; }
    public decimal Salary { get; set; }
}

public class CompanyDbContext : DbContext
{
    public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
    {
    }

    public DbSet<Employee> Employees { get; set; }
}

```

#### **Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### **Code for "HomeController.cs"**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Mvc;
using jQueryAJAXGetExample.Models;

namespace jQueryAJAXGetExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        [HttpGet]
        public ActionResult GetEmp()
        {
            CompanyDbContext db = new CompanyDbContext();
            List<Employee> emps = db.Employees.ToList();
            return Json(emps, JsonRequestBehavior.AllowGet);
        }
    }
}

```

#### **Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### **Code for "Views\Home\Index.cshtml"**

```

<html>
  <head>
    <title>jQuery AJAX - Get</title>
  </head>
  <body>
    <h1>jQuery AJAX - Get</h1>
    <form>
      <input type="button" id="button1" value="Get data" />
      <table id="table1" border="1"></table>
    </form>

```

```

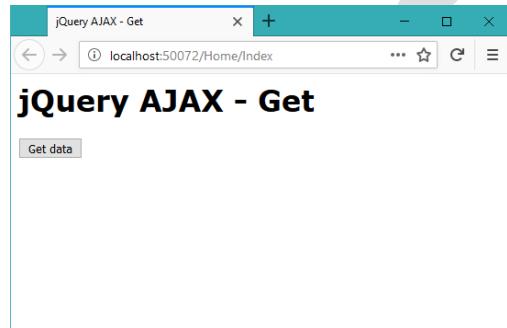
<script src="~/Scripts/jquery-3.3.1.js"></script>
<script>
  $("#button1").click(fun1);
  function fun1()
  {
    $.ajax({ url: "/Home/GetEmp", type: "GET", success: fun2, error: fun3 });
  }
  function fun2(response)
  {
    $("#table1").html("<tr><th>Emp ID</th><th>Emp Name</th><th>Salary</th></tr>");
    for (var i = 0; i < response.length; i++)
    {
      $("#table1").append("<tr><td>" + response[i].EmpID + "</td><td>" + response[i].EmpName + "</td><td>" +
      response[i].Salary + "</td></tr>");
    }
  }
  function fun3(xhr)
  {
    alert(xhr.responseText);
  }
</script>
</body>
</html>

```

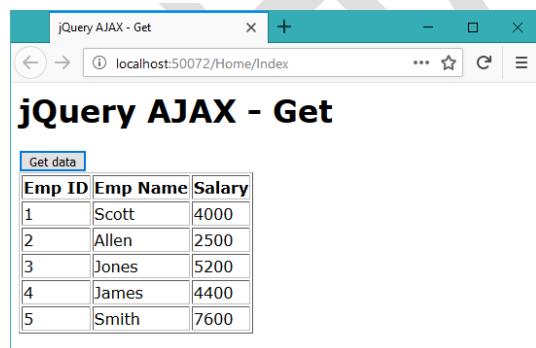
#### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



Click on "Get data".



### jQuery AJAX - Search - Example

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "jQueryAJAXSearchExample". Type the location as "C:\Mvc". Type the solution name as "jQueryAJAXSearchExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### **Creating Database**

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```

create database company
go
use company
go
create table Employees(
EmpID int primary key,
EmpName nvarchar(max),
Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go

```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### **Installing Packages**

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Type the following command in "Package Manager Console" and press Enter.  
**install-package EntityFramework  
install-package jQuery**

### **Creating Employee.cs**

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".
- Code for "Employee.cs"**
- ```

using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;

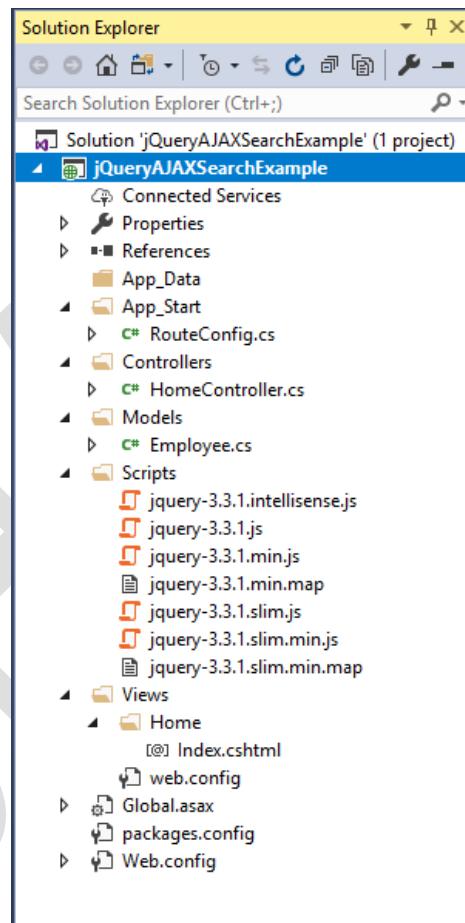
namespace jQueryAJAXSearchExample.Models
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }
        public DbSet<Employee> Employees { get; set; }
    }
}

```

### **Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".



**Code for "HomeController.cs"**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Mvc;
using jQueryAJAXSearchExample.Models;

namespace jQueryAJAXSearchExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        [HttpGet]
        public ActionResult SearchEmp(string str)
        {
            CompanyDbContext db = new CompanyDbContext();
            List<Employee> emps = db.Employees.Where(temp => temp.EmpName.Contains(str)).ToList();
            return Json(emps, JsonRequestBehavior.AllowGet);
        }
    }
}

```

**Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Index.cshtml"**

```

<html>
  <head>
    <title>jQuery AJAX - Search</title>
  </head>
  <body>
    <h1>jQuery AJAX - Search</h1>
    <form>
      Search: <input type="text" id="txt1" placeholder="Search" />
      <input type="submit" id="button1" value="Search" />
      <table id="table1" border="1"></table>
    </form>
    <script src="~/Scripts/jquery-3.3.1.js"></script>
    <script>
      $("#button1").click(fun1);
      function fun1(event)
      {
        event.preventDefault();
        $.ajax({ url: "/Home/SearchEmp?str=" + $("#txt1").val(), type: "GET", success: fun2, error: fun3 });
      }
      function fun2(response)
      {
        $("#table1").html("<tr><th>Emp ID</th><th>Emp Name</th><th>Salary</th></tr>");
        for (var i = 0; i < response.length; i++)
        {
          $("#table1").append("<tr><td>" + response[i].EmpID + "</td><td>" + response[i].EmpName + "</td><td>" +
response[i].Salary + "</td></tr>");
        }
      }
      function fun3(xhr)
      {
        alert(xhr.responseText);
      }
    </script>

```

```

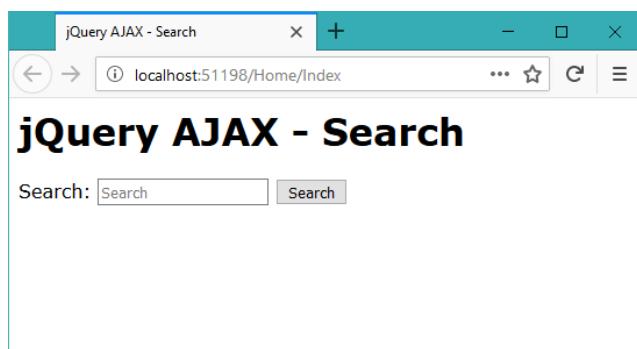
        }
    </script>
</body>
</html>

```

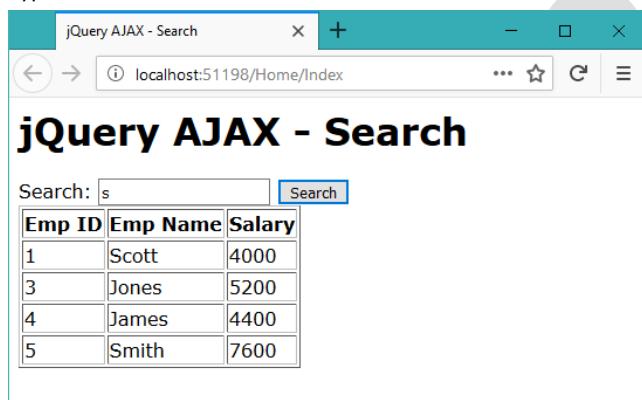
#### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



Type some text and click on "Search".



## jQuery AJAX - Post - Example

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "jQueryAJAXPostExample". Type the location as "C:\Mvc". Type the solution name as "jQueryAJAXPostExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

#### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```

create database company
go
use company
go
create table Employees(
    EmpID int primary key,
    EmpName nvarchar(max),
    Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)

```

```

insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go

```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### **Installing Packages**

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Type the following command in "Package Manager Console" and press Enter.

```

install-package EntityFramework
install-package jQuery

```

### **Creating Employee.cs**

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

#### **Code for "Employee.cs"**

```

using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Data.Entity;

namespace jQueryAJAXPostExample.Models
{
    public class Employee
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data
source=localhost\sqlexpress; integrated security=yes; initial
catalog=company")
        {
        }
        public DbSet<Employee> Employees { get; set; }
    }
}

```

### **Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

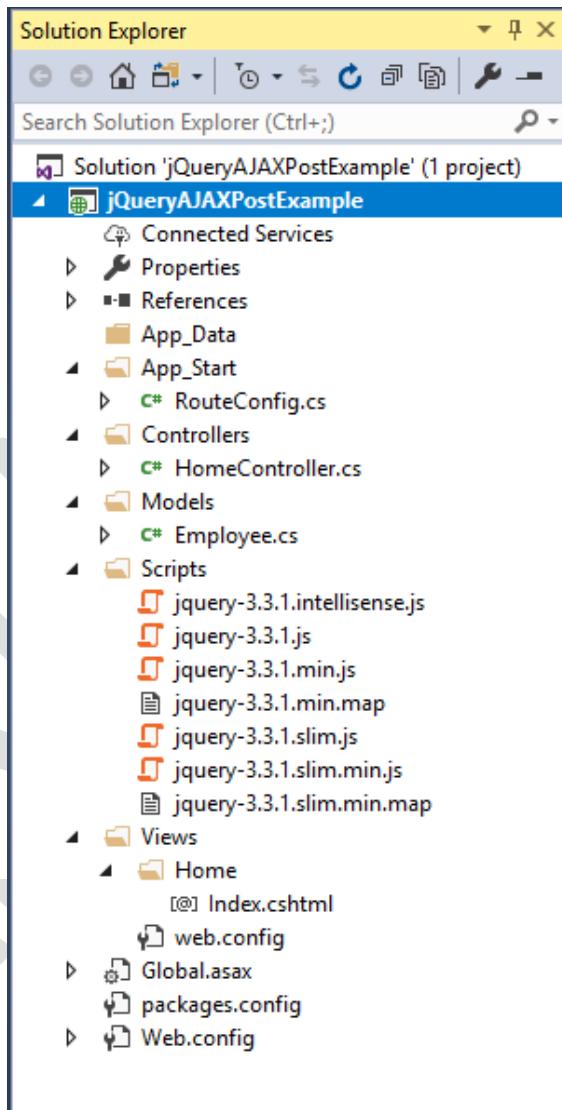
#### **Code for "HomeController.cs"**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Mvc;
using jQueryAJAXPostExample.Models;

namespace jQueryAJAXPostExample.Controllers
{
    public class HomeController : Controller

```



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

    [HttpPost]
    public ActionResult InsertEmp(Employee emp)
    {
        CompanyDbContext db = new CompanyDbContext();
        db.Employees.Add(emp);
        db.SaveChanges();
        return Content("Successfully Inserted");
    }
}
```

**Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Index.cshtml"**

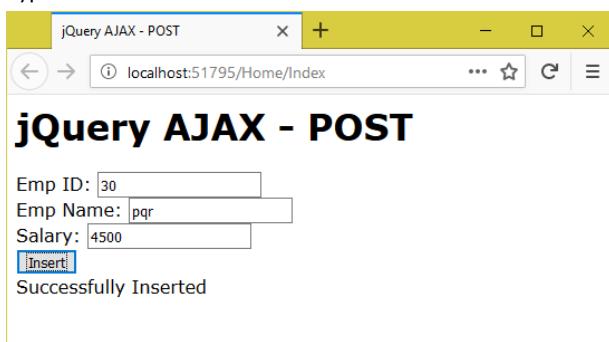
```
<html>
    <head>
        <title>jQuery AJAX - POST</title>
    </head>
    <body>
        <h1>jQuery AJAX - POST</h1>
        <form>
            Emp ID: <input type="text" id="txt1" /><br />
            Emp Name: <input type="text" id="txt2" /><br />
            Salary: <input type="text" id="txt3" /><br />
            <input type="submit" id="button1" value="Insert" />
            <div id="div1"></div>
        </form>
        <script src="~/Scripts/jquery-3.3.1.js"></script>
        <script>
            $("#button1").click(fun1);
            function fun1(event)
            {
                event.preventDefault();
                var mydata = { "EmpID":$("#txt1").val(), "EmpName":$("#txt2").val(), "Salary":$("#txt3").val() };
                $.ajax({ url: "/Home/InsertEmp", type: "POST", data: mydata, success: fun2, error: fun3 });
            }
            function fun2(response)
            {
                $("#div1").html(response);
            }
            function fun3(xhr)
            {
                alert(xhr.responseText);
            }
        </script>
    </body>
</html>
```

**Running the application**

- Press "F5" to run the application.
  - Type "http://localhost:portnumber/Home/Index".
- Output:



Type some details and click on "Insert".



## jQuery AJAX - Put - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "jQueryAJAXPutExample". Type the location as "C:\Mvc". Type the solution name as "jQueryAJAXPutExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```
create database company
go
use company
go
create table Employees(EmpID int primary key, EmpName nvarchar(max), Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
install-package jQuery
```

### **Creating Employee.cs**

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

### **Code for "Employee.cs"**

```

using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;

namespace jQueryAJAXPutExample.Models
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress;
integrated security=yes; initial catalog=company")
        {
        }

        public DbSet<Employee> Employees { get; set; }
    }
}

```

### **Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### **Code for "HomeController.cs"**

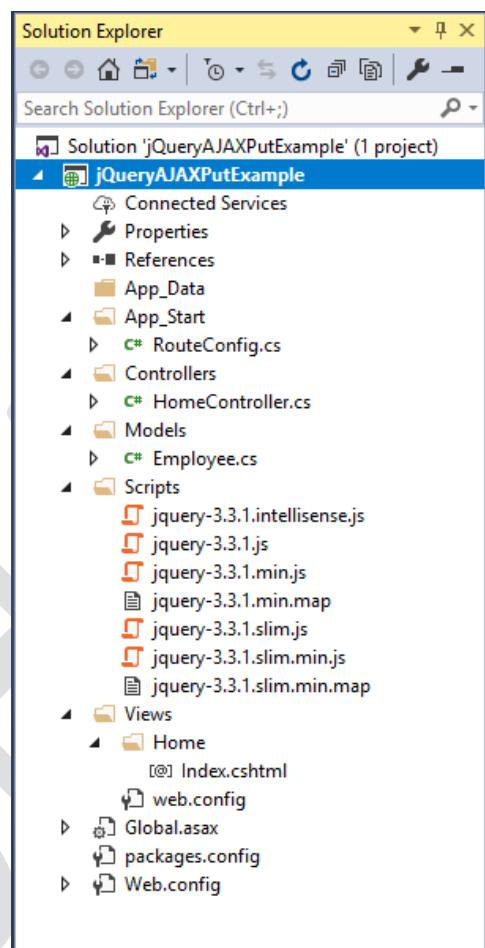
```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Mvc;
using jQueryAJAXPutExample.Models;

namespace jQueryAJAXPutExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        [HttpPost]
        public ActionResult UpdateEmp(Employee emp)
        {
            CompanyDbContext db = new CompanyDbContext();
            Employee e = db.Employees.Where(temp => temp.EmpID == emp.EmpID).FirstOrDefault();
            e.EmpName = emp.EmpName;
            e.Salary = emp.Salary;
            db.SaveChanges();
            return Content("Successfully Updated");
        }
    }
}

```



**Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Index.cshtml"**

```

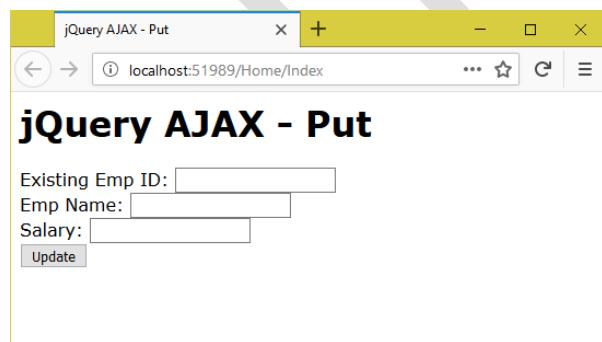
<html>
  <head>
    <title>jQuery AJAX - Put</title>
  </head>
  <body>
    <h1>jQuery AJAX - Put</h1>
    <form>
      Existing Emp ID: <input type="text" id="txt1" /><br />
      Emp Name: <input type="text" id="txt2" /><br />
      Salary: <input type="text" id="txt3" /><br />
      <input type="submit" id="button1" value="Update" />
    <div id="div1"></div>
    </form>
    <script src="~/Scripts/jquery-3.3.1.js"></script>
    <script>
      $("#button1").click(fun1);
      function fun1(event)
      {
        event.preventDefault();
        var mydata = { "EmpID": $("#txt1").val(), "EmpName": $("#txt2").val(), "Salary": $("#txt3").val() };
        $.ajax({ url: "/Home/UpdateEmp", type: "PUT", data: mydata, success: fun2, error: fun3 });
      }
      function fun2(response)
      {
        $("#div1").html(response);
      }
      function fun3(xhr)
      {
        alert(xhr.responseText);
      }
    </script>
  </body>
</html>

```

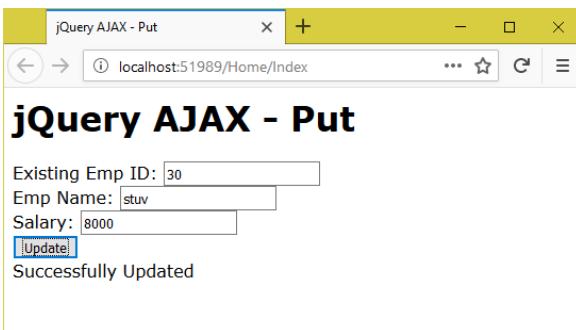
**Running the application**

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



Type some details and click on "Update".



## jQuery AJAX - Delete - Example

### **Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "jQueryAJAXDeleteExample". Type the location as "C:\Mvc". Type the solution name as "jQueryAJAXDeleteExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### **Creating Database**

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```
create database company
go
use company
go
create table Employees(
EmpID int primary key,
EmpName nvarchar(max),
Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### **Installing Packages**

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
install-package jQuery
```

### **Creating Employee.cs**

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

#### **Code for "Employee.cs"**

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;

namespace jQueryAJAXDeleteExample.Models
{
    public class Employee
```

```

{
    [Key]
    public int EmpID { get; set; }
    public string EmpName { get; set; }
    public decimal Salary { get; set; }
}

public class CompanyDbContext : DbContext
{
    public CompanyDbContext() : base(@"data source=localhost\sqlexpress;
integrated security=yes; initial catalog=company")
    {
    }

    public DbSet<Employee> Employees { get; set; }
}

```

#### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### Code for "HomeController.cs"

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Mvc;
using jQueryAJAXDeleteExample.Models;

namespace jQueryAJAXDeleteExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        [HttpDelete]
        public ActionResult DeleteEmp(int EmpIDToDelete)
        {
            CompanyDbContext db = new CompanyDbContext();
            Employee e = db.Employees.Where(temp => temp.EmpID == EmpIDToDelete).FirstOrDefault();
            db.Employees.Remove(e);
            db.SaveChanges();
            return Content("Successfully Deleted");
        }
    }
}

```

#### Creating Index.cshtml

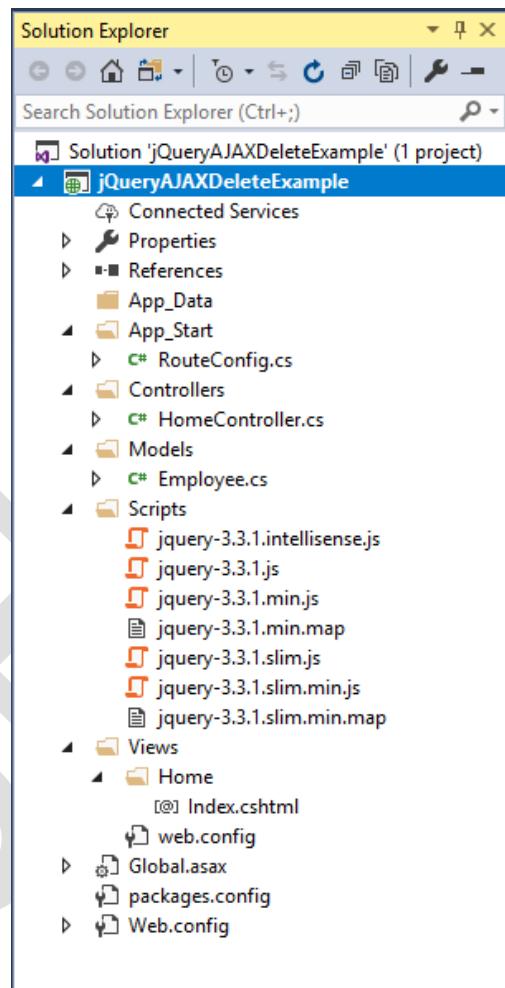
- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### Code for "Views\Home\Index.cshtml"

```

<html>
    <head>
        <title>jQuery AJAX - Delete</title>
    </head>
    <body>
        <h1>jQuery AJAX - Delete</h1>
        <form>

```



```

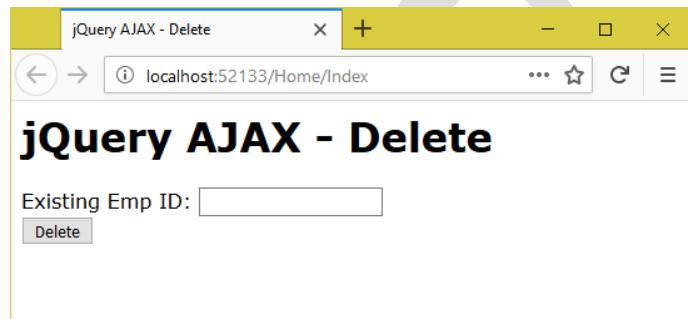
Existing Emp ID: <input type="text" id="txt1" /><br />
<input type="submit" id="button1" value="Delete" />
<div id="div1"></div>
</form>
<script src="~/Scripts/jquery-3.3.1.js"></script>
<script>
$("#button1").click(fun1);
function fun1(event)
{
event.preventDefault();
$.ajax({ url: "/Home/DeleteEmp?EmpIDToDelete=" + $("#txt1").val(), type: "DELETE", success: fun2, error: fun3 });
}
function fun2(response)
{
$("#div1").html(response);
}
function fun3(xhr)
{
alert(xhr.responseText);
}
</script>
</body>
</html>

```

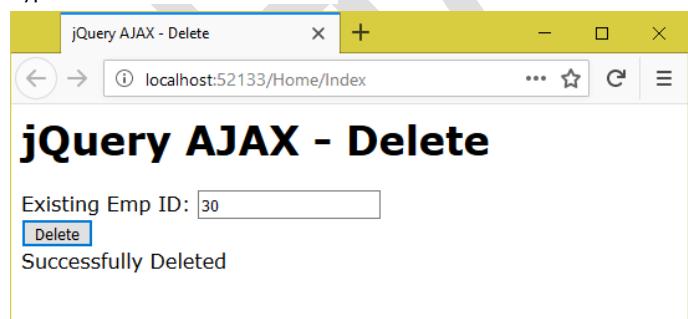
#### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



Type some id and click on "Insert".



## AngularJS AJAX

#### Introduction to AngularJS AJAX

- "AngularJS" is a "JavaScript library", which is used to develop "single page applications". "AngularJS" provides a set of methods to perform "AJAX" easily. "AngularJS AJAX" internally uses "JavaScript AJAX".
- In order to use "AngularJS AJAX", we have to import the angularjs library file. Ex: angular.js

### Syntax of AngularJS AJAX

- **GET:**

- Used to retrieve data from server.

Syntax: \$http.get("url").then(successcallback, errorcallback);

- **POST:**

- Used to insert data into database.

Syntax: \$http.post("url", object).then(successcallback, errorcallback);

- **PUT:**

- Used to update data in database.

Syntax: \$http.put("url", object).then(successcallback, errorcallback);

- **DELETE:**

- Used to delete data in database.

Syntax: \$http.delete("url").then(successcallback, errorcallback);

- ❖ **Success Callback:** Represents the function, which will be called automatically when the browser receives response successfully.
- ❖ **Error Callback:** Represents the function, which will be called automatically when the browser receives some error as response.

### AngularJS AJAX - Simple - Example

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "AngularJSAjaxSimple". Type the location as "C:\Mvc". Type the solution name as "AngularJSAjaxSimple". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

#### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

install-package angularjs

#### Creating HomeController.cs

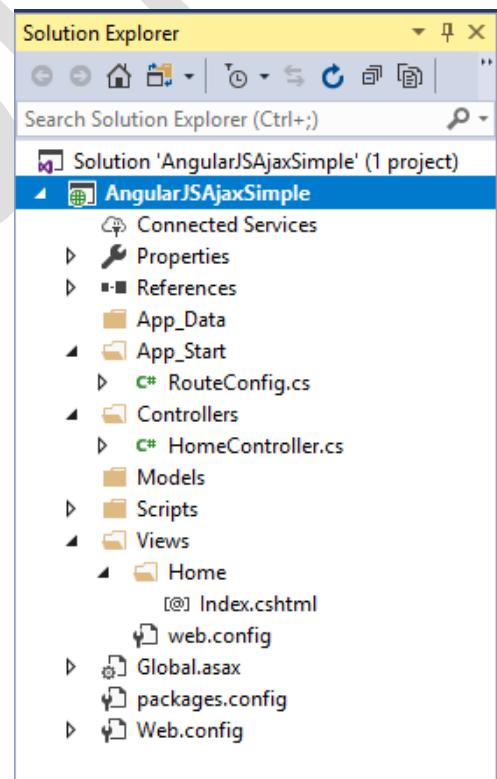
- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;

namespace AngularJSAjaxSimple.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        public ActionResult GetData()
        {
            return Content("Hello from Server at " + DateTime.Now);
        }
    }
}
```



**Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Index.cshtml"**

```

<html>
<head>
    <title>AngularJS AJAX - Simple</title>
    <script src="~/Scripts/angular.js"></script>
    <script>
        var app = angular.module("mymodule", []);
        app.controller("mycontroller", fum);
        function fum($scope, $http)
        {
            $scope.getdata = function ()
            {
                $http.get("/home/getdata").then(fun2, fun3);
            };
            function fun2(response)
            {
                $scope.message = response.data;
            }
            function fun3(err)
            {
                alert(JSON.stringify(err));
            }
        }
    </script>
</head>
<body>
    <div ng-app="mymodule" ng-controller="mycontroller">
        <h1>AngularJS AJAX - Simple</h1>
        <input type="button" ng-click="getdata()" value="Get data from server">
        <div>{{message}}</div>
    </div>
</body>
</html>

```

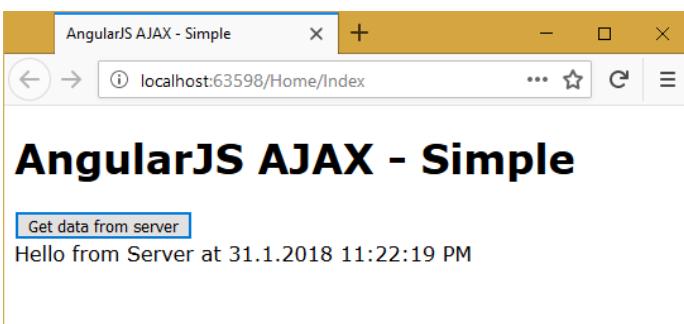
**Running the application**

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



Click on "Get data from server".



## AngularJS AJAX - Get - Example

### Creating Project

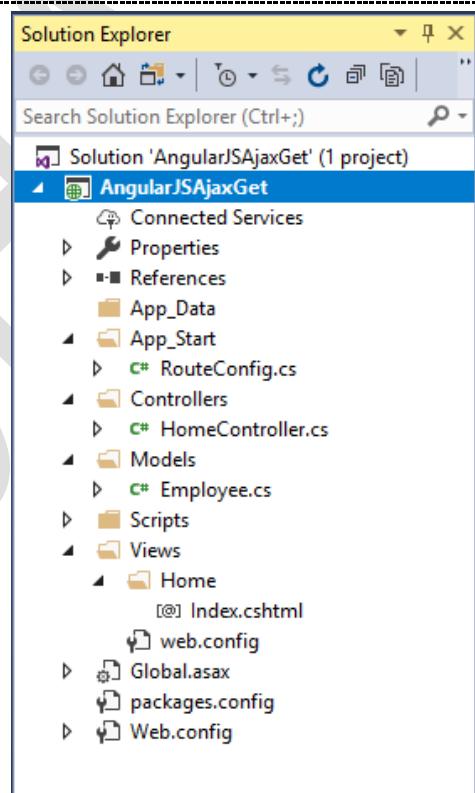
- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "AngularJSAjaxGet". Type the location as "C:\Mvc". Type the solution name as "AngularJSAjaxGet". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```
create database company
go
use company
go
create table Employees(EmpID int primary key, EmpName nvarchar(max),
Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.



### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.

- Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
install-package angularjs
```

### Creating Employee.cs

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

### Code for "Employee.cs"

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;

namespace AngularJSAjaxGet.Models
{
```

```

public class Employee
{
    [Key]
    public int EmpID { get; set; }
    public string EmpName { get; set; }
    public decimal Salary { get; set; }
}

public class CompanyDbContext : DbContext
{
    public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
    {
    }

    public DbSet<Employee> Employees { get; set; }
}

```

**Creating HomeController.cs**

- Right click on “Controllers” folder and click on “Add” - “Controller”. Select “MVC 5 Controller - Empty”. Click on “Add”. Type the controller name as “HomeController”. Click on “Add”.

**Code for “HomeController.cs”**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Mvc;
using AngularJSAjaxGet.Models;

namespace AngularJSAjaxGet.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        public ActionResult GetEmployees()
        {
            CompanyDbContext db = new CompanyDbContext();
            List<Employee> emps = db.Employees.ToList();
            return Json(emps, JsonRequestBehavior.AllowGet);
        }
    }
}

```

**Creating Index.cshtml**

- Right click on “Views\Home” folder and click on “Add” - “View”. Type the view name as “Index”. Select the template “Empty (without model)”. Uncheck all the checkboxes. Click on “Add”.

**Code for “Views\Home\Index.cshtml”**

```

<html>
<head>
    <title>AngularJS AJAX - Get</title>
    <script src="~/Scripts/angular.js"></script>
    <script>
        var app = angular.module("myModule", []);
        app.controller("myController", fun);
        function fun($scope, $http)
        {
            $scope.employees = [];
            $scope.f1 = function ()

```

```

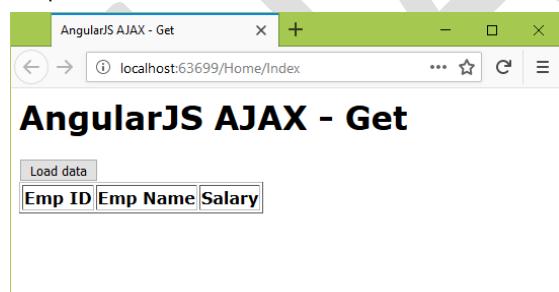
{
  $http.get("/home/getemployees").then(f2, f3);
}
function f2(response)
{
  $scope.employees = response.data;
}
function f3(err)
{
  alert(JSON.stringify(err));
}
}
</script>
</head>
<body>
<h1>AngularJS AJAX - Get</h1>
<div ng-app="myModule" ng-controller="myController">
  <input type="button" value="Load data" ng-click="f1()"/>
  <table border="1" id="table1">
    <tr>
      <th>Emp ID</th>
      <th>Emp Name</th>
      <th>Salary</th>
    </tr>
    <tr ng-repeat="emp in employees">
      <td>{{emp.EmpID}}</td>
      <td>{{emp.EmpName}}</td>
      <td>{{emp.Salary}}</td>
    </tr>
  </table>
</div>
</body>
</html>

```

#### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



Click on "Load data".

Emp ID	Emp Name	Salary
1	Scott	4000
2	Allen	2500
3	Jones	5200
4	James	4400
5	Smith	7600

## AngularJS AJAX - Search - Example

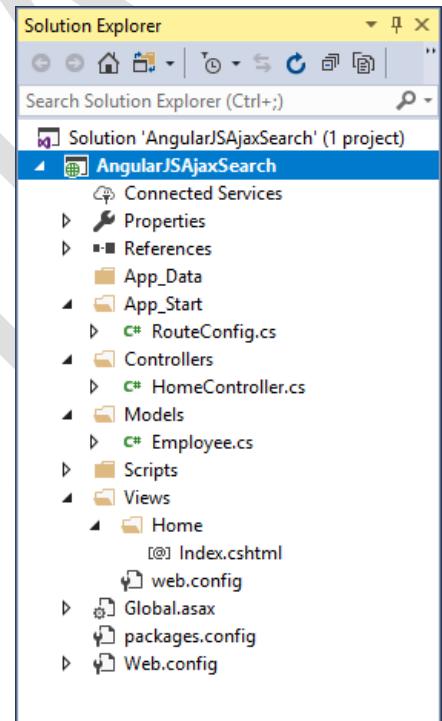
### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "AngularJSAjaxSearch". Type the location as "C:\Mvc". Type the solution name as "AngularJSAjaxSearch". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```
create database company
go
use company
go
create table Employees(
    EmpID int primary key,
    EmpName nvarchar(max),
    Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```



- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
install-package angularjs
```

### Creating Employee.cs

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

#### Code for "Employee.cs"

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;

namespace AngularJSAjaxSearch.Models
```

```
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }
        public DbSet<Employee> Employees { get; set; }
    }
}
```

**Creating HomeController.cs**

- Right click on “Controllers” folder and click on “Add” - “Controller”. Select “MVC 5 Controller - Empty”. Click on “Add”. Type the controller name as “HomeController”. Click on “Add”.

**Code for “HomeController.cs”**

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

namespace AngularJSAjaxSearch.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        [HttpGet]
        public ActionResult SearchEmployees(string searchstr)
        {
            CompanyDbContext db = new CompanyDbContext();
            List<Employee> emps = db.Employees.Where(temp => temp.EmpName.Contains(searchstr)).ToList();
            return Json(emps, JsonRequestBehavior.AllowGet);
        }
    }
}
```

**Creating Index.cshtml**

- Right click on “Views\Home” folder and click on “Add” - “View”. Type the view name as “Index”. Select the template “Empty (without model)”. Uncheck all the checkboxes. Click on “Add”.

**Code for “Views\Home\Index.cshtml”**

```
<html>
<head>
    <title>AngularJS AJAX - Search</title>
    <script src="~/Scripts/angular.js"></script>
    <script>
        var app = angular.module("myModule", []);
        app.controller("myController", fun1);
        function fun1($scope, $http)
        {
```

```

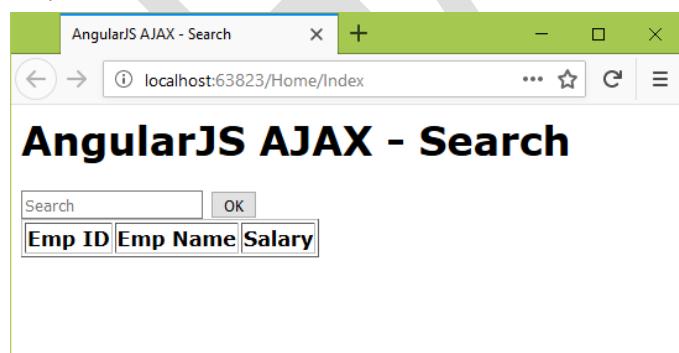
$scope.str="";
$scope.employees=[];
$scope.getdata = function () {
    $http.get("/home/searchemployees?searchstr=" + $scope.str).then(f1,f2);
};
function f1(response) {
    $scope.employees = response.data;
}
function f2(err) {
    alert(JSON.stringify(err));
}
}
</script>
</head>
<body>
<h1>AngularJS AJAX - Search</h1>
<div ng-app="mymodule" ng-controller="mycontroller">
<form>
<input type="text" placeholder="Search" ng-model="str">
<input type="submit" value="OK" ng-click="getdata()"><br>
<table id="table1" border="1">
<tr><th>Emp ID</th><th>Emp Name</th><th>Salary</th></tr>
<tr ng-repeat="emp in employees">
<td>{{emp.EmpID}}</td>
<td>{{emp.EmpName}}</td>
<td>{{emp.Salary}}</td>
</tr>
</table>
</form>
</div>
</body>
</html>

```

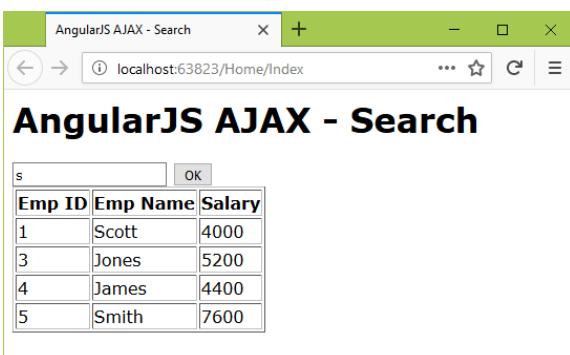
#### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



Type some text and click on OK.



## AngularJS AJAX - Post - Example

### Creating Project

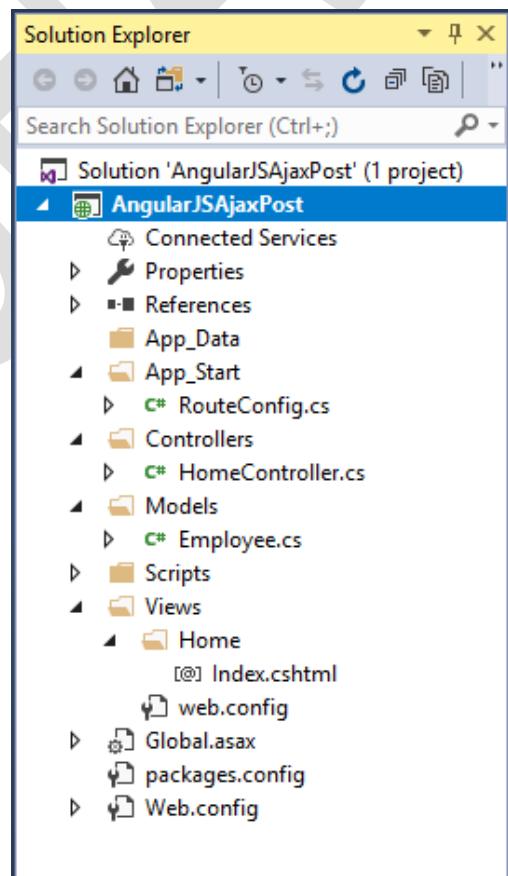
- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "AngularJSAjaxPost". Type the location as "C:\Mvc". Type the solution name as "AngularJSAjaxPost". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication".
- Click on "Connect".
- Ignore this step if "company" database already exists.
- Click on "New Query".
- Type the following code:

```
create database company
go
use company
go
create table Employees(
    EmpID int primary key,
    EmpName nvarchar(max),
    Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.



### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Type the following command in "Package Manager Console" and press Enter.
 

```
install-package EntityFramework
install-package angularjs
```

### Creating Employee.cs

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

**Code for "Employee.cs"**

```

using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Data.Entity;

namespace AngularJSAjaxPost.Models
{
    public class Employee
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }

        public DbSet<Employee> Employees { get; set; }
    }
}

```

**Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

**Code for "HomeController.cs"**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using AngularJSAjaxPost.Models;

namespace AngularJSAjaxPost.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        [HttpPost]
        public ActionResult InsertEmp(Employee emp)
        {
            CompanyDbContext db = new CompanyDbContext();
            db.Employees.Add(emp);
            db.SaveChanges();
            return Content("Successfully Inserted");
        }
    }
}

```

**Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Index.cshtml"**

```

<html>
  <head>
    <title>AngularJS AJAX - Post</title>
    <script src="~/Scripts/angular.js"></script>
    <script>
      var app = angular.module("mymodule", []);
      app.controller("mycontroller", fun1);
      function fun1($scope, $http)
      {
        $scope.emp = { EmpID: null, EmpName: null, Salary: null };
        $scope.message = "";
        $scope.savedata = function ()
        {
          $http.post("/home/InsertEmp", $scope.emp).then(f1, f2);
        };
        function f1(response)
        {
          $scope.message = response.data;
        }
        function f2(err)
        {
          alert(JSON.stringify(err));
        }
      }
    </script>
  </head>
  <body>
    <h1>AngularJS AJAX - Post</h1>
    <div ng-app="mymodule" ng-controller="mycontroller">
      <form>
        Emp ID: <input type="text" ng-model="emp.EmpID"><br>
        Emp Name: <input type="text" ng-model="emp.EmpName"><br>
        Salary: <input type="text" ng-model="emp.Salary"><br>
        <input type="submit" value="Insert" ng-click="savedata()"/>
        <br>{{message}}
      </form>
    </div>
  </body>
</html>

```

**Running the application**

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

**Output:**

Type some details and click on "Insert".

AngularJS AJAX - Post

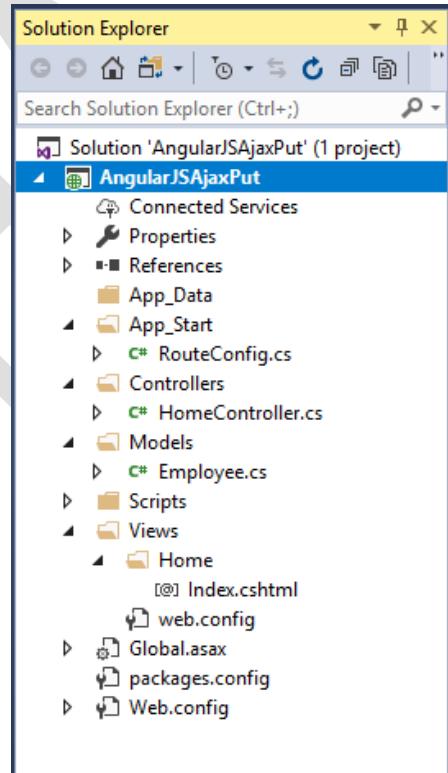
Emp ID: 40  
Emp Name: xyz  
Salary: 4570  
**Insert**

Successfully Inserted

## AngularJS AJAX - Put - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name as "AngularJSAjaxPut". Type the location as "C:\Mvc". Type the solution name as "AngularJSAjaxPut". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.



### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```
create database company
go
use company
go
create table Employees(
    EmpID int primary key,
    EmpName nvarchar(max),
    Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
install-package angularjs
```

### Creating Employee.cs

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

### Code for "Employee.cs"

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;

namespace AngularJSAjaxPut.Models
{
```

```

public class Employee
{
    [Key]
    public int EmpID { get; set; }
    public string EmpName { get; set; }
    public decimal Salary { get; set; }
}

public class CompanyDbContext : DbContext
{
    public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
    {
    }

    public DbSet<Employee> Employees { get; set; }
}

```

#### **Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### **Code for "HomeController.cs"**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Mvc;
using AngularJSAjaxPut.Models;

namespace AngularJSAjaxPut.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        [HttpPost]
        public ActionResult UpdateEmp(Employee emp)
        {
            CompanyDbContext db = new CompanyDbContext();
            Employee e = db.Employees.Where(temp => temp.EmpID == emp.EmpID).FirstOrDefault();
            e.EmpName = emp.EmpName;
            e.Salary = emp.Salary;
            db.SaveChanges();
            return Content("Successfully Updated");
        }
    }
}

```

#### **Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### **Code for "Views\Home\Index.cshtml"**

```

<html>
<head>
    <title>AngularJS AJAX - Put</title>
    <script src="~/Scripts/angular.js"></script>
    <script>
        var app = angular.module("myModule", []);
        app.controller("myController", fun);

```

```

function fun1($scope, $http)
{
    $scope.emp = { EmpID: null, EmpName: null, Salary: null };
    $scope.message = "";
    $scope.savedata = function ()
    {
        $http.put("/home/UpdateEmp", $scope.emp).then(f1, f2);
    };
    function f1(response)
    {
        $scope.message = response.data;
    }
    function f2(err)
    {
        alert(JSON.stringify(err));
    }
}
</script>
</head>
<body>
    <h1>AngularJS AJAX - Put</h1>
    <div ng-app="mymodule" ng-controller="mycontroller">
        <form>
            Existing Emp ID: <input type="text" ng-model="emp.EmpID"><br>
            Emp Name: <input type="text" ng-model="emp.EmpName"><br>
            Salary: <input type="text" ng-model="emp.Salary"><br>
            <input type="submit" value="Update" ng-click="savedata()">
            <br>{{message}}
        </form>
    </div>
</body>
</html>

```

#### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:

AngularJS AJAX - Put

Existing Emp ID:

Emp Name:

Salary:

Type some details and click on "Update".

AngularJS AJAX - Put

Existing Emp ID:

Emp Name:

Salary:

Successfully Updated

## AngularJS AJAX - Delete - Example

### Creating Project

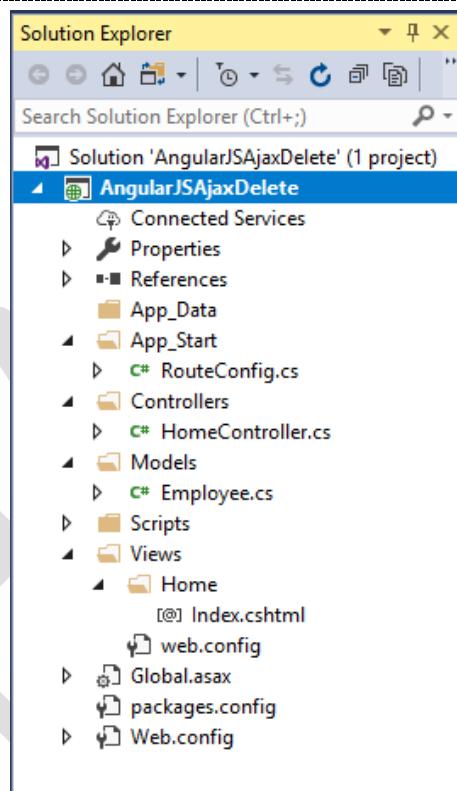
- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name as "AngularJSAjaxDelete". Type the location as "C:\Mvc". Type the solution name as "AngularJSAjaxDelete". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```
create database company
go
use company
go
create table Employees(
EmpID int primary key,
EmpName nvarchar(max),
Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.



### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
install-package angularjs
```

### Creating Employee.cs

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

#### Code for "Employee.cs"

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;

namespace AngularJSAjaxDelete.Models
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }
    }
}
```

```

    public DbSet<Employee> Employees { get; set; }
}
}

```

#### **Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### **Code for "HomeController.cs"**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Mvc;
using AngularJSAjaxDelete.Models;

namespace AngularJSAjaxDelete.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        [HttpDelete]
        public ActionResult DeleteEmp(int EmpIDToDelete)
        {
            CompanyDbContext db = new CompanyDbContext();
            Employee e = db.Employees.Where(temp => temp.EmpID == EmpIDToDelete).FirstOrDefault();
            db.Employees.Remove(e);
            db.SaveChanges();
            return Content("Successfully Deleted");
        }
    }
}

```

#### **Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### **Code for "Views\Home\Index.cshtml"**

```

<html>
<head>
    <title>AngularJS AJAX - Delete</title>
    <script src="~/Scripts/angular.js"></script>
    <script>
        var app = angular.module("mymodule", []);
        app.controller("mycontroller", fun1);
        function fun1($scope, $http)
        {
            $scope.EmpID = "";
            $scope.message = "";
            $scope.deletedata = function ()
            {
                $http.delete("/home/DeleteEmp?EmpIDToDelete=" + $scope.EmpID).then(f1, f2);
            };
            function f1(response)
            {
                $scope.message = response.data;
            }
            function f2(err)
            {

```

```

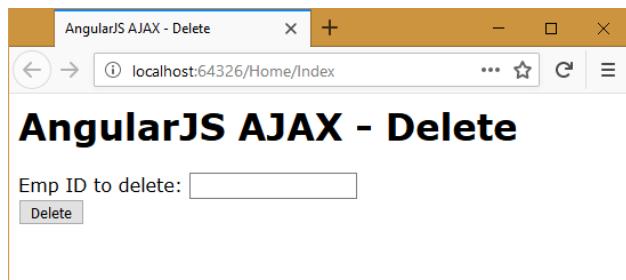
        alert(JSON.stringify(err));
    }
}
</script>
</head>
<body>
<h1>AngularJS AJAX - Delete</h1>
<div ng-app="mymodule" ng-controller="mycontroller">
<form>
    Emp ID to delete: <input type="text" ng-model="EmplID"><br>
    <input type="submit" value="Delete" ng-click="deletedata()"><br>
    {{message}}
</form>
</div>
</body>
</html>

```

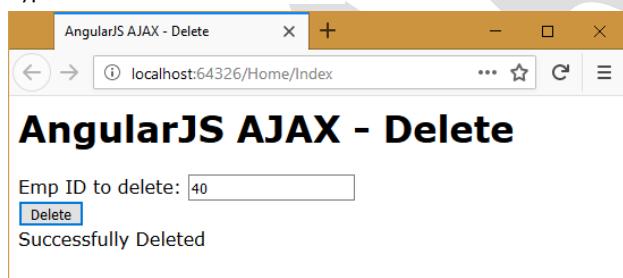
#### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

#### Output:



Type some id and click on "Delete".



## WEB API

### Introduction to Web API

- "ASP.NET Web API" is a framework, which is part of "ASP.NET" framework, which is used to create RESTful HTTP services, that can be called from any type of application. A "HTTP service" is a re-usable class, which provides methods for inserting, updating, deleting, retrieving and other operations on a database table. Web API services are called from any type of application like windows forms application, wpf application, windows phone application, iPhone application, Android application, asp.net web forms application, asp.net mvc application etc. Web API is alternative to "WCF REST".

#### Features / Advantages of ASP.NET Web API

- It supports REST (Representational State Transfer) standards. That means it supports four HTTP methods.
  - GET: Selecting / retrieving api/controllername
  - GET: Selecting / retrieving api/controllername/1
  - POST: Inserting api/controllername
  - PUT: Updating api/controllername

- **DELETE:** Deleting [api/controllername/id](#)

- We will separate AJAX related action methods from MVC controllers. The base class for all the web api controllers is "System.Web.Http.ApiController". Web API controllers work based on another routing file called "WebApiConfig.cs". It's configuration is easy. It supports "Automatic Model Binding". It supports "JSON serialization" by default. We have to use "AJAX" to call "Web API Controllers".

## HTTP Methods in Web API

Sl. No	HTTP Method	Description	Example URL	HTTP Request Body
1	GET	To get all records from the database table.	/api/controllername	[None]
2	GET	To get an existing record from the database table, based on id.	/api/controllername/id	[None]
3	GET	To get all records from the database table based on the given parameter.	/api/controllername?parameter=value	[None]
4	POST	To insert a new record into the database table.	/api/controllername	JSON
5	PUT	To update an existing record in the database table.	/api/controllername	JSON
6	DELETE	To delete an existing record in the database table.	/api/controllername/id	[None]

Web API - jQuery AJAX - Simple – Example

## **Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "jQueryAJAXWebApiSimpleExample". Type the location as "C:\Mvc". Type the solution name as "jQueryAJAXWebApiSimpleExample". Click on OK. Select "Empty". Check both checkboxes "MVC" and "Web API". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

## Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
  - "Package Manager Console" window will be opened.
  - Type the following command in "Package Manager Console" and press Enter.  
`install-package jQuery`

## Code for "WebApiConfig.cs"

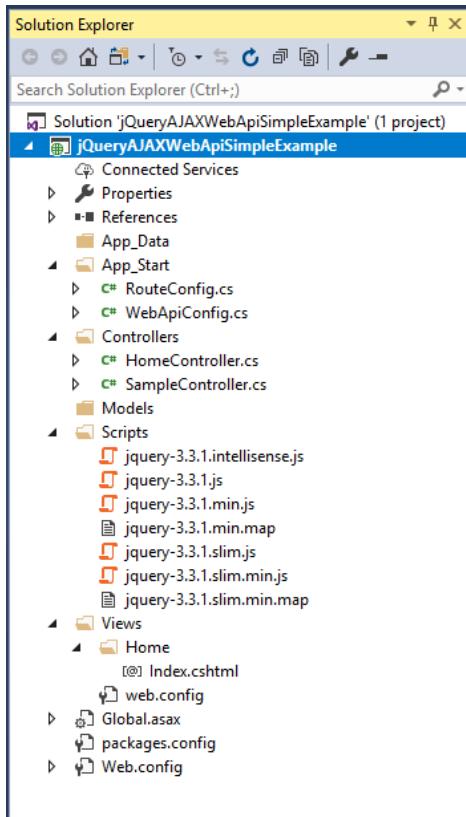
- Open "App\_Start"\WebApiConfig.cs";

## Code for “WebApiConfig.cs”

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

namespace jQueryAJAXWebApiSimpleExample
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional } );
```



```
    );
}
}
```

#### Creating SampleController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "Web API 2 Controller - Empty". Click on "Add". Type the controller name as "SampleController". Click on "Add".

#### Code for "SampleController.cs"

```
using System;
using System.Web.Http;

namespace jQueryAJAXWebApiSimpleExample.Controllers
{
    public class SampleController : ApiController
    {
        public string Get()
        {
            return "Hello";
        }
    }
}
```

#### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;

namespace jQueryAJAXWebApiSimpleExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

#### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### Code for "Views\Home\Index.cshtml"

```
<html>
    <head>
        <title>Web API - Simple</title>
    </head>
    <body>
        <h1>jQuery AJAX - Web API - Simple Example</h1>
        <input type="submit" id="btn1" value="Send AJAX Request" />
        <div id="div1"></div>
        <script src="~/Scripts/jquery-3.3.1.js"></script>
        <script>
            $("#btn1").click(fun1);
            function fun1(event)
            {
                event.preventDefault();
                $.ajax({ url: "/api/Sample", type: "GET", success: fun2, error: fun3 });
            }
        </script>
    </body>
</html>
```

```

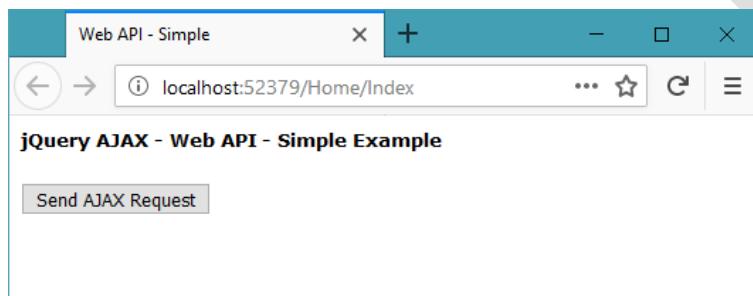
        }
        function funz(response)
        {
            $("#div1").append("<p>" + response + "</p>");
        }
        function fun3(xhr)
        {
            alert(xhr.responseText);
        }
    </script>
</body>
</html>

```

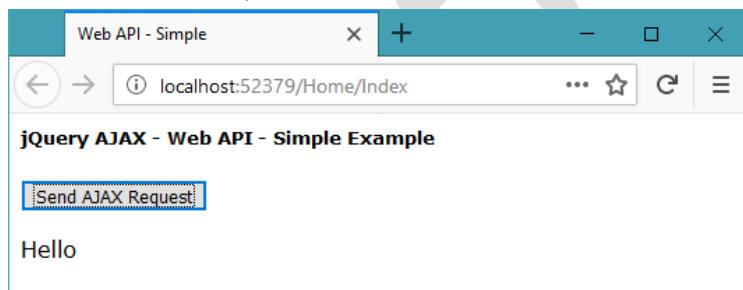
### **Running the application**

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



Click on "Send AJAX Request".



## Web API - jQuery AJAX - Get - Example

### **Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "jQueryAJAXWebAPIGetExample". Type the location as "C:\Mvc". Type the solution name as "jQueryAJAXWebAPIGetExample". Click on OK. Select "Empty". Check both checkboxes "MVC" and "Web API". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### **Creating Database**

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```

create database company
go
use company
go
create table Employees(
EmpID int primary key,
EmpName nvarchar(max),
Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)

```

```

insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go

```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### **Installing Packages**

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```

install-package EntityFramework
install-package jQuery

```

### **Creating Employee.cs**

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

### **Code for "Employee.cs"**

```

using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;

namespace jQueryAJAXWebAPIGetExample.Models
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }
        public DbSet<Employee> Employees { get; set; }
    }
}

```

### **Code for "WebApiConfig.cs"**

- Open "App\_Start\WebApiConfig.cs";

### **Code for "WebApiConfig.cs"**

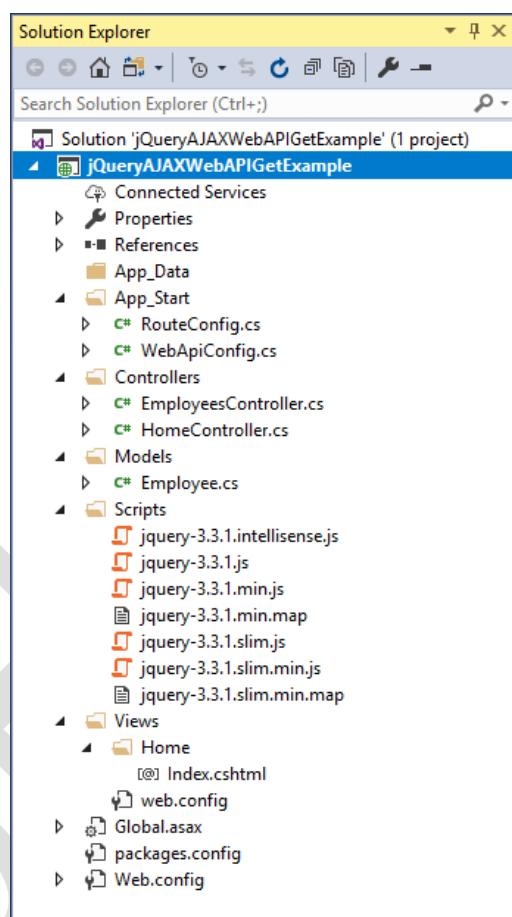
```

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

namespace jQueryAJAXWebAPIGetExample
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(

```



```

        name: "DefaultApi",
        routeTemplate: "api/{controller}/{id}",
        defaults: new { id = RouteParameter.Optional }
    );
}
}
}

```

#### **Creating EmployeesController.cs**

- Right click on “Controllers” folder and click on “Add” - “Controller”. Select “Web API 2 Controller - Empty”. Click on “Add”. Type the controller name as “EmployeesController”. Click on “Add”.

#### **Code for “EmployeesController.cs”**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Http;
using jQueryAJAXWebAPIGetExample.Models;

namespace jQueryAJAXWebAPIGetExample.Controllers
{
    public class EmployeesController : ApiController
    {
        public List<Employee> Get()
        {
            CompanyDbContext db = new CompanyDbContext();
            List<Employee> emps = db.Employees.ToList();
            return emps;
        }
    }
}

```

#### **Creating HomeController.cs**

- Right click on “Controllers” folder and click on “Add” - “Controller”. Select “MVC 5 Controller - Empty”. Click on “Add”. Type the controller name as “HomeController”. Click on “Add”.

#### **Code for “HomeController.cs”**

```

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

namespace jQueryAJAXWebAPIGetExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}

```

#### **Creating Index.cshtml**

- Right click on “Views\Home” folder and click on “Add” - “View”. Type the view name as “Index”. Select the template “Empty (without model)”. Uncheck all the checkboxes. Click on “Add”.

#### **Code for “Views\Home\Index.cshtml”**

```

<html>
<head>
    <title>Web API - Get</title>
</head>

```

```

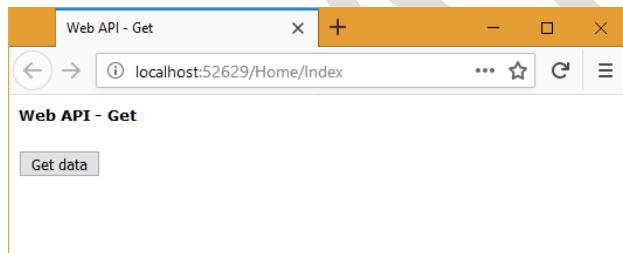
<body>
    <h1>Web API - Get</h1>
    <form>
        <input type="button" id="button1" value="Get data" />
        <table id="table1" border="1"></table>
    </form>
    <script src="~/Scripts/jquery-3.3.1.js"></script>
    <script>
        $("#button1").click(fun1);
        function fun1()
        {
            $.ajax({ url: "/api/Employees", type: "GET", success: fun2, error: fun3 });
        }
        function fun2(response)
        {
            $("#table1").html("<tr><th>Emp ID</th><th>Emp Name</th><th>Salary</th></tr>");
            for (var i = 0; i < response.length; i++)
            {
                $("#table1").append("<tr><td>" + response[i].EmpID + "</td><td>" + response[i].EmpName + "</td><td>" +
                    response[i].Salary + "</td></tr>");
            }
        }
        function fun3(xhr)
        {
            alert(xhr.responseText);
        }
    </script>
</body>
</html>

```

#### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

#### Output:



Click on "Get data".

Emp ID	Emp Name	Salary
1	Scott	4000
2	Allen	2500
3	Jones	5200
4	James	4400
5	Smith	7600

### Web API - jQuery AJAX - Search - Example

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "jQueryAJAXWebAPISearchExample". Type the location as "C:\Mvc". Type

the solution name as "jQueryAJAXWebAPISearchExample". Click on OK. Select "Empty". Check both checkboxes "MVC" and "Web API". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication".
- Click on "Connect".
- Ignore this step if "company" database already exists.
- Click on "New Query".
- Type the following code:

```
create database company
go
use company
go
create table Employees(
    EmpID int primary key,
    EmpName nvarchar(max),
    Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Type the following command in "Package Manager Console" and press Enter.  
`install-package EntityFramework  
install-package jQuery`

### Creating Employee.cs

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

#### Code for "Employee.cs"

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;

namespace jQueryAJAXWebAPISearchExample.Models
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }

        public DbSet<Employee> Employees { get; set; }
    }
}
```

```

}

}

Code for "WebApiConfig.cs"
• Open "App_Start"\WebApiConfig.cs";
Code for "WebApiConfig.cs"
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Http;

namespace jQueryAJAXWebAPISearchExample
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            config.MapHttpAttributeRoutes();

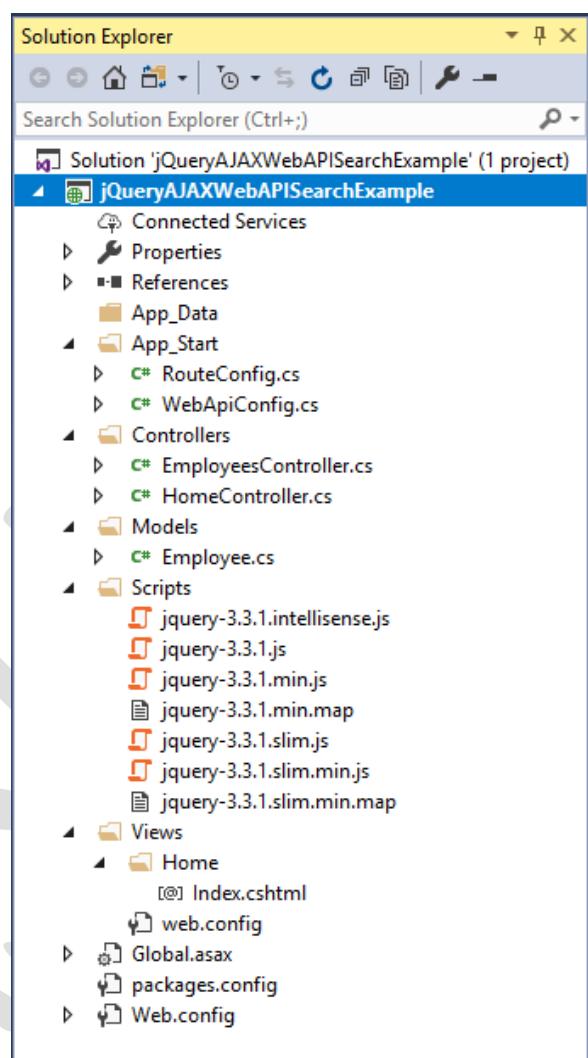
            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );
        }
    }
}

Creating EmployeesController.cs
• Right click on "Controllers" folder and click on "Add" - "Controller". Select "Web API 2 Controller - Empty". Click on "Add". Type the controller name as "EmployeesController". Click on "Add".
Code for "EmployeesController.cs"
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Http;
using jQueryAJAXWebAPISearchExample.Models;

namespace jQueryAJAXWebAPISearchExample.Controllers
{
    public class EmployeesController : ApiController
    {
        public List<Employee> Get(string str)
        {
            CompanyDbContext db = new CompanyDbContext();
            List<Employee> emps = db.Employees.Where(temp => temp.EmpName.Contains(str)).ToList();
            return emps;
        }
    }
}

Creating HomeController.cs
• Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".
Code for "HomeController.cs"
using System;
using System.Web.Mvc;

```



```
namespace jQueryAJAXWebAPISearchExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

#### **Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

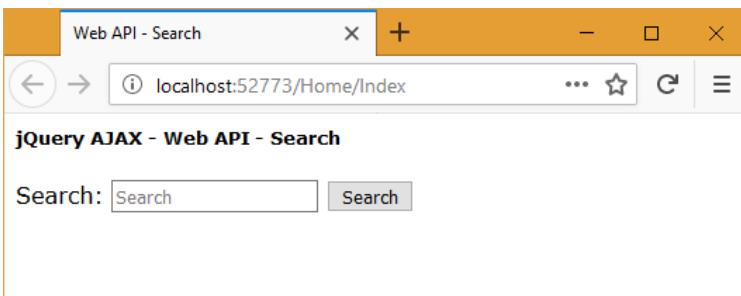
#### **Code for "Views\Home\Index.cshtml"**

```
<html>
<head>
    <title>Web API - Search</title>
</head>
<body>
    <h1>jQuery AJAX - Web API - Search</h1>
    <form>
        Search: <input type="text" id="txt1" placeholder="Search" />
        <input type="submit" id="button1" value="Search" />
        <table id="table1" border="1"></table>
    </form>
    <script src="~/Scripts/jquery-3.3.1.js"></script>
    <script>
        $("#button1").click(fun1);
        function fun1(event)
        {
            event.preventDefault();
            $.ajax({ url: "/api/Employees?str=" + $("#txt1").val(), type: "GET", success: fun2, error: fun3 });
        }
        function fun2(response)
        {
            $("#table1").html("<tr><th>Emp ID</th><th>Emp Name</th><th>Salary</th></tr>");
            for (var i = 0; i < response.length; i++)
            {
                $("#table1").append("<tr><td>" + response[i].EmpID + "</td><td>" + response[i].EmpName + "</td><td>" +
response[i].Salary + "</td></tr>");
            }
        }
        function fun3(xhr)
        {
            alert(xhr.responseText);
        }
    </script>
</body>
</html>
```

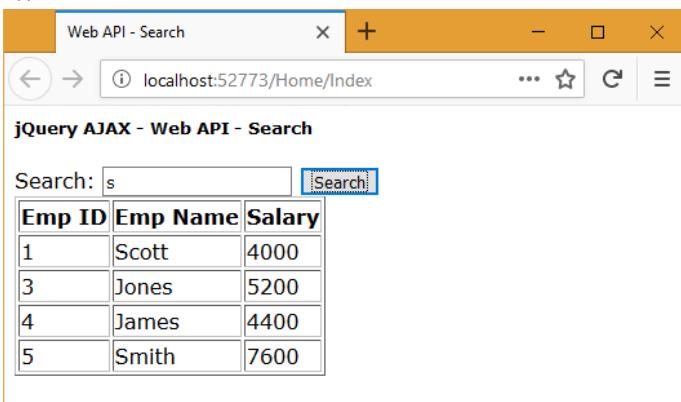
#### **Running the application**

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



Type some text and click on "Search".



### Web API - jQuery AJAX - Post - Example

#### **Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "jQueryAJAXWebAPIPostExample". Type the location as "C:\Mvc". Type the solution name as "jQueryAJAXWebAPIPostExample". Click on OK. Select "Empty". Check both checkboxes "MVC" and "Web API". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

#### **Creating Database**

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```
create database company
go
use company
go
create table Employees(EmpID int primary key, EmpName nvarchar(max), Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

#### **Installing Packages**

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
install-package jQuery
```

### **Creating Employee.cs**

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

### **Code for "Employee.cs"**

```

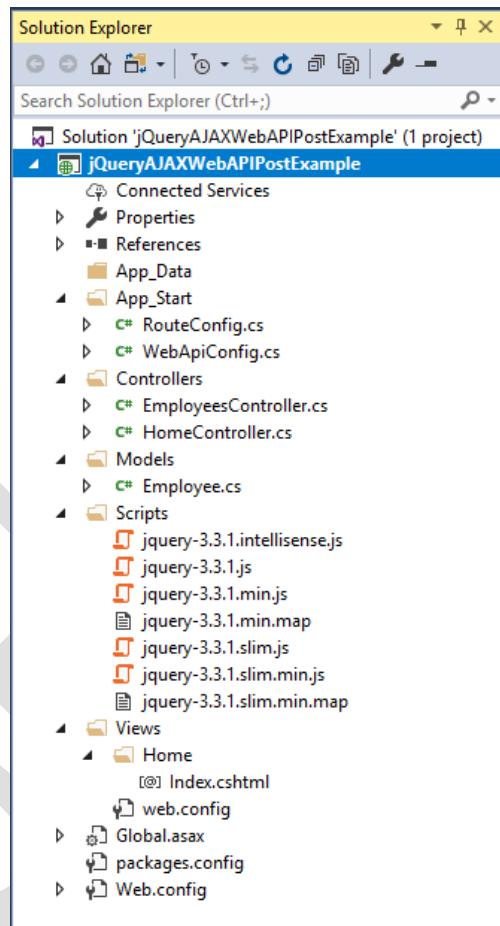
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Data.Entity;

namespace jQueryAJAXWebAPIPostExample.Models
{
    public class Employee
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data
source=localhost\sqlexpress; integrated security=yes; initial
catalog=company")
        {
        }

        public DbSet<Employee> Employees { get; set; }
    }
}

```



### **Code for "WebApiConfig.cs"**

- Open "App\_Start\WebApiConfig.cs";

### **Code for "WebApiConfig.cs"**

```

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

namespace jQueryAJAXWebAPIPostExample
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );
        }
    }
}

```

### Creating EmployeesController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "Web API 2 Controller - Empty". Click on "Add". Type the controller name as "EmployeesController". Click on "Add".

### Code for "EmployeesController.cs"

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Http;
using jQueryAJAXWebAPIPostExample.Models;

namespace jQueryAJAXWebAPIPostExample.Controllers
{
    public class EmployeesController : ApiController
    {
        public string Post(Employee emp)
        {
            CompanyDbContext db = new CompanyDbContext();
            db.Employees.Add(emp);
            db.SaveChanges();
            return "Successfully Inserted";
        }
    }
}
```

### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;

namespace jQueryAJAXWebAPIPostExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### Code for "Views\Home\Index.cshtml"

```
<html>
<head>
    <title>Web API - POST</title>
</head>
<body>
    <h1>Web API - POST</h1>
    <form>
        Emp ID: <input type="text" id="txt1" /><br />
        Emp Name: <input type="text" id="txt2" /><br />
        Salary: <input type="text" id="txt3" /><br />
        <input type="submit" id="button1" value="Insert" />
        <div id="div1"></div>
    </form>
    <script src="~/Scripts/jquery-3.3.1.js"></script>
```

```

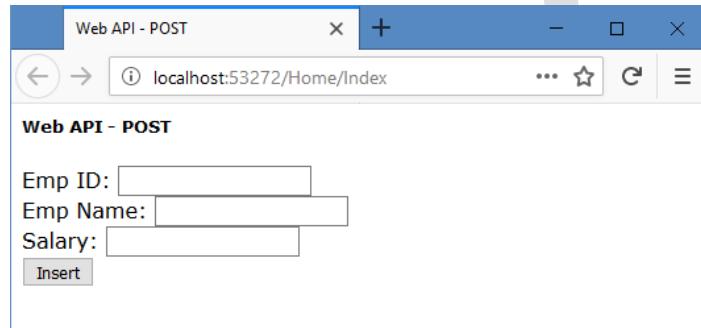
<script>
    $("#button1").click(fun1);
    function fun1(event)
    {
        event.preventDefault();
        var mydata = { "EmpID":$("#txt1").val(), "EmpName":$("#txt2").val(), "Salary":$("#txt3").val() };
        $.ajax({ url: "/api/Employees", type: "POST", data: mydata, success: fun2, error: fun3 });
    }
    function fun2(response)
    {
        $("#div1").html(response);
    }
    function fun3(xhr)
    {
        alert(xhr.responseText);
    }
</script>
</body>
</html>

```

### **Running the application**

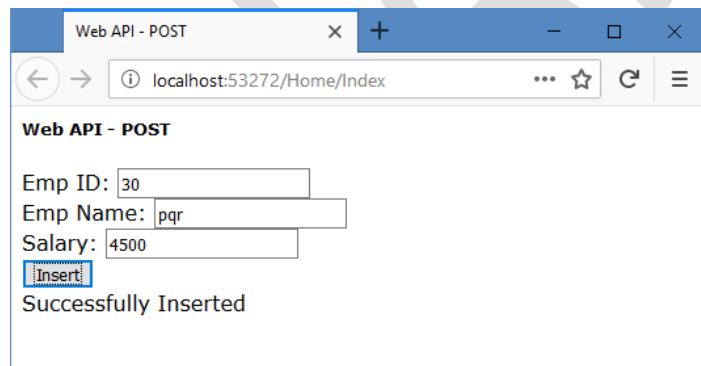
- Press “F5” to run the application.
- Type “<http://localhost:portnumber/Home/Index>”.

Output:



The screenshot shows a browser window with the title "Web API - POST". The address bar shows "localhost:53272/Home/Index". The main content area is titled "Web API - POST" and contains three input fields: "Emp ID:", "Emp Name:", and "Salary:". Below these fields is a blue "Insert" button.

Type some details and click on "Insert".



The screenshot shows the same browser window after entering data into the fields. The "Emp ID:" field has "30", the "Emp Name:" field has "pqr", and the "Salary:" field has "4500". The "Insert" button is now highlighted in blue. Below the form, the text "Successfully Inserted" is displayed in green.

## Web API - jQuery AJAX - Put - Example

### **Creating Project**

- Open Visual Studio 2017. Go to “File” – “New” – “Project”. Select “.NET Framework 4.7”. Select “Visual C#”. Select “ASP.NET Web Application (.NET Framework)”. Type the project name “jQueryAJAXWebAPIPutExample”. Type the location as “C:\Mvc”. Type the solution name as “jQueryAJAXWebAPIPutExample”. Click on OK. Select “Empty”. Check both checkboxes “MVC” and “Web API”. Uncheck the checkbox “Enable Docker support”. Uncheck the checkbox “Add unit tests”. Click on OK.

### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication".
- Click on "Connect".
- Ignore this step if "company" database already exists.
- Click on "New Query".
- Type the following code:

```
create database company
go
use company
go
create table Employees(
    EmpID int primary key,
    EmpName nvarchar(max),
    Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
install-package jQuery
```

### Creating Employee.cs

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

### Code for "Employee.cs"

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;

namespace jQueryAJAXWebAPIPutExample.Models
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }
        public DbSet<Employee> Employees { get; set; }
    }
}
```

### **Code for "WebApiConfig.cs"**

- Open "App\_Start"\WebApiConfig.cs";

### **Code for "WebApiConfig.cs"**

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

namespace jQueryAJAXWebAPIPutExample
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );
        }
    }
}
```

### **Creating EmployeesController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "Web API 2 Controller - Empty". Click on "Add". Type the controller name as "EmployeesController". Click on "Add".

### **Code for "EmployeesController.cs"**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Http;
using jQueryAJAXWebAPIPutExample.Models;
```

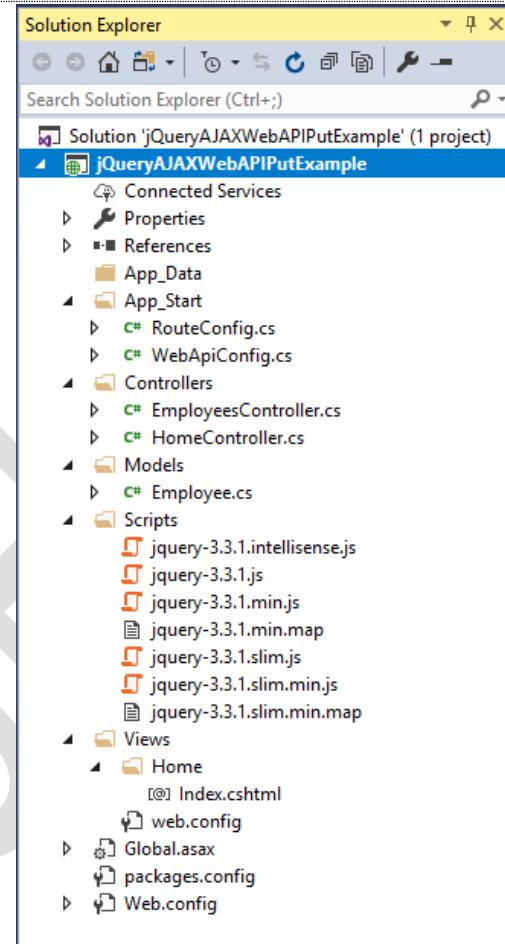
```
namespace jQueryAJAXWebAPIPutExample.Controllers
{
    public class EmployeesController : ApiController
    {
        public string Put(Employee emp)
        {
            CompanyDbContext db = new CompanyDbContext();
            Employee e = db.Employees.Where(temp => temp.EmpID == emp.EmpID).FirstOrDefault();
            e.EmpName = emp.EmpName;
            e.Salary = emp.Salary;
            db.SaveChanges();
            return "Successfully Updated";
        }
    }
}
```

### **Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### **Code for "HomeController.cs"**

```
using System;
using System.Web.Mvc;
```



```
namespace jQueryAJAXWebAPIPutExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

#### **Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

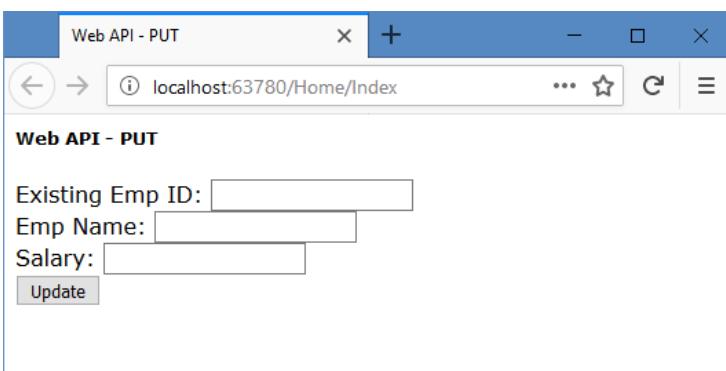
#### **Code for "Views\Home\Index.cshtml"**

```
<html>
    <head>
        <title>Web API - PUT</title>
    </head>
    <body>
        <h1>Web API - PUT</h1>
        <form>
            Existing Emp ID: <input type="text" id="txt1" /><br />
            Emp Name: <input type="text" id="txt2" /><br />
            Salary: <input type="text" id="txt3" /><br />
            <input type="submit" id="button1" value="Update" />
            <div id="div1"></div>
        </form>
        <script src="~/Scripts/jquery-3.3.1.js"></script>
        <script>
            $("#button1").click(fun1);
            function fun1(event)
            {
                event.preventDefault();
                var mydata = { "EmpID":$("#txt1").val(), "EmpName":$("#txt2").val(), "Salary":$("#txt3").val() };
                $.ajax({ url: "/api/Employees", type: "PUT", data: mydata, success: fun2, error: fun3 });
            }
            function fun2(response)
            {
                $("#div1").html(response);
            }
            function fun3(xhr)
            {
                alert(xhr.responseText);
            }
        </script>
    </body>
</html>
```

#### **Running the application**

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



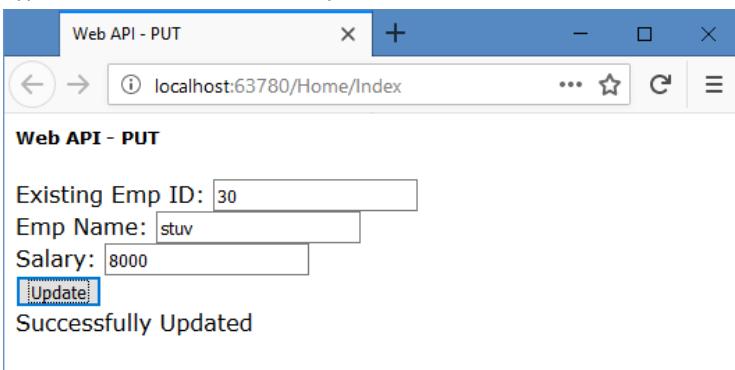
**Web API - PUT**

Existing Emp ID:

Emp Name:

Salary:

Type some details and click on "Update".



**Web API - PUT**

Existing Emp ID:

Emp Name:

Salary:

Successfully Updated



### Web API - jQuery AJAX - Delete - Example

#### **Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "jQueryAJAXWebAPIDeleteExample". Type the location as "C:\Mvc". Type the solution name as "jQueryAJAXWebAPIDeleteExample". Click on OK. Select "Empty". Check both checkboxes "MVC" and "Web API". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

#### **Creating Database**

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```
create database company
go
use company
go
create table Employees(EmpID int primary key,
EmpName nvarchar(max), Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

#### **Installing Packages**

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
install-package jQuery
```

### **Creating Employee.cs**

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

### **Code for "Employee.cs"**

```

using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;

namespace jQueryAJAXWebAPIDeleteExample.Models
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data
source=localhost\sqlexpress; integrated security=yes; initial
catalog=company")
        {

        }

        public DbSet<Employee> Employees { get; set; }
    }
}

```

### **Code for "WebApiConfig.cs"**

- Open "App\_Start\WebApiConfig.cs";

### **Code for "WebApiConfig.cs"**

```

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

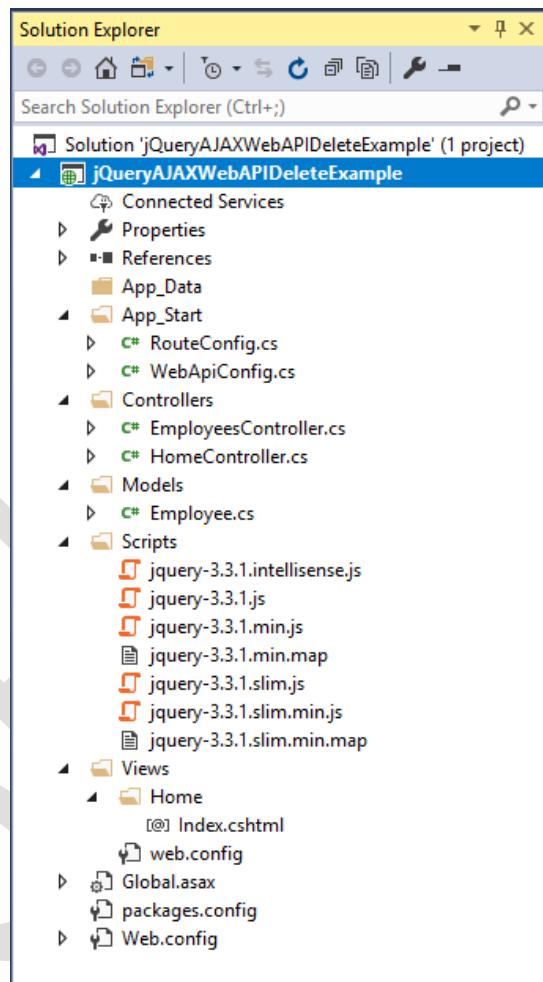
namespace jQueryAJAXWebAPIDeleteExample
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );
        }
    }
}

```

### **Creating EmployeesController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "Web API 2 Controller - Empty". Click on "Add". Type the controller name as "EmployeesController". Click on "Add".



**Code for "EmployeesController.cs"**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Http;
using jQueryAJAXWebAPIDeleteExample.Models;

namespace jQueryAJAXWebAPIDeleteExample.Controllers
{
    public class EmployeesController : ApiController
    {
        public string DeleteEmp(int EmpIDToDelete)
        {
            CompanyDbContext db = new CompanyDbContext();
            Employee e = db.Employees.Where(temp => temp.EmpID == EmpIDToDelete).FirstOrDefault();
            db.Employees.Remove(e);
            db.SaveChanges();
            return "Successfully Deleted";
        }
    }
}
```

**Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

**Code for "HomeController.cs"**

```
using System;
using System.Web.Mvc;

namespace jQueryAJAXWebAPIDeleteExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

**Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Index.cshtml"**

```
<html>
<head>
    <title>Web API - Delete</title>
</head>
<body>
    <h1>Web API - Delete</h1>
    <form>
        Existing Emp ID: <input type="text" id="txt1" /><br />
        <input type="submit" id="button1" value="Delete" />
        <div id="div1"></div>
    </form>
    <script src="~/Scripts/jquery-3.3.1.js"></script>
    <script>
        $("#button1").click(fun1);
        function fun1(event)
        {
```

```
event.preventDefault();
$.ajax({ url: "/api/Employees?EmpIDToDelete=" + $("#txt1").val(), type: "DELETE", success: fun2, error: fun3 });
}
function fun2(response)
{
    $("#div1").html(response);
}
function fun3(xhr)
{
    alert(xhr.responseText);
}
</script>
</body>
</html>
```

**Running the application**

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:

Web API - Delete

localhost:64327/Home/Index

Web API - Delete

Existing Emp ID:

Type some id and click on "Delete".

Web API - Delete

localhost:64327/Home/Index

Web API - Delete

Existing Emp ID:

Successfully Deleted

## Web API - AngularJS AJAX - Simple - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "AngularJSAXWebApiSimple". Type the location as "C:\Mvc". Type the solution name as "AngularJSAXWebApiSimple". Click on OK. Select "Empty". Check both checkboxes "MVC" and "Web API". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package angularjs
```

### Code for "WebApiConfig.cs"

- Open "App\_Start"\WebApiConfig.cs";

### Code for "WebApiConfig.cs"

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

namespace AngularJSAXWebApiSimple
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );
        }
    }
}
```

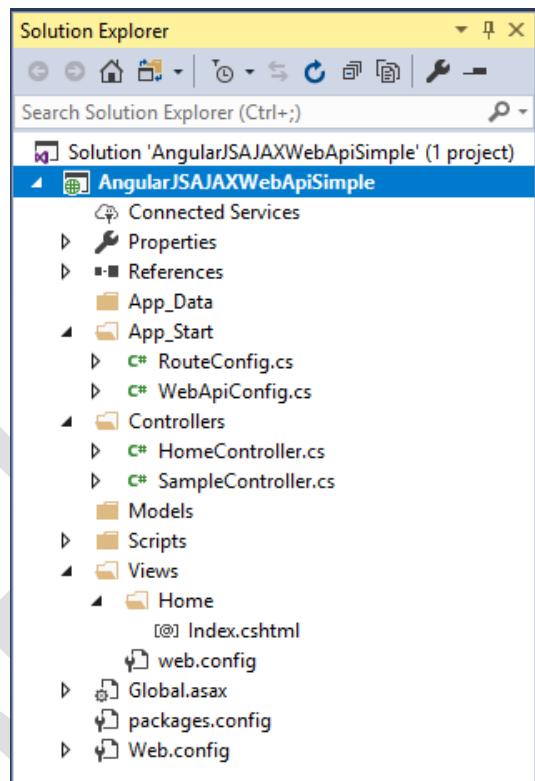
### Creating SampleController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "Web API 2 Controller - Empty". Click on "Add". Type the controller name as "SampleController". Click on "Add".

### Code for "SampleController.cs"

```
using System;
using System.Web.Http;

namespace AngularJSAXWebApiSimple.Controllers
{
    public class SampleController : ApiController
    {
        public string Get()
        {
            return "Hello";
        }
    }
}
```



### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```
using System;
using System.Web.Mvc;

namespace AngularJSAJAXWebApiSimple.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

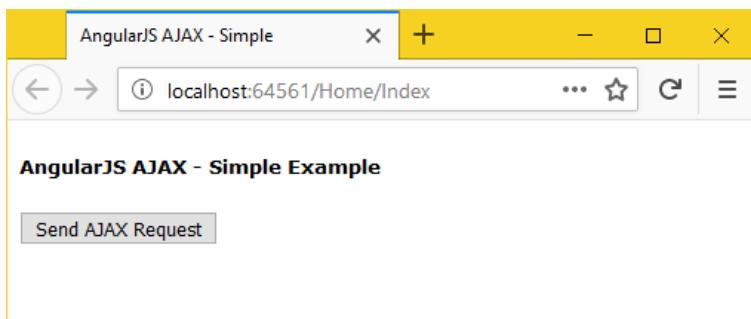
### Code for "Views\Home\Index.cshtml"

```
<html>
<head>
    <title>AngularJS AJAX - Simple</title>
    <script src="~/Scripts/angular.js"></script>
    <script>
        var app = angular.module("mymodule", []);
        app.controller("mycontroller", ctrlfun);
        function ctrlfun($scope, $http)
        {
            $scope.message = "";
            $scope.getdata = function ()
            {
                $http.get("/api/Sample").then(fun1, fun2);
            };
            function fun1(response)
            {
                $scope.message += response.data;
            }
            function fun2(err)
            {
                console.log(err);
            }
        }
    </script>
</head>
<body>
    <div ng-app="mymodule" ng-controller="mycontroller">
        <h1>AngularJS AJAX - Simple Example</h1>
        <input type="button" ng-click="getdata()" value="Send AJAX Request" />
        <div id="div1">{{message}}</div>
    </div>
</body>
</html>
```

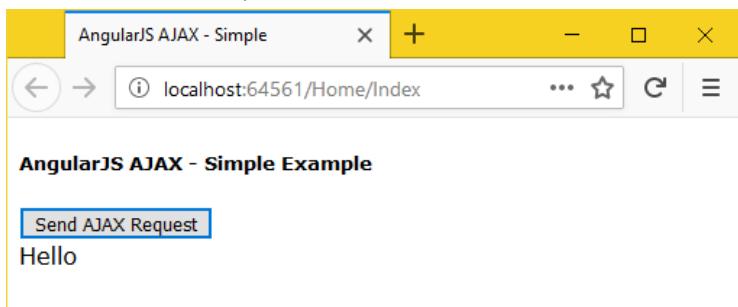
### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



Click on "Send AJAX Request".



### Web API - AngularJS AJAX - Get - Example

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "AngularJSAXWebAPIGet". Type the location as "C:\Mvc". Type the solution name as "AngularJSAXWebAPIGet". Click on OK. Select "Empty". Check both checkboxes "MVC" and "Web API". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

#### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query".
- Type the following code:

```
create database company
go
use company
go
create table Employees(
    EmpID int primary key,
    EmpName nvarchar(max),
    Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

#### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
install-package angularjs
```

### **Creating Employee.cs**

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

### **Code for "Employee.cs"**

```

using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;

namespace AngularJSAJAXWebAPIGet.Models
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data
source=localhost\sqlexpress; integrated security=yes; initial
catalog=company")
        {

        }

        public DbSet<Employee> Employees { get; set; }
    }
}

```

### **Code for "WebApiConfig.cs"**

- Open "App\_Start\WebApiConfig.cs";

### **Code for "WebApiConfig.cs"**

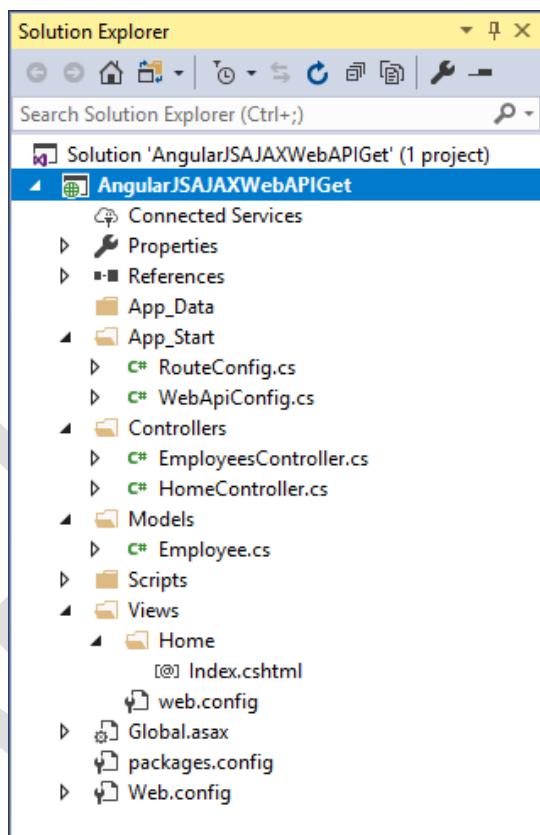
```

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

namespace AngularJSAJAXWebAPIGet
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );
        }
    }
}

```



### **Creating EmployeesController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "Web API 2 Controller - Empty". Click on "Add". Type the controller name as "EmployeesController". Click on "Add".

**Code for "EmployeesController.cs"**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Http;
using AngularJSAJAXWebAPIGet.Models;

namespace AngularJSAJAXWebAPIGet.Controllers
{
    public class EmployeesController : ApiController
    {
        public List<Employee> Get()
        {
            CompanyDbContext db = new CompanyDbContext();
            List<Employee> emps = db.Employees.ToList();
            return emps;
        }
    }
}
```

**Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

**Code for "HomeController.cs"**

```
using System;
using System.Web.Mvc;

namespace AngularJSAJAXWebAPIGet.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

**Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Index.cshtml"**

```
<html>
<head>
<title>Web API - Get</title>
<script src="~/Scripts/angular.js"></script>
<script>
var app = angular.module("myModule", []);
app.controller("myController", ctrlfun);
function ctrlfun($scope, $http)
{
    $scope.employees = [];
    $scope.getData = function ()
    {
        $http.get("/api/employees").then(fun1, fun2);
    };
    function fun1(response)
    {
        $scope.employees = response.data;
    }
}
```

```

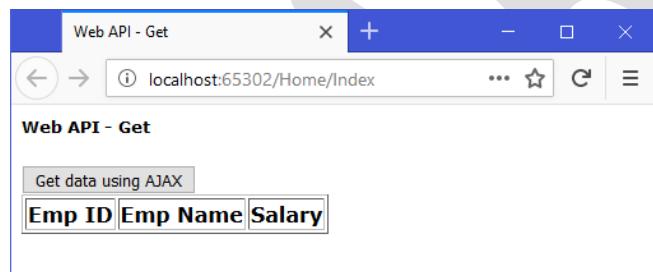
function funz(err)
{
  console.log(err);
}
</script>
</head>
<body>
<h1>Web API - Get</h1>
<div ng-app="mymodule" ng-controller="mycontroller">
<input type="button" value="Get data using AJAX" ng-click="getdata()" />
<table border="1" id="table1">
<tr>
<th>Emp ID</th>
<th>Emp Name</th>
<th>Salary</th>
</tr>
<tr ng-repeat="emp in employees">
<td>{{emp.EmpID}}</td>
<td>{{emp.EmpName}}</td>
<td>{{emp.Salary}}</td>
</tr>
</table>
</div>
</body>
</html>

```

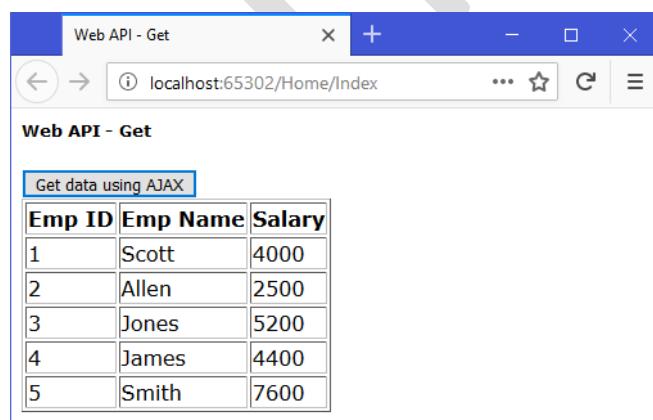
#### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

#### Output:



Click on "Get data using AJAX".



## Web API - AngularJS AJAX - Search - Example

### **Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "AngularJSAJAXWebAPISearch". Type the location as "C:\Mvc". Type the solution name as "AngularJSAJAXWebAPISearch". Click on OK. Select "Empty". Check both checkboxes "MVC" and "Web API". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### **Creating Database**

- Open SQL Server Management Studio. Select "Windows Authentication".
- Click on "Connect".
- Ignore this step if "company" database already exists.
- Click on "New Query".
- Type the following code:

```
create database company
go
use company
go
create table Employees(
EmpID int primary key,
EmpName nvarchar(max),
Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### **Installing Packages**

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Type the following command in "Package Manager Console" and press Enter.
   
install-package EntityFramework
   
install-package angularjs

### **Creating Employee.cs**

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

#### **Code for "Employee.cs"**

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;

namespace AngularJSAJAXWebAPISearch.Models
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
```

```
public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
{
}

public DbSet<Employee> Employees { get; set; }
}
```

### Code for "WebApiConfig.cs"

- Open "App\_Start\WebApiConfig.cs";

### Code for "WebApiConfig.cs"

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

namespace AngularJSAJAXWebAPISearch
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );
        }
    }
}
```

### Creating EmployeesController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "Web API 2 Controller - Empty". Click on "Add". Type the controller name as "EmployeesController". Click on "Add".

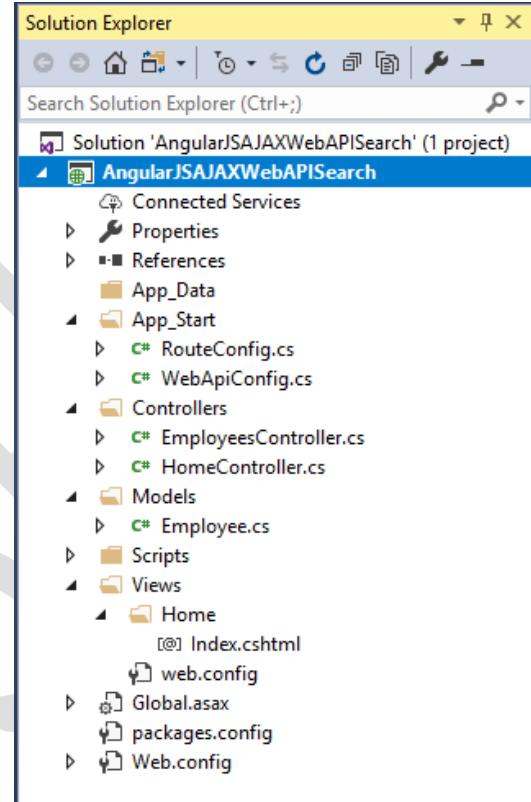
### Code for "EmployeesController.cs"

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Http;
using AngularJSAJAXWebAPISearch.Models;

namespace AngularJSAJAXWebAPISearch.Controllers
{
    public class EmployeesController : ApiController
    {
        public List<Employee> Get(string str)
        {
            CompanyDbContext db = new CompanyDbContext();
            List<Employee> emps = db.Employees.Where(temp => temp.EmpName.Contains(str)).ToList();
            return emps;
        }
    }
}
```

### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".



**Code for "HomeController.cs"**

```
using System;
using System.Web.Mvc;

namespace AngularJSAJAXWebAPISearch.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

**[Creating Index.cshtml]**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Index.cshtml"**

```
<html>
<head>
<title>Web API - Search</title>
<script src="~/Scripts/angular.js"></script>
<script>
var app = angular.module("mymodule", []);
app.controller("mycontroller", ctrlfun);
function ctrlfun($scope, $http)
{
    $scope.str = "";
    $scope.employees = [];
    $scope.getdata = function ()
    {
        $http.get("/api/employees?str=" + $scope.str).then(fun1, fun2);
    };
    function fun1(response)
    {
        $scope.employees = response.data;
    }
    function fun2(err)
    {
        console.log(err);
    }
}
</script>
</head>
<body>
<h1>Web API - Search</h1>
<div ng-app="mymodule" ng-controller="mycontroller">
<form>
Search: <input type="text" placeholder="Search" ng-model="str" />
<input type="submit" value="Search" ng-click="getdata()" />
<table border="1" id="table1">
<tr>
<th>Emp ID</th>
<th>Emp Name</th>
<th>Salary</th>
</tr>
<tr ng-repeat="emp in employees">
<td>{{emp.EmpID}}</td>
<td>{{emp.EmpName}}</td>
<td>{{emp.Salary}}</td>

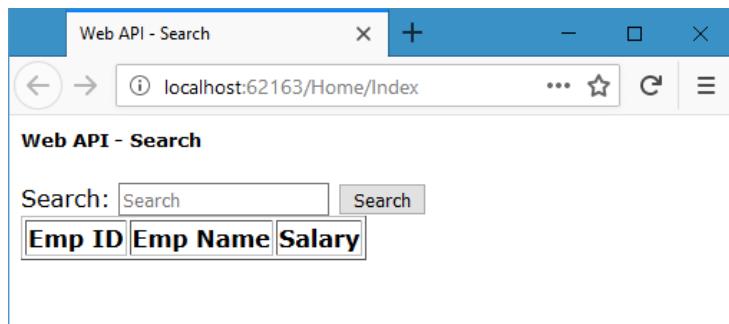
```

```
</tr>
</table>
</form>
</div>
</body>
</html>
```

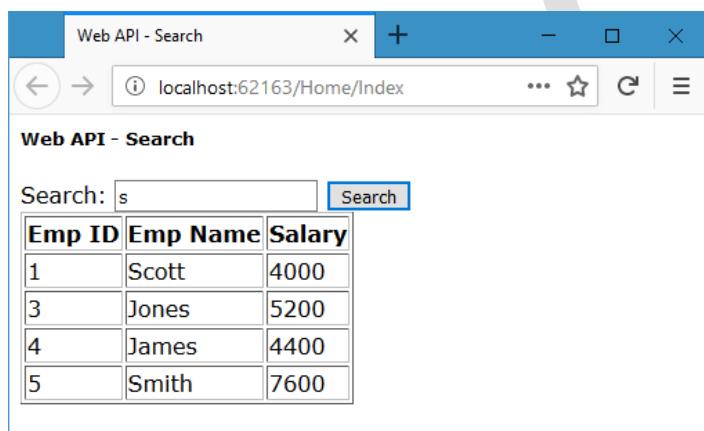
### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



Type some text and click on Search.



## Web API - AngularJS AJAX - Post - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "AngularJSAXWebAPIPost". Type the location as "C:\Mvc". Type the solution name as "AngularJSAXWebAPIPost". Click on OK. Select "Empty". Check both checkboxes "MVC" and "Web API". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication".
- Click on "Connect".
- Ignore this step if "company" database already exists.
- Click on "New Query".
- Type the following code:

```
create database company
go
use company
go
create table Employees(
```

```

EmpID int primary key,
EmpName nvarchar(max),
Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go

```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### **Installing Packages**

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Type the following command in "Package Manager Console" and press Enter.

```

install-package EntityFramework
install-package angularjs

```

### **Creating Employee.cs**

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

### **Code for "Employee.cs"**

```

using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Data.Entity;

namespace AngularJSAJAXWebAPIPost.Models
{
    public class Employee
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {

        }

        public DbSet<Employee> Employees { get; set; }
    }
}

```

### **Code for "WebApiConfig.cs"**

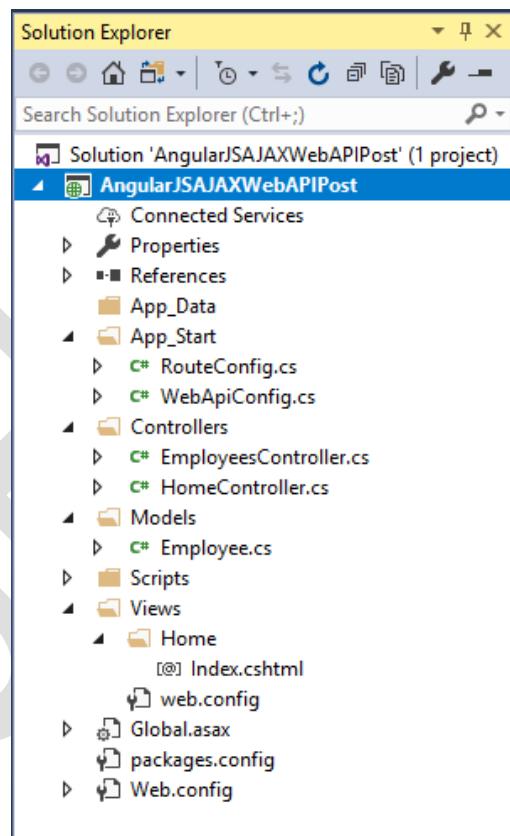
- Open "App\_Start"\WebApiConfig.cs";

### **Code for "WebApiConfig.cs"**

```

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

```



```

namespace AngularJSAJAXWebAPIPost
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );
        }
    }
}

```

#### **Creating EmployeesController.cs**

- Right click on “Controllers” folder and click on “Add” - “Controller”. Select “Web API 2 Controller - Empty”. Click on “Add”. Type the controller name as “EmployeesController”. Click on “Add”.

#### **Code for “EmployeesController.cs”**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Http;
using AngularJSAJAXWebAPIPost.Models;

namespace AngularJSAJAXWebAPIPost.Controllers
{
    public class EmployeesController : ApiController
    {
        CompanyDbContext db = new CompanyDbContext();

        public string Post(Employee emp)
        {
            db.Employees.Add(emp);
            db.SaveChanges();
            return "Successfully Inserted";
        }
    }
}

```

#### **Creating HomeController.cs**

- Right click on “Controllers” folder and click on “Add” - “Controller”. Select “MVC 5 Controller - Empty”. Click on “Add”. Type the controller name as “HomeController”. Click on “Add”.

#### **Code for “HomeController.cs”**

```

using System;
using System.Web.Mvc;

namespace AngularJSAJAXWebAPIPost.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}

```

**Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Index.cshtml"**

```

<html>
  <head>
    <title>AngularJS AJAX - Post</title>
    <script src="~/Scripts/angular.js"></script>
    <script>
      var app = angular.module("mymodule", []);
      app.controller("mycontroller", function ($scope, $http)
      {
        $scope.emp = { EmpID: "", EmpName: "", Salary: "" };
        $scope.message = "";
        $scope.savedata = function()
        {
          $http.post("/api/employees", $scope.emp).then(fun1, fun2);
        };
        function fun1(response)
        {
          $scope.message = response.data;
        }
        function fun2(err)
        {
          console.log(err);
        }
      });
    </script>
  </head>
  <body>
    <h1>AngularJS AJAX - POST</h1>
    <div ng-app="mymodule" ng-controller="mycontroller">
      <form>
        Emp ID: <input type="text" ng-model="emp.EmpID" /><br />
        Emp Name: <input type="text" ng-model="emp.EmpName" /><br />
        Salary: <input type="text" ng-model="emp.Salary" /><br />
        <input type="submit" value="Insert" ng-click="savedata()" /><br />
        {{message}}
      </form>
    </div>
  </body>
</html>

```

**Running the application**

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:

AngularJS AJAX - Post

localhost:53006/Home/Index

**AngularJS AJAX - POST**

Emp ID:

Emp Name:

Salary:

Type some details and click on "Insert".

AngularJS AJAX - Post

localhost:53006/Home/Index

**AngularJS AJAX - POST**

Emp ID: 40

Emp Name: xyz

Salary: 4570

Successfully Inserted

### Web API - AngularJS AJAX - Put - Example

#### **[Creating Project]**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "AngularJSAJAXWebAPIPut". Type the location as "C:\Mvc". Type the solution name as "AngularJSAJAXWebAPIPut". Click on OK. Select "Empty". Check both checkboxes "MVC" and "Web API". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

#### **[Creating Database]**

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```
create database company
go
use company
go
create table Employees(EmpID int primary key, EmpName nvarchar(max), Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

#### **[Installing Packages]**

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
install-package angularjs
```

### **Creating Employee.cs**

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

### **Code for "Employee.cs"**

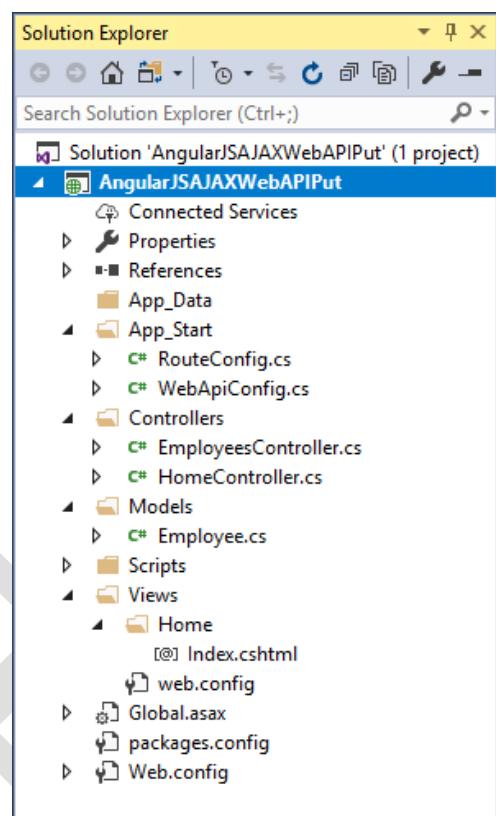
```
using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;

namespace AngularJSAJAXWebAPIPut.Models
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress;
integrated security=yes; initial catalog=company")
        {

        }

        public DbSet<Employee> Employees { get; set; }
    }
}
```



### **Code for "WebApiConfig.cs"**

- Open "App\_Start\WebApiConfig.cs";

### **Code for "WebApiConfig.cs"**

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

namespace AngularJSAJAXWebAPIPut
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );
        }
    }
}
```

### **Creating EmployeesController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "Web API 2 Controller - Empty". Click on "Add". Type the controller name as "EmployeesController". Click on "Add".

**Code for "EmployeesController.cs"**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Http;
using AngularJSAJAXWebAPIPut.Models;

namespace AngularJSAJAXWebAPIPut.Controllers
{
    public class EmployeesController : ApiController
    {
        public string Put(Employee emp)
        {
            CompanyDbContext db = new CompanyDbContext();
            Employee e = db.Employees.Where(temp => temp.EmpID == emp.EmpID).FirstOrDefault();
            e.EmpName = emp.EmpName;
            e.Salary = emp.Salary;
            db.SaveChanges();
            return "Successfully Updated";
        }
    }
}
```

**Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

**Code for "HomeController.cs"**

```
using System;
using System.Web.Mvc;

namespace AngularJSAJAXWebAPIPut.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

**Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Index.cshtml"**

```
<html>
<head>
    <title>Web API - PUT</title>
    <script src="~/Scripts/angular.js"></script>
    <script>
        var app = angular.module("myModule", []);
        app.controller("myController", function ($scope, $http)
        {
            $scope.emp = { EmpID: "", EmpName: "", Salary: "" };
            $scope.message = "";
            $scope.savedata = function ()
            {
                $http.put("/api/employees", $scope.emp).then(fun1, fun2);
            };
            function fun1(response)
```

```

{
  $scope.message = response.data;
}
function fun2(err)
{
  console.log(err);
}
});
</script>
</head>
<body>
<h1>Web API - PUT</h1>
<div ng-app="myModule" ng-controller="myController">
<form>
  Existing Emp ID: <input type="text" ng-model="emp.EmpID" /><br />
  Emp Name: <input type="text" ng-model="emp.EmpName" /><br />
  Salary: <input type="text" ng-model="emp.Salary" /><br />
  <input type="submit" value="Update" ng-click="savedata0()" />
  <br />{{message}}
</form>
</div>
</body>
</html>

```

**Running the application**

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:

Web API - PUT

Existing Emp ID:

Emp Name:

Salary:

Type some details and click on "Update".

Web API - PUT

Existing Emp ID:

Emp Name:

Salary:

Successfully Updated

**Web API - AngularJS AJAX - Delete - Example****Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "AngularJSAXWebAPIDelete". Type the location as "C:\Mvc". Type the

solution name as "AngularJSAJAXWebAPIDelete". Click on OK. Select "Empty". Check both checkboxes "MVC" and "Web API". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### **Creating Database**

- Open SQL Server Management Studio. Select "Windows Authentication".
- Click on "Connect".
- Ignore this step if "company" database already exists.
- Click on "New Query".
- Type the following code:

```
create database company
go
use company
go
create table Employees(
    EmpID int primary key,
    EmpName nvarchar(max),
    Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### **Installing Packages**

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Type the following command in "Package Manager Console" and press Enter.
   
install-package EntityFramework
   
install-package angularjs

### **Creating Employee.cs**

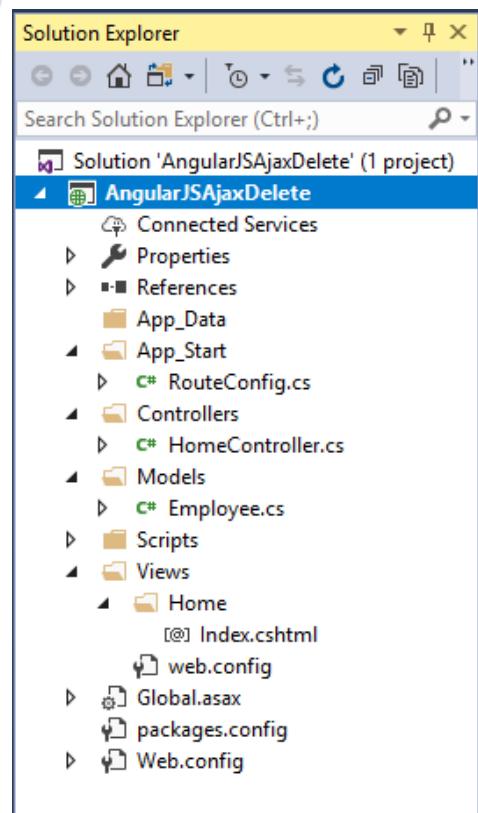
- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

#### **Code for "Employee.cs"**

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;

namespace AngularJSAJAXWebAPIDelete.Models
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress;
integrated security=yes; initial catalog=company")
    }
}
```



```
public DbSet<Employee> Employees { get; set; }
}
```

#### Code for "WebApiConfig.cs"

- Open "App\_Start"\WebApiConfig.cs";

#### Code for "WebApiConfig.cs"

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

namespace AngularJSAJAXWebAPIDelete
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );
        }
    }
}
```

#### Creating EmployeesController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "Web API 2 Controller - Empty". Click on "Add". Type the controller name as "EmployeesController". Click on "Add".

#### Code for "EmployeesController.cs"

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Http;
using AngularJSAJAXWebAPIDelete.Models;

namespace AngularJSAJAXWebAPIDelete.Controllers
{
    public class EmployeesController : ApiController
    {
        CompanyDbContext db = new CompanyDbContext();

        public string Delete(int id)
        {
            Employee emp = db.Employees.Where(temp => temp.EmpID == id).FirstOrDefault();
            db.Employees.Remove(emp);
            db.SaveChanges();
            return "Successfully Deleted";
        }
    }
}
```

#### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

**Code for "HomeController.cs"**

```
using System;
using System.Web.Mvc;

namespace AngularJSAJAXWebAPIDelete.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

**[Creating Index.cshtml]**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Index.cshtml"**

```
<html>
<head>
<title>Web API - Delete</title>
<script src="~/Scripts/angular.js"></script>
<script>
var app = angular.module("mymodule", []);
app.controller("mycontroller", function ($scope, $http)
{
    $scope.empid = "";
    $scope.message = "";
    $scope.deletedata = function ()
    {
        $http.delete("/api/employees/" + $scope.empid).then(fun1, fun2);
    };
    function fun1(response)
    {
        $scope.message = response.data;
    }
    function fun2(err)
    {
        console.log(err);
    }
});
</script>
</head>
<body>
<h1>Web API - Delete</h1>
<div ng-app="mymodule" ng-controller="mycontroller">
<form>
    Emp ID to delete: <input type="text" ng-model="empid" /><br />
    <input type="submit" value="Delete" ng-click="deletedata()" /><br />
    {{message}}
</form>
</div>
</body>
</html>
```

**[Running the application]**

- Press "F5" to run the application.
  - Type "http://localhost:portnumber/Home/Index".
- Output:

Web API - Delete

Emp ID to delete:

**Delete**

Type some id and click on "Delete".

Web API - Delete

Emp ID to delete:

**Delete**

Successfully Deleted

## HTML HELPERS AND VALIDATIONS

### Introduction to HTML Helpers

- HTML helpers are the predefined methods in ASP.NET MVC, that can be used in the view, to create a specific html tag. HTML helper methods execute at server and generate (render) a specific html tag at run time.
- Advantages:**
  - HTML helper methods supports data bindings. The "property name" in model class and "field name" in the view will be maintained as same, consistently. HTML helper methods are required because, html doesn't support "data bindings" directly. For example, "<input type='text'" tag can't be binded to a model property directly. For every input type, mvc provides a html helper method.

#### List of HTML Helper Methods

Sl. No	HTML Helper	HTML Tag
1	@Html.TextBoxFor()	<input type="text">
2	@Html.PasswordFor()	<input type="password">
3	@Html.TextAreaFor()	<textarea></textarea>
4	@Html.RadioButtonFor()	<input type="radio">
5	@Html.CheckBoxFor()	<input type="checkbox">
6	@Html.DropDownListFor()	<select></select>
7	@Html.ListBoxFor()	<select multiple="multiple"></select>
8	@Html.LabelFor()	<label></label>
9	@Html.BeginForm()	<form></form>
10	@Html.ValidationMessageFor()	<span>error message</span>
11	@Html.ActionLink()	<a></a>
12	@Html.DisplayFor()	[model value]
13	@Html.DisplayNameFor()	[Property name]
14	@Html.EditorFor()	<input type="text">
15	@Html.HiddenFor()	<input type="hidden">
16	@Html.AntiForgeryToken()	<input type="hidden">
17	@Html.ValidationMessageFor()	<ul>errors list</ul>

## HTML Helpers - Example

### **Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "HtmlHelpersExample". Type the location as "C:\Mvc". Type the solution name as "HtmlHelpersExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### **Creating Database**

- Open SQL Server Management Studio. Select "Windows Authentication".
- Click on "Connect".
- Ignore this step if "htmlhelpersdatabase" database already exists.
- Click on "New Query".
- Type the following code:

```
create database htmlhelpersdatabase
go
use htmlhelpersdatabase
go
create table Employees(
    EmpID int primary key,
    EmpName nvarchar(max),
    Username nvarchar(max),
    Password nvarchar(max),
    Mobile nvarchar(max),
    Email nvarchar(max),
    Address nvarchar(max),
    Gender nvarchar(max),
    Salary decimal,
    Country nvarchar(max),
    AcceptLicenseAgreement bit not null)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

### **Installing Packages**

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.  
`install-package EntityFramework`

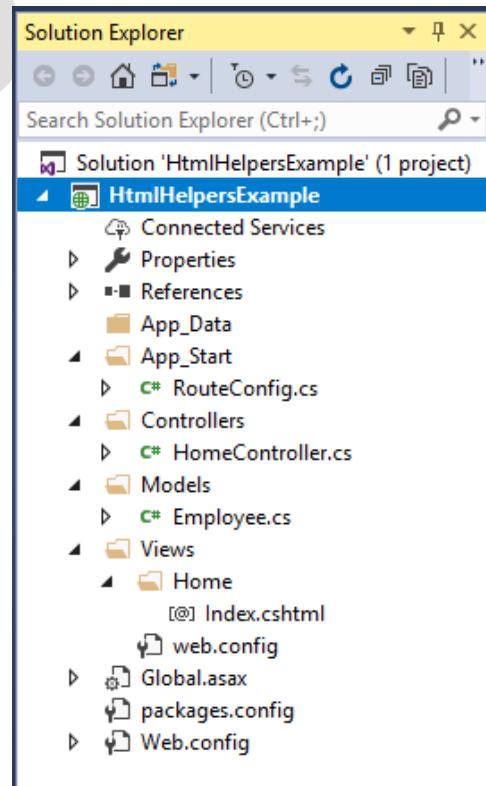
### **Creating Employee.cs**

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

#### **Code for "Employee.cs"**

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Data.Entity;

namespace HtmlHelpersExample.Models
{
    public class Employee
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
    }
}
```



```

public string Username { get; set; }
public string Password { get; set; }
[NotMapped]
public string ConfirmPassword { get; set; }
public string Mobile { get; set; }
public string Email { get; set; }
public string Address { get; set; }
public string Gender { get; set; }
public decimal Salary { get; set; }
public string Country { get; set; }
public bool AcceptLicenseAgreement { get; set; }
}

public class HtmlHelpersDatabaseDbContext : DbContext
{
    public HtmlHelpersDatabaseDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=htmlhelpersdatabase")
    {
    }

    public DbSet<Employee> Employees { get; set; }
}

```

#### **Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### **Code for "HomeController.cs"**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Mvc;
using HtmlHelpersExample.Models;

namespace HtmlHelpersExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        [HttpPost]
        public ActionResult InsertEmp(Employee emp)
        {
            HtmlHelpersDatabaseDbContext db = new HtmlHelpersDatabaseDbContext();
            db.Employees.Add(emp);
            db.SaveChanges();
            return Content("Successfully Inserted");
        }
    }
}

```

#### **Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### **Code for "Views\Home\Index.cshtml"**

```

@model HtmlHelpersExample.Models.Employee
<html>
    <head>
        <title>HTML Helpers</title>

```

```

</head>
<body>
<h1>HTML Helpers</h1>
@using (Html.BeginForm("InsertEmp", "Home", FormMethod.Post))
{
    @Html.LabelFor(temp => temp.EmpID)
    @Html.TextBoxFor(temp => temp.EmpID)<br />

    @Html.LabelFor(temp => temp.EmpName)
    @Html.TextBoxFor(temp => temp.EmpName)<br />

    @Html.LabelFor(temp => temp.Username)
    @Html.TextBoxFor(temp => temp.Username)<br />

    @Html.LabelFor(temp => temp.Password)
    @Html.PasswordFor(temp => temp.Password)<br />

    @Html.LabelFor(temp => temp.ConfirmPassword)
    @Html.PasswordFor(temp => temp.ConfirmPassword)<br />

    @Html.LabelFor(temp => temp.Mobile)
    @Html.TextBoxFor(temp => temp.Mobile)<br />

    @Html.LabelFor(temp => temp.Email)
    @Html.TextBoxFor(temp => temp.Email)<br />

    @Html.LabelFor(temp => temp.Address)
    @Html.TextAreaFor(temp => temp.Address)<br />

    @Html.LabelFor(temp => temp.Gender)
    @Html.RadioButtonFor(temp => temp.Gender, "Male") @:Male
    @Html.RadioButtonFor(temp => temp.Gender, "Female") @:Female
    <br />

    @Html.LabelFor(temp => temp.Salary)
    @Html.TextBoxFor(temp => temp.Salary)<br />

    @Html.LabelFor(temp => temp.Country)
    @Html.DropDownListFor(temp => temp.Country, new List<SelectListItem>{
        new SelectListItem() { Text = "India", Value = "IND" },
        new SelectListItem() { Text = "China", Value = "CHN" },
        new SelectListItem() { Text = "Japan", Value = "JPN" },
        }, "Please Select")
    <br />

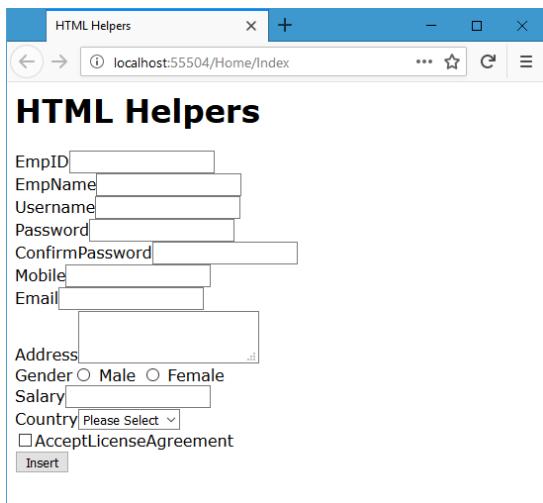
    @Html.CheckBoxFor(temp => temp.AcceptLicenseAgreement)
    @Html.LabelFor(temp => temp.AcceptLicenseAgreement)<br />
    <input type="submit" value="Insert"/>
}
</body>
</html>

```

**Running the application**

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



## Custom HTML Helpers - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "CustomHtmlHelpersExample". Type the location as "C:\Mvc". Type the solution name as "CustomHtmlHelpersExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating Employee.cs

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

#### Code for "Employee.cs"

```
using System;

namespace CustomHtmlHelpersExample.Models
{
    public class Employee
    {
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public double Salary { get; set; }
    }
}
```

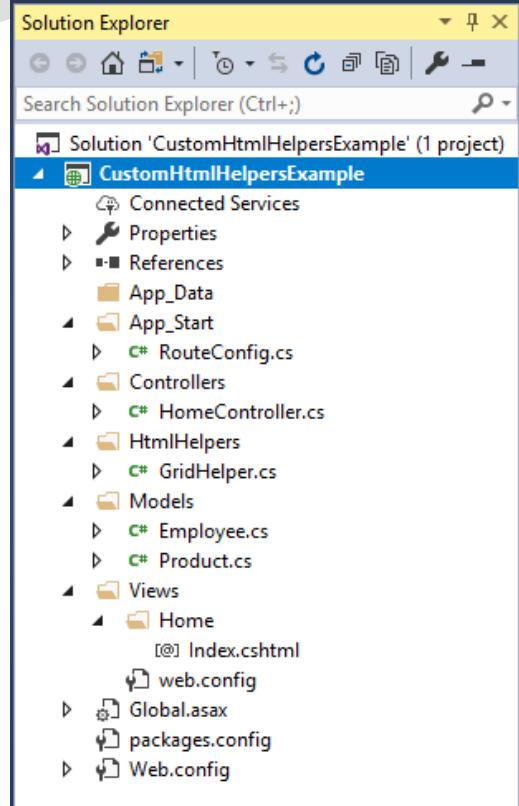
### Creating Product.cs

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Product.cs". Click on "Add".

#### Code for "Product.cs"

```
using System;

namespace CustomHtmlHelpersExample.Models
{
    public class Product
    {
        public int ProductID { get; set; }
        public string ProductName { get; set; }
        public double Cost { get; set; }
    }
}
```



**Creating GridHelper.cs**

- Right click on the project (CustomHtmlHelpersExample), and click on "Add" - "New Folder". Type the folder name as "HtmlHelpers" and press Enter. Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "GridHelper.cs". Click on "Add".

**Code for "GridHelper.cs"**

```

using System;
using System.Collections.Generic;
using System.Web.Mvc;
using CustomHtmlHelpersExample.Models;
using System.Reflection;
using System.Collections;

namespace CustomHtmlHelpersExample.HtmlHelpers
{
    public static class GridHelper
    {
        public static MvcHtmlString Grid(this HtmlHelper helper, IList mycollection)
        {
            //var mycollection = (IList)mycollectionparam;
            PropertyInfo[] properties = mycollection[0].GetType().GetProperties();

            string str = "<table border='1'>";
            str += "<tr>";
            for (int i = 0; i < properties.Length; i++)
            {
                str += "<th>" + properties[i].Name + "</th>";
            }
            str += "</tr>";
            for (int i = 0; i < mycollection.Count; i++)
            {
                str += "<tr>";
                for (int j = 0; j < properties.Length; j++)
                {
                    str += "<td>" + properties[j].GetValue(mycollection[i]) + "</td>";
                }
                str += "</tr>";
            }
            str += "</table>";

            return new MvcHtmlString(str);
        }
    }
}

```

**Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

**Code for "HomeController.cs"**

```

using System;
using System.Collections.Generic;
using System.Web.Mvc;
using CustomHtmlHelpersExample.Models;

namespace CustomHtmlHelpersExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            List<Employee> emps = new List<Employee>()

```

```

    {
        new Employee() { EmpID = 1, EmpName = "Scott", Salary = 6677 },
        new Employee() { EmpID = 2, EmpName = "Allen", Salary = 2938 },
        new Employee() { EmpID = 3, EmpName = "John", Salary = 8832 }
    };
    List<Product> prods = new List<Product>()
    {
        new Product() { ProductID = 1, ProductName = "Mobile", Cost = 20000 },
        new Product() { ProductID = 2, ProductName = "Laptop", Cost = 40000 },
        new Product() { ProductID = 3, ProductName = "TV", Cost = 60000 },
        new Product() { ProductID = 4, ProductName = "Tablet", Cost = 25000 }
    };
    ViewBag.myemployees = emps;
    ViewBag.myproducts = prods;
    return View();
}
}
}

```

**Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Index.cshtml"**

```

@using CustomHtmlHelpersExample.HtmlHelpers
@using CustomHtmlHelpersExample.Models
<html>
<head>
<title>Custom HTML Helpers</title>
</head>
<body>
<h1>Custom HTML Helpers</h1>
@{
    List<Employee> emps = (List<Employee>)ViewBag.myemployees;
    List<Product> prods = (List<Product>)ViewBag.myproducts;
}
@Html.Grid(emps)
@Html.Grid(prods)
</body>
</html>

```

**Running the application**

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:

EmpID	EmpName	Salary
1	Scott	6677
2	Allen	2938
3	John	8832

ProductID	ProductName	Cost
1	Mobile	20000
2	Laptop	40000
3	TV	60000
4	Tablet	25000

### Custom HTML Helpers

- "Custom HTML Helpers" are re-usable UI blocks, which can be invoked in the view. We can pass dynamic data to the html helper.
- HTML helper method should be an extension method, which will be added to "System.Web.Mvc.HtmlHelper" class and which returns "System.Web.Mvc.MvcHtmlString" class.

#### Syntax to Create Custom HTML Helpers

```
public static class classname
{
    public static MvcHtmlString Methodname(this HtmlHelper helper, additionalarguments)
    {
        Code here
    }
}
```

## VALIDATIONS

### Introduction to Validations

- "Validation" is a process of checking the form input values whether those are correct or not. "ASP.NET MVC" recommends to use "model level validations".
- Advantage: We can perform both server side validation & client side validation also.

#### Data Annotations

- Data annotations are pre-defined attributes (classes) that are applicable to the model properties, to set validation rule.
- All the data annotation classes are available in "System.ComponentModel.DataAnnotations" namespace.

1. [Required]
2. [Range]
3. [Compare]
4. [StringLength]
5. [RegularExpression]
6. [Key]
7. [DatabaseGenerated]
8. [ForeignKey]
9. [NotMapped]

#### 1. [Required]

- The "Required" attribute is used to make a field mandatory (cannot be blank).
- **Syntax:** [Required(ErrorMessage = "any message here")]

#### 2. [Range]

- The "Range" attribute is used to specify the minimum value and maximum value for a number.
- **Syntax:** [Range(minimum, maximum, ErrorMessage = "any message here")]

#### 3. [Compare]

- The "Compare" attribute is used to check whether two property values are same or not.
- **Syntax:** [Compare("other property", ErrorMessage = "any message here")]

#### 4. [StringLength]

- The "StringLength" attribute is used to specify the maximum no. of characters that can be accepted in the property.
- **Syntax:** [StringLength(n, ErrorMessage = "any message here")]

#### 5. [RegularExpression]

- The "RegularExpression" attribute is used to specify the pattern of the value that you want to accept.
- **Syntax:** [RegularExpression("reg exp here", ErrorMessage = "any message here")]

### 6. [Key]

- The "Key" attribute is used to define primary key.
- **Syntax:** [Key]

### 7. [DatabaseGenerated]

- The "DatabaseGenerated" attribute is used to specify that the field is not "identity" column.
- **Syntax:** [DatabaseGenerated(DatabaseGeneratedOption.None)]

### 8. [ForeignKey]

- The "ForeignKey" attribute is used to specify foreign key (reference key).
- **Syntax:** [ForeignKey("property name")]

### 9. [NotMapped]

- The "NotMapped" attribute is used to specify that the model property doesn't mapped with any database column.
- **Syntax:** [NotMapped]

#### ModelState.IsValid

- This property returns true / false, which represents whether the model object is valid or not.

## Custom Validations

#### Custom Validations

- To create a custom validation attribute, you have to create a class & inherit from a pre-defined class called "System.ComponentModel.DataAnnotations.ValidationAttribute".
- Then override the virtual method called "IsValid" that has the following signature:

```
protected virtual ValidationResult IsValid(object value,  
ValidationContext validationContext)
```

#### Syntax of Custom Validation

```
using System;  
using System.ComponentModel.DataAnnotations;  
namespace Projectname.CustomValidations  
{  
    public class ClassNameAttribute : ValidationAttribute  
    {  
        protected override ValidationResult IsValid(object value, ValidationContext validationContext)  
        {  
            Code here  
            if(condition)  
            {  
                return ValidationResult.Success;  
            }  
            else  
            {  
                return new ValidationResult(this.ErrorMessage);  
            }  
        }  
    }  
}
```

}

### Validations - Example

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "ValidationsExample". Type the location as "C:\Mvc". Type the solution name as "ValidationsExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

#### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "htmlhelpersdatabase" database already exists. Click on "New Query". Type the following code:

```
create database htmlhelpersdatabase
go
use htmlhelpersdatabase
go
create table Employees(
    EmpID int primary key,
    EmpName nvarchar(max),
    Username nvarchar(max),
    Password nvarchar(max),
    Mobile nvarchar(max),
    Email nvarchar(max),
    Address nvarchar(max),
    Gender nvarchar(max),
    Salary decimal,
    Country nvarchar(max),
    AcceptLicenseAgreement bit not null)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

#### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
```

#### Creating Unique.cs

- Right click on the project (ValidationsExample), and click on "Add" - "New Folder". Type the folder name as "CustomValidations" and press Enter. Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Unique.cs". Click on "Add".

#### Code for "CustomValidations\Unique.cs"

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using ValidationsExample.Models;

namespace ValidationsExample.CustomValidations
{
    public class UniqueAttribute : ValidationAttribute
    {
        protected override ValidationResult IsValid(object value, ValidationContext validationContext)
        {
            HtmlHelpersDatabaseDbContext db = new HtmlHelpersDatabaseDbContext();
            string currentusername = Convert.ToString(value);
            if(db.Employees.Where(temp => temp.Username == currentusername).Count() == 0)
            {

```

```

        return ValidationResult.Success;
    }
    else
    {
        return new ValidationResult(this.ErrorMessage);
    }
}
}
}

```

#### **Creating Employee.cs**

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

#### **Code for "Employee.cs"**

```

using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Data.Entity;
using ValidationsExample.CustomValidations;

namespace ValidationsExample.Models
{
    public class Employee
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        [Required(ErrorMessage = "Emp ID can't be blank.")]
        public int EmpID { get; set; }

        [Required(ErrorMessage = "Emp Name can't be blank.")]
        [RegularExpression(@"^([a-zA-Z]+)$", ErrorMessage = "Emp Name should contain alphabets only.")]
        public string EmpName { get; set; }

        [Required(ErrorMessage = "Username can't be blank.")]
        [Unique(ErrorMessage = "Duplicate username")]
        public string Username { get; set; }

        [Required(ErrorMessage = "Password can't be blank.")]
        public string Password { get; set; }

        [Required(ErrorMessage = "Confirm Password can't be blank.")]
        [Compare("Password", ErrorMessage = "Password and confirm password do not match.")]
        [NotMapped]
        public string ConfirmPassword { get; set; }

        [Required(ErrorMessage = "Mobile can't be blank.")]
        public string Mobile { get; set; }

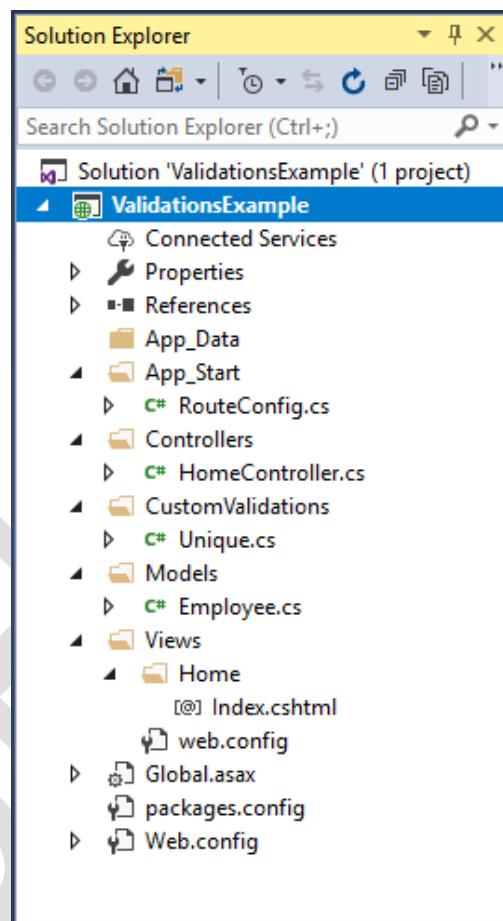
        [Required(ErrorMessage = "Email can't be blank.")]
        [RegularExpression(@"^(w+@[a-zA-Z_]+?\.[a-zA-Z]{2,6})$", ErrorMessage = "Invalid E-Mail.")]
        public string Email { get; set; }

        public string Address { get; set; }

        [Required(ErrorMessage = "Gender can't be blank.")]
        public string Gender { get; set; }

        [Range(1000, 50000, ErrorMessage = "Salary should be in between 1000 and 50000.")]
        [Required(ErrorMessage = "Salary can't be blank")]
        public decimal Salary { get; set; }
    }
}

```



```
[Required(ErrorMessage = "Country can't be blank.")]
public string Country { get; set; }

public bool AcceptLicenseAgreement { get; set; }
}

public class HtmlHelpersDatabaseDbContext : DbContext
{
    public HtmlHelpersDatabaseDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=htmlhelpersdatabase")
    {
    }

    public DbSet<Employee> Employees { get; set; }
}
}
```

**Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

**Code for "HomeController.cs"**

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

namespace ValidationsExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        HtmlHelpersDatabaseDbContext db = new HtmlHelpersDatabaseDbContext();

        [HttpPost]
        public ActionResult InsertEmp(Employee emp)
        {
            if (ModelState.IsValid == true)
            {
                db.Employees.Add(emp);
                db.SaveChanges();
                return Content("Successfully Inserted");
            }
            else
            {
                return View("Index");
            }
        }
    }
}
```

**Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Index.cshtml"**

```
@model ValidationsExample.Models.Employee
```

```

<html>
  <head>
    <title>Validations</title>
  </head>
  <body>
    <h1>Validations</h1>
    @using (Html.BeginForm("InsertEmp", "Home", FormMethod.Post))
    {
      @Html.LabelFor(temp => temp.EmpID)
      @Html.TextBoxFor(temp => temp.EmpID)
      @Html.ValidationMessageFor(temp => temp.EmpID)
      <br />

      @Html.LabelFor(temp => temp.EmpName)
      @Html.TextBoxFor(temp => temp.EmpName)
      @Html.ValidationMessageFor(temp => temp.EmpName)
      <br />

      @Html.LabelFor(temp => temp.Username)
      @Html.TextBoxFor(temp => temp.Username)
      @Html.ValidationMessageFor(temp => temp.Username)
      <br />

      @Html.LabelFor(temp => temp.Password)
      @Html.PasswordFor(temp => temp.Password)
      @Html.ValidationMessageFor(temp => temp.Password)
      <br />

      @Html.LabelFor(temp => temp.ConfirmPassword)
      @Html.PasswordFor(temp => temp.ConfirmPassword)
      @Html.ValidationMessageFor(temp => temp.ConfirmPassword)
      <br />

      @Html.LabelFor(temp => temp.Mobile)
      @Html.TextBoxFor(temp => temp.Mobile)
      @Html.ValidationMessageFor(temp => temp.Mobile)
      <br />

      @Html.LabelFor(temp => temp.Email)
      @Html.TextBoxFor(temp => temp.Email)
      @Html.ValidationMessageFor(temp => temp.Email)
      <br />

      @Html.LabelFor(temp => temp.Address)
      @Html.TextAreaFor(temp => temp.Address)
      @Html.ValidationMessageFor(temp => temp.Address)
      <br />

      @Html.LabelFor(temp => temp.Gender)
      @Html.RadioButtonFor(temp => temp.Gender, "Male") @:Male
      @Html.RadioButtonFor(temp => temp.Gender, "Female") @:Female
      @Html.ValidationMessageFor(temp => temp.Gender)
      <br />

      @Html.LabelFor(temp => temp.Salary)
      @Html.TextBoxFor(temp => temp.Salary)
      @Html.ValidationMessageFor(temp => temp.Salary)
      <br />

      @Html.LabelFor(temp => temp.Country)
      @Html.DropDownListFor(temp => temp.Country, new List<SelectListItem>{
    
```

```

new SelectListItem() { Text = "India" },
new SelectListItem() { Text = "United States", Value = "USA" },
new SelectListItem() { Text = "Japan", Value = "JPN" },
}, "Please Select")
@Html.ValidationMessageFor(temp => temp.Country)
<br />

@Html.CheckBoxFor(temp => temp.AcceptLicenseAgreement)
@Html.LabelFor(temp => temp.AcceptLicenseAgreement)<br />

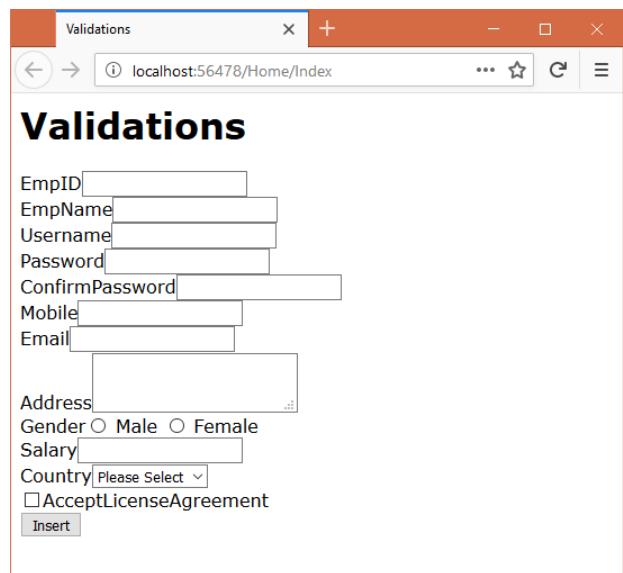
<input type="submit" value="Insert" />
@Html.ValidationSummary()
}
</body>
</html>

```

#### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

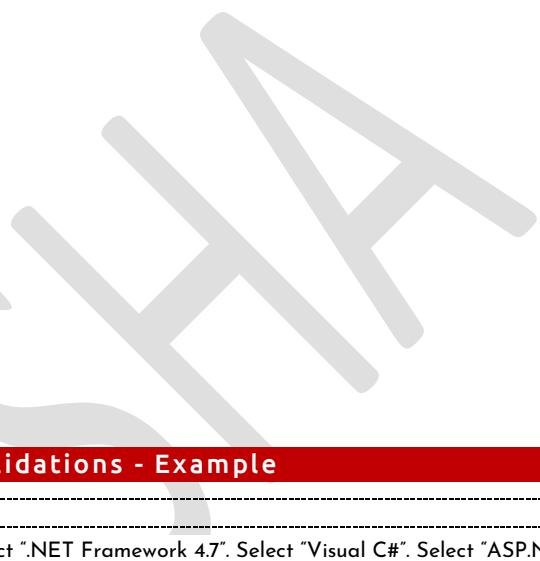
Output:



The screenshot shows a browser window titled "Validations" with the URL "localhost:56478/Home/Index". The page displays an ASP.NET MVC form for inserting employee data. The form fields include:

- EmpID
- EmpName
- Username
- Password
- ConfirmPassword
- Mobile
- Email
- Address
- Gender: Male (radio button)
- Salary
- Country: Please Select (dropdown menu)
- AcceptLicenseAgreement (checkbox)
- Insert button

Click on "Insert" button, without entering anything.



**Validations**

EmpID  Emp ID can't be blank.  
 EmpName  Emp Name can't be blank.  
 Username  Username can't be blank.  
 Password  Password can't be blank.  
 ConfirmPassword  Confirm Password can't be blank.  
 Mobile  Mobile can't be blank.  
 Email  Email can't be blank.

Address

Gender  Male  Female Gender can't be blank.  
 Salary  Salary can't be blank.  
 Country  Please Select Country can't be blank.  
 AcceptLicenseAgreement

• Emp ID can't be blank.  
 • Emp Name can't be blank.  
 • Username can't be blank.  
 • Password can't be blank.  
 • Confirm Password can't be blank.  
 • Mobile can't be blank.  
 • Email can't be blank.  
 • Salary can't be blank  
 • Country can't be blank.  
 • Gender can't be blank.

## Client Side Validations - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "ValidationsExample". Type the location as "C:\Mvc". Type the solution name as "ValidationsExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "htmlhelpersdatabase" database already exists. Click on "New Query".
- Type the following code:

```

create database htmlhelpersdatabase
go
use htmlhelpersdatabase
go
create table Employees(
EmpID int primary key,
EmpName nvarchar(max),
Username nvarchar(max),
Password nvarchar(max),
Mobile nvarchar(max),
Email nvarchar(max),
Address nvarchar(max),
Gender nvarchar(max),
Salary decimal,
Country nvarchar(max),
AcceptLicenseAgreement bit not null)
go

```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

#### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```

install-package EntityFramework
install-package jQuery
install-package jQuery.Validation
install-package Microsoft.jQuery.Unobtrusive.Validation

```

#### Creating Unique.cs

- Right click on the project (ValidationsExample), and click on "Add" - "New Folder". Type the folder name as "CustomValidations" and press Enter. Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Unique.cs". Click on "Add".

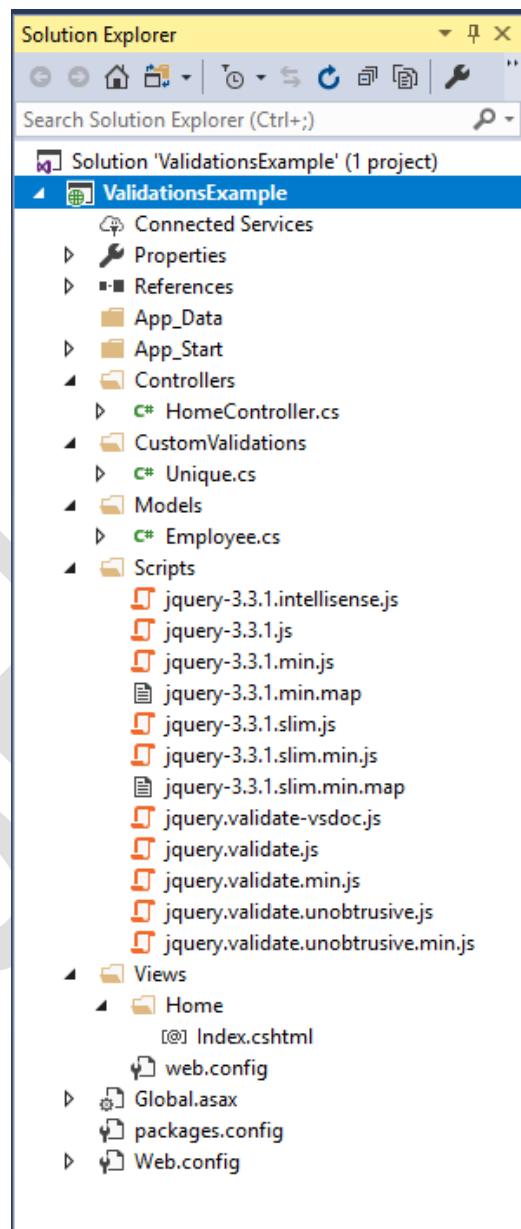
#### Code for "CustomValidations\Unique.cs"

```

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using ValidationsExample.Models;

namespace ValidationsExample.CustomValidations
{
    public class UniqueAttribute : ValidationAttribute
    {
        protected override ValidationResult IsValid(object value, ValidationContext validationContext)
        {
            HtmlHelpersDatabaseDbContext db = new HtmlHelpersDatabaseDbContext();
            string currentusername = Convert.ToString(value);
            if (db.Employees.Where(temp => temp.Username == currentusername).Count() == 0)
            {
                return ValidationResult.Success;
            }
            else
            {
                return new ValidationResult(this.ErrorMessage);
            }
        }
    }
}

```



```
}
```

#### **Creating Employee.cs**

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

#### **Code for "Employee.cs"**

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Data.Entity;
using ValidationsExample.CustomValidations;

namespace ValidationsExample.Models
{
    public class Employee
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        [Required(ErrorMessage = "Emp ID can't be blank.")]
        public int EmpID { get; set; }

        [Required(ErrorMessage = "Emp Name can't be blank.")]
        [RegularExpression(@"^([a-zA-Z ]*)$", ErrorMessage = "Emp Name should contain alphabets only.")]
        public string EmpName { get; set; }

        [Required(ErrorMessage = "Username can't be blank.")]
        [Unique(ErrorMessage = "Duplicate username")]
        public string Username { get; set; }

        [Required(ErrorMessage = "Password can't be blank.")]
        public string Password { get; set; }

        [Required(ErrorMessage = "Confirm Password can't be blank.")]
        [Compare("Password", ErrorMessage = "Password and confirm password do not match.")]
        [NotMapped]
        public string ConfirmPassword { get; set; }

        [Required(ErrorMessage = "Mobile can't be blank.")]
        public string Mobile { get; set; }

        [Required(ErrorMessage = "Email can't be blank.")]
        [RegularExpression(@"^(w+@[a-zA-Z_]+?\.[a-zA-Z]{2,6})$", ErrorMessage = "Invalid E-Mail.")]
        public string Email { get; set; }

        public string Address { get; set; }

        [Required(ErrorMessage = "Gender can't be blank.")]
        public string Gender { get; set; }

        [Range(1000, 50000, ErrorMessage = "Salary should be in between 1000 and 50000.")]
        [Required(ErrorMessage = "Salary can't be blank")]
        public decimal Salary { get; set; }

        [Required(ErrorMessage = "Country can't be blank.")]
        public string Country { get; set; }

        public bool AcceptLicenseAgreement { get; set; }
    }

    public class HtmlHelpersDatabaseDbContext : DbContext
    {
```

```

public HtmlHelpersDatabaseDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=htmlhelpersdatabase")
{
}

public DbSet<Employee> Employees { get; set; }
}

```

#### **Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### **Code for "HomeController.cs"**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Mvc;
using ValidationsExample.Models;

namespace ValidationsExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        HtmlHelpersDatabaseDbContext db = new HtmlHelpersDatabaseDbContext();

        [HttpPost]
        public ActionResult InsertEmp(Employee emp)
        {
            if (ModelState.IsValid == true)
            {
                db.Employees.Add(emp);
                db.SaveChanges();
                return Content("Successfully Inserted");
            }
            else
            {
                return View("Index");
            }
        }
    }
}

```

#### **Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### **Code for "Views\Home\Index.cshtml"**

```

@model ValidationsExample.Models.Employee
<html>
<head>
    <title>Validations</title>
    <script src="~/Scripts/jquery-3.3.1.js"></script>
    <script src="~/Scripts/jquery.validate.js"></script>
    <script src="~/Scripts/jquery.validate.unobtrusive.js"></script>
    <style type="text/css">
        .field-validation-error, .validation-summary-errors

```

```

{
    color: red;
}
</style>
</head>
<body>
<h1>Validations</h1>
@using (Html.BeginForm("InsertEmp", "Home", FormMethod.Post))
{
    @Html.LabelFor(temp => temp.EmpID)
    @Html.TextBoxFor(temp => temp.EmpID)
    @Html.ValidationMessageFor(temp => temp.EmpID)
    <br />

    @Html.LabelFor(temp => temp.EmpName)
    @Html.TextBoxFor(temp => temp.EmpName)
    @Html.ValidationMessageFor(temp => temp.EmpName)
    <br />

    @Html.LabelFor(temp => temp.Username)
    @Html.TextBoxFor(temp => temp.Username)
    @Html.ValidationMessageFor(temp => temp.Username)
    <br />

    @Html.LabelFor(temp => temp.Password)
    @Html.PasswordFor(temp => temp.Password)
    @Html.ValidationMessageFor(temp => temp.Password)
    <br />

    @Html.LabelFor(temp => temp.ConfirmPassword)
    @Html.PasswordFor(temp => temp.ConfirmPassword)
    @Html.ValidationMessageFor(temp => temp.ConfirmPassword)
    <br />

    @Html.LabelFor(temp => temp.Mobile)
    @Html.TextBoxFor(temp => temp.Mobile)
    @Html.ValidationMessageFor(temp => temp.Mobile)
    <br />

    @Html.LabelFor(temp => temp.Email)
    @Html.TextBoxFor(temp => temp.Email)
    @Html.ValidationMessageFor(temp => temp.Email)
    <br />

    @Html.LabelFor(temp => temp.Address)
    @Html.TextAreaFor(temp => temp.Address)
    @Html.ValidationMessageFor(temp => temp.Address)
    <br />

    @Html.LabelFor(temp => temp.Gender)
    @Html.RadioButtonFor(temp => temp.Gender, "Male") @:Male
    @Html.RadioButtonFor(temp => temp.Gender, "Female") @:Female
    @Html.ValidationMessageFor(temp => temp.Gender)
    <br />

    @Html.LabelFor(temp => temp.Salary)
    @Html.TextBoxFor(temp => temp.Salary)
    @Html.ValidationMessageFor(temp => temp.Salary)
    <br />

    @Html.LabelFor(temp => temp.Country)
}

```

```

@Html.DropDownListFor(temp => temp.Country, new List<SelectListItem>{
    new SelectListItem() {Text = "India", Value = "INDIA", Selected = true},
    new SelectListItem() {Text = "United States", Value = "USA", Selected = false},
    new SelectListItem() {Text = "Japan", Value = "JPN", Selected = false},
}, "Please Select")
@Html.ValidationMessageFor(temp => temp.Country)
<br />

@Html.CheckBoxFor(temp => temp.AcceptLicenseAgreement)
@Html.LabelFor(temp => temp.AcceptLicenseAgreement)<br />

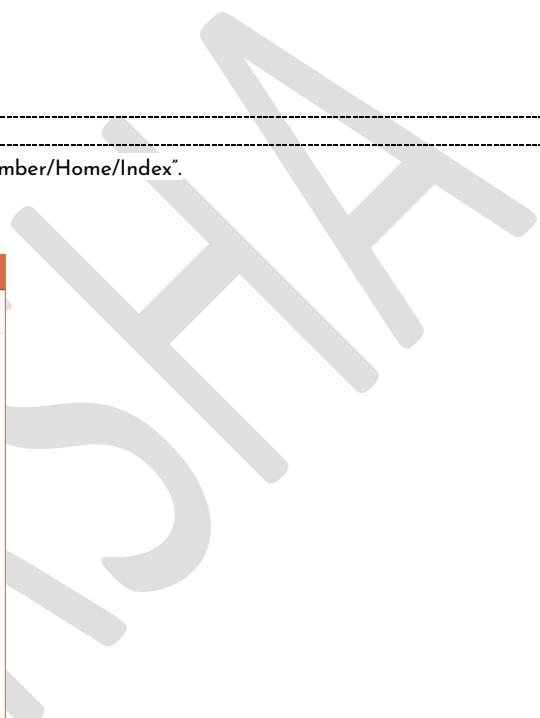
<input type="submit" value="Insert"/>
@Html.ValidationSummary()
}
</body>
</html>

```

#### Running the application

- Press "F5" to run the application. Type "http://localhost:portnumber/Home/Index".

Output:



Screenshot of a web browser showing the validation errors for an employee insertion form. The URL in the address bar is "localhost:56478/Home/InsertEmp". The page title is "Validations". The form fields and their validation messages are:

- EmpID: Emp ID can't be blank.
- EmpName: Emp Name can't be blank.
- Username: Username can't be blank.
- Password: Password can't be blank.
- ConfirmPassword: Confirm Password can't be blank.
- Mobile: Mobile can't be blank.
- Email: Email can't be blank.
- Address: Address can't be blank.
- Gender: Gender can't be blank. (radio buttons for Male and Female)
- Salary: Salary can't be blank.
- Country: Country can't be blank. (dropdown menu with placeholder "Please Select")
- AcceptLicenseAgreement: Accept License Agreement checkbox (unchecked)

The "Insert" button is at the bottom left of the form.

Below the form, a list of validation errors is displayed:

- Emp ID can't be blank.
- Emp Name can't be blank.
- Username can't be blank.
- Password can't be blank.
- Confirm Password can't be blank.
- Mobile can't be blank.
- Email can't be blank.
- Salary can't be blank
- Country can't be blank.
- Gender can't be blank.

Click on "Insert" button, without entering anything.

## SECURITY

### Forms Based Authentication

- “Forms Based Authentication” (also known as Forms based security) is used to avoid accessing the web pages without login. For example, you can access Gmail inbox, sent, compose etc., pages only after login. If the user send a request to a page but not logged in already, it should redirect to the login page automatically.
- ASP.NET identifies the user whether he isloggedin or not, by using "Authentication Cookie". The "Authentication Cookie" stores the username in the client side (browser).

#### Set set Authentication Mode (web.config)

```
<authentication mode="Forms">
  <forms loginUrl="Home/Login"></forms>
</authentication>
```

#### Create Authentication Cookie (Login)

```
System.Web.Security.FormsAuthentication.SetAuthCookie("username", false);
```

#### Delete the Authentication cookie (Logout)

```
System.Web.Security.FormsAuthentication.SignOut();
```

#### Verify the username whether logged-in

```
[Authorize]
```

## Forms Authentication - Example

#### Creating Project

- Open Visual Studio 2017. Go to “File” – “New” – “Project”. Select “.NET Framework 4.7”. Select “Visual C#”. Select “ASP.NET Web Application (.NET Framework)”. Type the project name “FormsAuthenticationExample”. Type the location as “C:\Mvc”. Type the solution name as “FormsAuthenticationExample”. Click on OK. Select “Empty”. Check the checkbox “MVC”. Uncheck the checkbox “Enable Docker support”. Uncheck the checkbox “Add unit tests”. Click on OK.

### **Creating Database**

- Open SQL Server Management Studio. Select "Windows Authentication".
- Click on "Connect".
- Ignore this step if "useraccountsdatabase" database already exists.
- Click on "New Query".
- Type the following code:

```

create database useraccountsdatabase
go

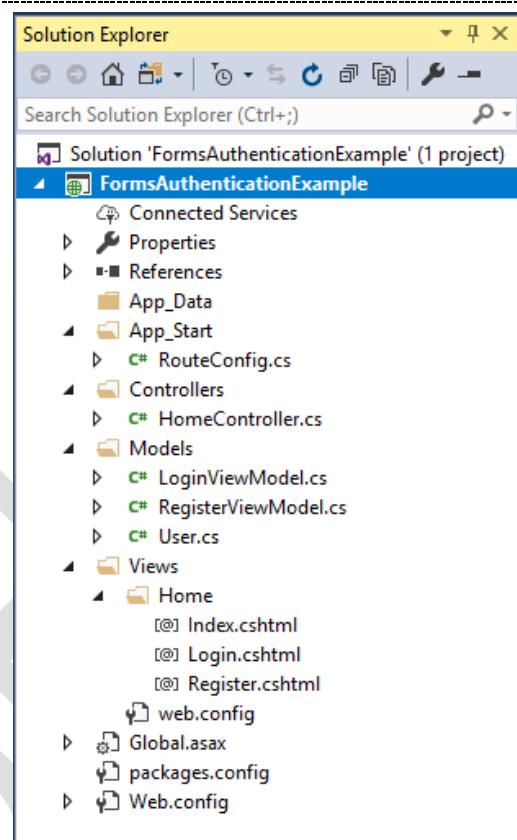
use useraccountsdatabase
go

create table Users(
    userid int primary key identity(1,1),
    username nvarchar(max),
    password nvarchar(max),
    email nvarchar(max))
go

insert into Users values('scott', 'scott123', 'scott@gmail.com')
insert into Users values('john', 'john123', 'john@gmail.com')
insert into Users values('jones', 'jones123', 'jones@gmail.com')
go

```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.



### **Installing Packages**

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Type the following command in "Package Manager Console" and press Enter.  
`install-package EntityFramework`

### **Creating User.cs**

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "User.cs". Click on "Add".

#### **Code for "User.cs"**

```

using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;

namespace FormsAuthenticationExample.Models
{
    public class User
    {
        [Key]
        public int UserID { get; set; }
        public string Username { get; set; }
        public string Password { get; set; }
    }

    public class UserAccountsDbContext : DbContext
    {
        public UserAccountsDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=useraccountsdatabase")
    }
}

```

```

    public DbSet<User> Users { get; set; }
}
}

```

#### **[Creating LoginViewModel.cs]**

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "LoginViewModel.cs". Click on "Add".

#### **Code for "LoginViewModel.cs"**

```

using System;
using System.ComponentModel.DataAnnotations;

namespace FormsAuthenticationExample.Models
{
    public class LoginViewModel
    {
        [Required]
        public string Username { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}

```

#### **[Creating RegisterViewModel.cs]**

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "RegisterViewModel.cs". Click on "Add".

#### **Code for "RegisterViewModel.cs"**

```

using System;
using System.ComponentModel.DataAnnotations;

namespace FormsAuthenticationExample.Models
{
    public class RegisterViewModel
    {
        [Required]
        public string Username { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}

```

#### **[Creating HomeController.cs]**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### **Code for "HomeController.cs"**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Mvc;
using FormsAuthenticationExample.Models;

namespace FormsAuthenticationExample.Controllers
{
    public class HomeController : Controller
    {
        UserAccountsDbContext db = new UserAccountsDbContext();

```

```

public ActionResult Register()
{
    return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Register(RegisterViewModel vm)
{
    if (ModelState.IsValid)
    {
        User dm = new User();
        dm.Username = vm.Username;
        dm.Password = vm.Password;
        db.Users.Add(dm);
        db.SaveChanges();
        Session["CurrentUsername"] = vm.Username;
        System.Web.Security.FormsAuthentication.SetAuthCookie(vm.Username, false);
        return RedirectToAction("Index", "Home");
    }
    else
    {
        ModelState.AddModelError("x", "Invalid Username or password");
        return View();
    }
}

[Authorize]
public ActionResult Index()
{
    return View();
}

public ActionResult Logout()
{
    Session.Abandon();
    System.Web.Security.FormsAuthentication.SignOut();
    return RedirectToAction("Login");
}

public ActionResult Login()
{
    return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Login(LoginViewModel vm)
{
    if (ModelState.IsValid)
    {
        List<User> matchingusers = db.Users.Where(temp => temp.Username == vm.Username && vm.Password ==
vm.Password).ToList();
        if (matchingusers.Count == 1)
        {
            Session["CurrentUsername"] = vm.Username;
            System.Web.Security.FormsAuthentication.SetAuthCookie(vm.Username, false);
            return RedirectToAction("Index", "Home");
        }
        else
        {
    
```

```
ModelState.AddModelError("x", "Invalid Username or password");
return View();
}
}
else
{
    ModelState.AddModelError("x", "Invalid Username or password");
    return View();
}
}
}
```

## Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Index.cshtml"**

```
<html>
<head>
    <title>MainPage</title>
</head>
<body>
    <h1>Main Page</h1>
    <h2>Welcome to @Session["CurrentUsername"]</h2>
    <a href="/Home/Logout">Logout</a>
</body>
</html>
```

## Creating Login.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Login". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for “Views\Home\Login.cshtml”**

```
@model FormsAuthenticationExample.Models.LoginViewModel
<html>
<head>
    <title> Login </title>
</head>
<body>
    <h1>Login</h1>
    @using (Html.BeginForm())
    {
        @Html.AntiForgeryToken()
        @Html.ValidationSummary()

        <table>
            <tr>
                <td>@Html.LabelFor(model => model.Username)</td>
                <td>
                    @Html.EditorFor(model => model.Username)
                    @Html.ValidationMessageFor(model => model.Username)
                </td>
            </tr>
            <tr>
                <td>@Html.LabelFor(model => model.Password)</td>
                <td>
                    @Html.EditorFor(model => model.Password)
                    @Html.ValidationMessageFor(model => model.Password)
                </td>
            </tr>
            <tr>
                <td></td>
                <td>
                    <input type="submit" value="Log In" />
                </td>
            </tr>
        </table>
    }

```

```

<td></td>
<td><input type="submit" value="Login" /></td>
</tr>
</table>
<a href="/Home/Register">Go to register page</a>
}
</body>
</html>

```

#### **Creating Register.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Register". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### **Code for "Views\Home\Register.cshtml"**

```

@model FormsAuthenticationExample.Models.RegisterViewModel
<html>
<head>
<title>Register</title>
</head>
<body>
<h1>Register</h1>
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()
    @Html.ValidationSummary()

    <table>
        <tr>
            <td>@Html.LabelFor(model => model.Username)</td>
            <td>
                @Html.EditorFor(model => model.Username)
                @Html.ValidationMessageFor(model => model.Username)
            </td>
        </tr>
        <tr>
            <td>@Html.LabelFor(model => model.Password)</td>
            <td>
                @Html.EditorFor(model => model.Password)
                @Html.ValidationMessageFor(model => model.Password)
            </td>
        </tr>
        <tr>
            <td></td>
            <td><input type="submit" value="Register" /></td>
        </tr>
    </table>
    <a href="/Home/Login">Go to login page</a>
}
</body>
</html>

```

#### **Add Code in Web.Config**

- Open "Web.config" file and add the following code in <system.web> tag.

#### **Code for "Web.config"**

```

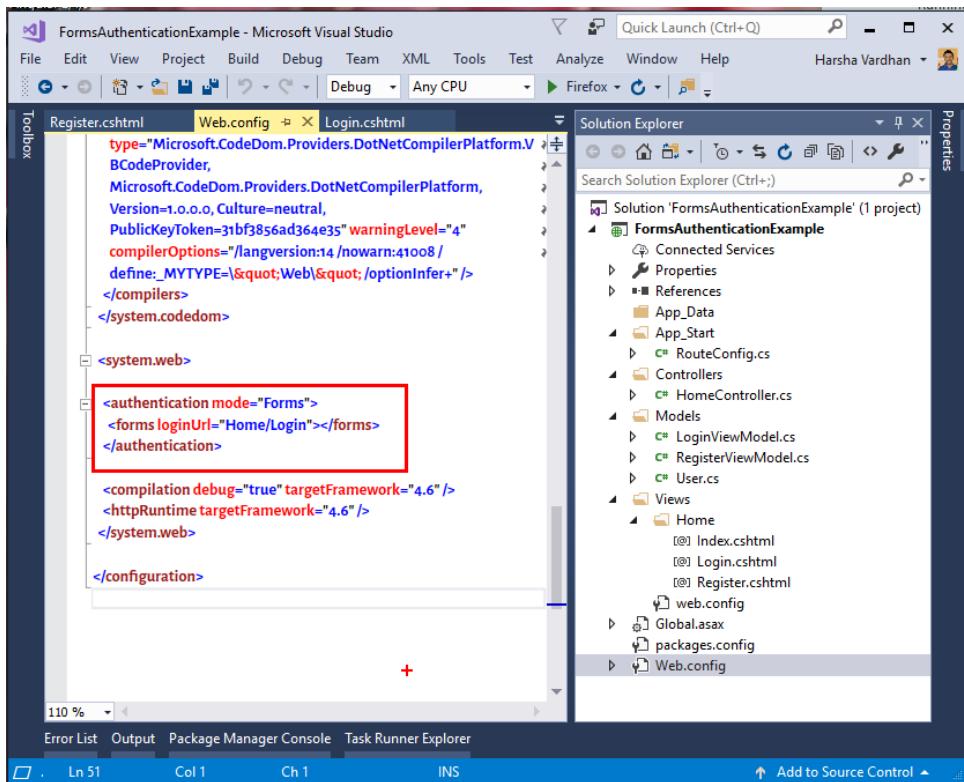
<system.web>

<authentication mode="Forms">
<forms loginUrl="Home/Login"></forms>
</authentication>

<compilation debug="true" targetFramework="4.6" />

```

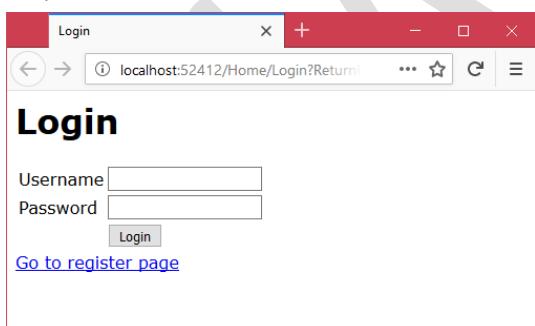
```
<httpRuntime targetFramework="4.6" />  
</system.web>
```



### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



Try registration and login process.

## CSRF

### CSRF

- "AntiForgeryToken" is an attribute, which is used to avoid "XSRF" / "CSRF" (Cross Site Request Forgery).
- "XSRF" means, sending a request to your website, by creating a similar form in other website. For example, sending a request to bank website, by creating similar funds transfer form in other attacker website.

### How to enable AntiForgeryToken

- Use the following html help in the view (inside the form):

```
@Html.AntiForgeryToken()
```

- Use the following action filter in the controller:

```
[ValidateAntiForgeryToken]
```

### How AntiForgeryToken Works?

- The "AntiForgeryToken" method generates a random key (alphanumeric) and the same will be sent to the browser as both "hidden field" and "cookie". When the form gets submitted to server, both "hidden field" and "cookie" will be submitted to the server.
- In controller, the "ValidateAntiForgeryToken" attribute checks whether both "hidden field" and "cookie" matching or not. If matching, the request will be treated as genuine request and the action method executes; otherwise the request will be assumed as malicious and will be rejected. In realtime, it is recommended to use this concept in every form, if you are submitting it to server.

## CSRF - Example

### Creating Project

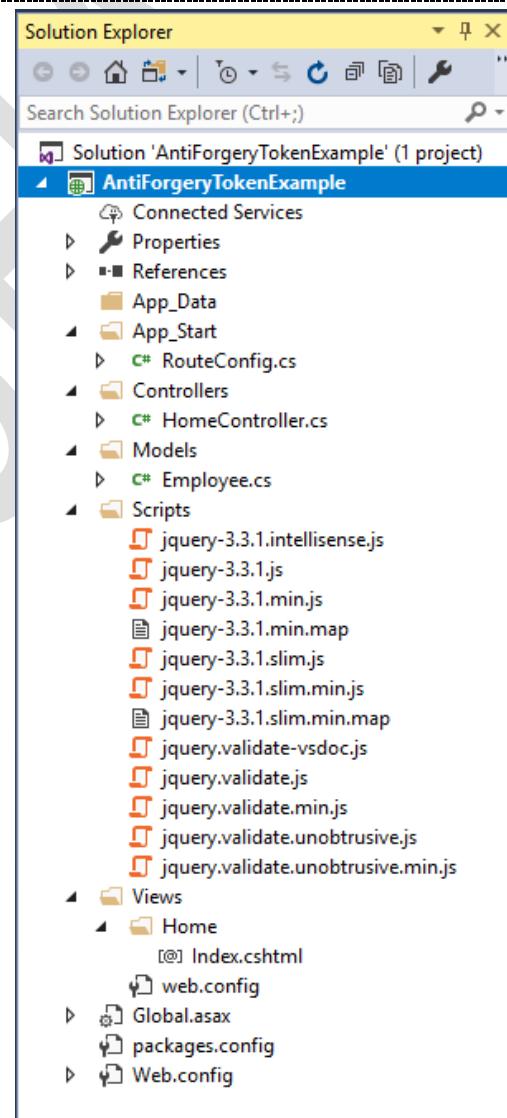
- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name as "AntiForgeryTokenExample". Type the location as "C:\Mvc". Type the solution name as "AntiForgeryTokenExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "htmlhelpersdatabase" database already exists. Click on "New Query". Type the following code:

```
create database htmlhelpersdatabase
go
use htmlhelpersdatabase
go
create table Employees(
    EmpID int primary key,
    EmpName nvarchar(max),
    Username nvarchar(max),
    Password nvarchar(max),
    Mobile nvarchar(max),
    Email nvarchar(max),
    Address nvarchar(max),
    Gender nvarchar(max),
    Salary decimal,
    Country nvarchar(max),
    AcceptLicenseAgreement bit not null)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.



### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
install-package jQuery
install-package jQuery.Validation
install-package Microsoft.jQuery.Unobtrusive.Validation
```

### Creating Employee.cs

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

**Code for "Employee.cs"**

```

using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Data.Entity;

namespace AntiForgeryTokenExample.Models
{
    public class Employee
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        [Required(ErrorMessage = "Emp ID can't be blank.")]
        public int EmpID { get; set; }

        [Required(ErrorMessage = "Emp Name can't be blank.")]
        [RegularExpression(@"^([a-zA-Z]+)$", ErrorMessage = "Emp Name should contain alphabets only.")]
        public string EmpName { get; set; }

        [Required(ErrorMessage = "Username can't be blank.")]
        public string Username { get; set; }

        [Required(ErrorMessage = "Password can't be blank.")]
        public string Password { get; set; }

        [Required(ErrorMessage = "Confirm Password can't be blank.")]
        [Compare("Password", ErrorMessage = "Password and confirm password do not match.")]
        [NotMapped]
        public string ConfirmPassword { get; set; }

        [Required(ErrorMessage = "Mobile can't be blank.")]
        public string Mobile { get; set; }

        [Required(ErrorMessage = "Email can't be blank.")]
        [RegularExpression(@"^(\\w+@[a-zA-Z_]+?\\.[a-zA-Z]{2,6})$", ErrorMessage = "Invalid E-Mail.")]
        public string Email { get; set; }

        public string Address { get; set; }

        [Required(ErrorMessage = "Gender can't be blank.")]
        public string Gender { get; set; }

        [Range(1000, 50000, ErrorMessage = "Salary should be in between 1000 and 50000.")]
        public decimal Salary { get; set; }

        [Required(ErrorMessage = "Country can't be blank.")]
        public string Country { get; set; }

        public bool AcceptLicenseAgreement { get; set; }
    }

    public class HtmlHelpersDatabaseDbContext : DbContext
    {
        public HtmlHelpersDatabaseDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=htmlhelpersdatabase")
        {
        }

        public DbSet<Employee> Employees { get; set; }
    }
}

```

### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### Code for "HomeController.cs"

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

namespace AntiForgeryTokenExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        HtmlHelpersDatabaseDbContext db = new HtmlHelpersDatabaseDbContext();

        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult InsertEmp(Employee emp)
        {
            if (ModelState.IsValid == true)
            {
                db.Employees.Add(emp);
                db.SaveChanges();
                return Content("Successfully Inserted");
            }
            else
            {
                return View("Index");
            }
        }
    }
}
```

### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### Code for "Views\Home\Index.cshtml"

```
@model AntiForgeryTokenExample.Models.Employee
<html>
<head>
    <title>Anti Forgery Token</title>
    <script src="~/Scripts/jquery-3.3.1.js"></script>
    <script src="~/Scripts/jquery.validate.js"></script>
    <script src="~/Scripts/jquery.validate.unobtrusive.js"></script>
    <style type="text/css">
        .field-validation-error, .validation-summary-errors
    {
        color: red;
    }
    </style>
</head>
<body>
    <h1>Anti Forgery Token</h1>
```

```

@using (Html.BeginForm("InsertEmp", "Home", FormMethod.Post))
{
    @Html.AntiForgeryToken()

    @Html.LabelFor(temp => temp.EmpID)
    @Html.TextBoxFor(temp => temp.EmpID)
    @Html.ValidationMessageFor(temp => temp.EmpID)
    <br />

    @Html.LabelFor(temp => temp.EmpName)
    @Html.TextBoxFor(temp => temp.EmpName)
    @Html.ValidationMessageFor(temp => temp.EmpName)
    <br />

    @Html.LabelFor(temp => temp.Username)
    @Html.TextBoxFor(temp => temp.Username)
    @Html.ValidationMessageFor(temp => temp.Username)
    <br />

    @Html.LabelFor(temp => temp.Password)
    @Html.PasswordFor(temp => temp.Password)
    @Html.ValidationMessageFor(temp => temp.Password)
    <br />

    @Html.LabelFor(temp => temp.ConfirmPassword)
    @Html.PasswordFor(temp => temp.ConfirmPassword)
    @Html.ValidationMessageFor(temp => temp.ConfirmPassword)
    <br />

    @Html.LabelFor(temp => temp.Mobile)
    @Html.TextBoxFor(temp => temp.Mobile)
    @Html.ValidationMessageFor(temp => temp.Mobile)
    <br />

    @Html.LabelFor(temp => temp.Email)
    @Html.TextBoxFor(temp => temp.Email)
    @Html.ValidationMessageFor(temp => temp.Email)
    <br />

    @Html.LabelFor(temp => temp.Address)
    @Html.TextAreaFor(temp => temp.Address)
    @Html.ValidationMessageFor(temp => temp.Address)
    <br />

    @Html.LabelFor(temp => temp.Gender)
    @Html.RadioButtonFor(temp => temp.Gender, "Male") @:Male
    @Html.RadioButtonFor(temp => temp.Gender, "Female") @:Female
    @Html.ValidationMessageFor(temp => temp.Gender)
    <br />

    @Html.LabelFor(temp => temp.Salary)
    @Html.TextBoxFor(temp => temp.Salary)
    @Html.ValidationMessageFor(temp => temp.Salary)
    <br />

    @Html.LabelFor(temp => temp.Country)
    @Html.DropDownListFor(temp => temp.Country, new List<SelectListItem>{
        new SelectListItem() {Text = "India", Value = "IND"}, 
        new SelectListItem() {Text = "United States", Value = "USA"}, 
        new SelectListItem() {Text = "Japan", Value = "JPN"}, 
    }, "Please Select")
}

```

```

@Html.ValidationMessageFor(temp => temp.Country)
<br />

@Html.CheckBoxFor(temp => temp.AcceptLicenseAgreement)
@Html.LabelFor(temp => temp.AcceptLicenseAgreement)<br />

<input type="submit" value="Insert" />
@Html.ValidationSummary()
}
</body>
</html>

```

#### Running the application

- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:

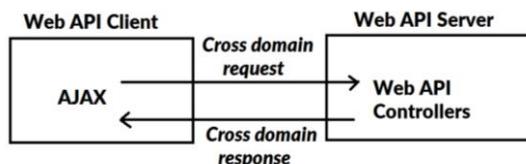
The screenshot shows a Microsoft Edge browser window with the title 'Anti Forgery Token'. The address bar displays 'localhost:53263/Home/Index'. The page content is a form titled 'Anti Forgery Token' with the following fields:

- EmpID
- EmpName
- Username
- Password
- ConfirmPassword
- Mobile
- Email
- Address
- Gender: Male (radio button)
- Salary
- Country: Please Select (dropdown menu)
- AcceptLicenseAgreement
- Insert** button

## CORS

#### Web API - Cross Domain

- This is a concept of creating Web API controllers (HTTP Services) in one project and calling the same from other project. That means, we will create two projects:
  - **Web API Server:** Web API application, contains Web API controllers.
  - **Web API Client:** It is MVC app, we will call "Web API controllers", using AJAX.
- We are accessing the Web API services across the domains.



#### The "Microsoft.AspNet.WebApi.Cors" package

- CORS stands for "Cross Origin Resource Sharing". The "Microsoft.AspNet.WebApi.Cors" is a NuGet package, which is used to give permission to client app, to send cross-domain ajax calls.

## CORS – Example - Server Application

### Creating Project

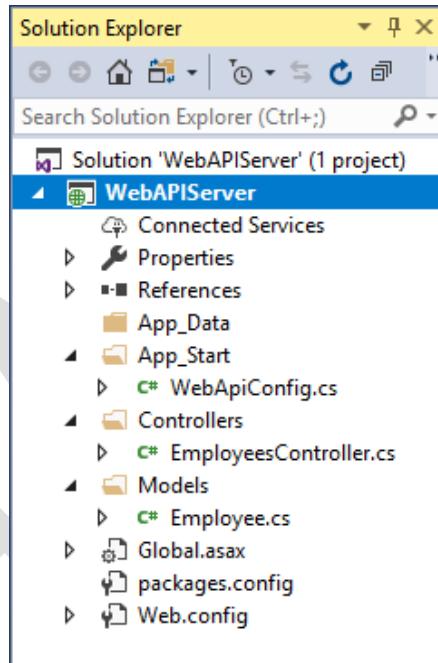
- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "WebAPIServer". Type the location as "C:\Mvc". Type the solution name as "WebAPIServer". Click on OK. Select 'Empty'. Check the checkbox "Web API". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Creating Database

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```
create database company
go
use company
go
create table Employees(
    EmpID int primary key,
    EmpName nvarchar(max),
    Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.



### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package EntityFramework
install-package Microsoft.AspNet.Cors
```

### Creating Employee.cs

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

#### Code for "Employee.cs"

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;

namespace WebAPIServer.Models
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }

        public DbSet<Employee> Employees { get; set; }
    }
}
```

```

    }
}

```

#### Code for WebApiConfig.cs

- Open "App\_Start" - "WebApiConfig.cs".

#### Code for "WebApiConfig.cs"

```

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

namespace WebAPIServer
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            config.MapHttpAttributeRoutes();
            config.EnableCors();

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );

            config.Formatters.Remove(config.Formatters.XmlFormatter());
        }
    }
}

```

#### Creating EmployeesController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "Web API 2 Controller - Empty". Click on "Add". Type the controller name as "EmployeesController". Click on "Add".

#### Code for "EmployeesController.cs"

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Http;
using System.Web.Http.Cors;
using WebAPIServer.Models;

namespace WebAPIServer.Controllers
{
    [EnableCors(origins: "http://localhost:6060", headers: "*", methods: "*")]
    public class EmployeesController : ApiController
    {
        CompanyDbContext db = new CompanyDbContext();

        public List<Employee> Get()
        {
            List<Employee> emps = db.Employees.ToList();
            return emps;
        }
    }
}

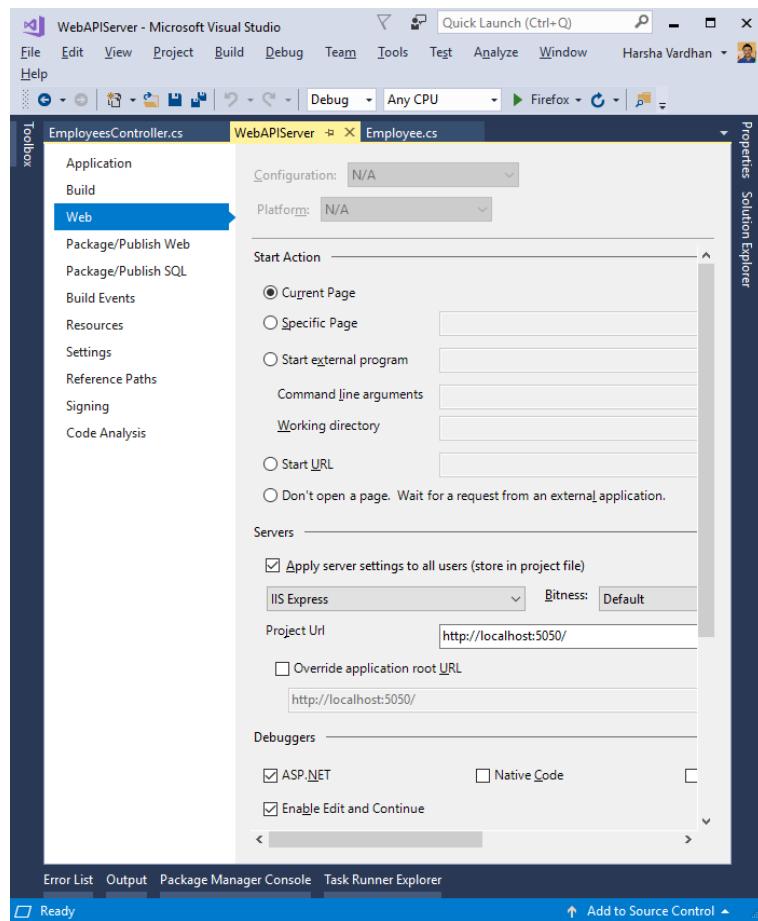
```

//Note: http://localhost:6060 is the address of client app.

## Asp.Net Mvc 5

### Project Properties

- Right click on the project (WebAPIServer) and click on "Properties". Click on "Web". Enter the project url as "http://localhost:5050/".

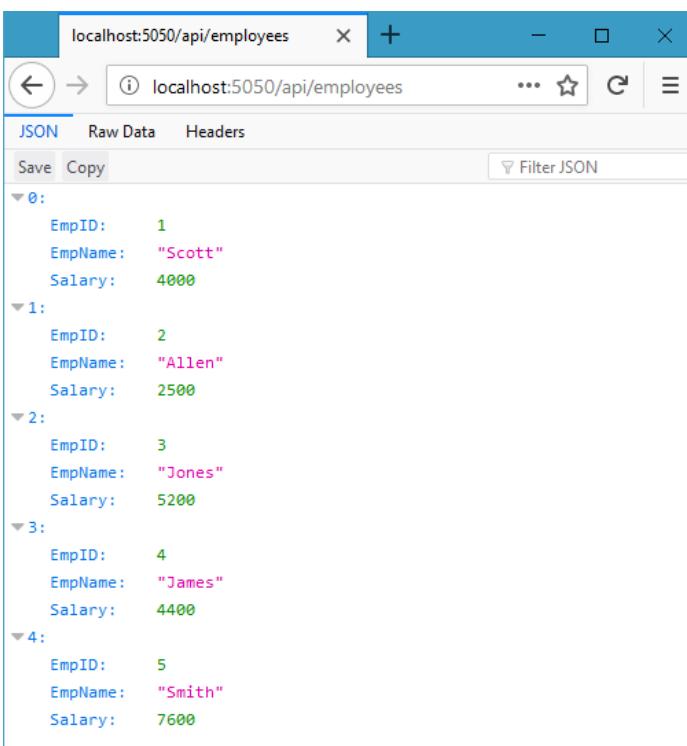


- Close the "WebAPIServer" properties window.

### Running the application

- Go to "Debug" menu - "Start Without Debugging".
- Type "http://localhost:5050/api/employees".

Output:



## CORS – Example – Client Application

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name as "WebAPIClient". Type the location as "C:\Mvc". Type the solution name as "WebAPIClient". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package jQuery
```

### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```

using System;
using System.Web.Mvc;

namespace WebAPIClient.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}

```

### Creating Index.cshtml

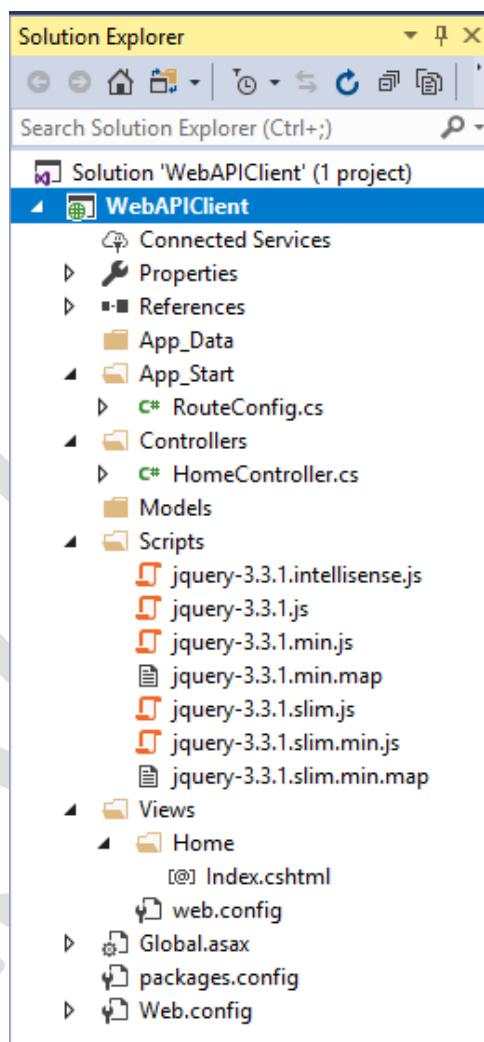
- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Index.cshtml"**

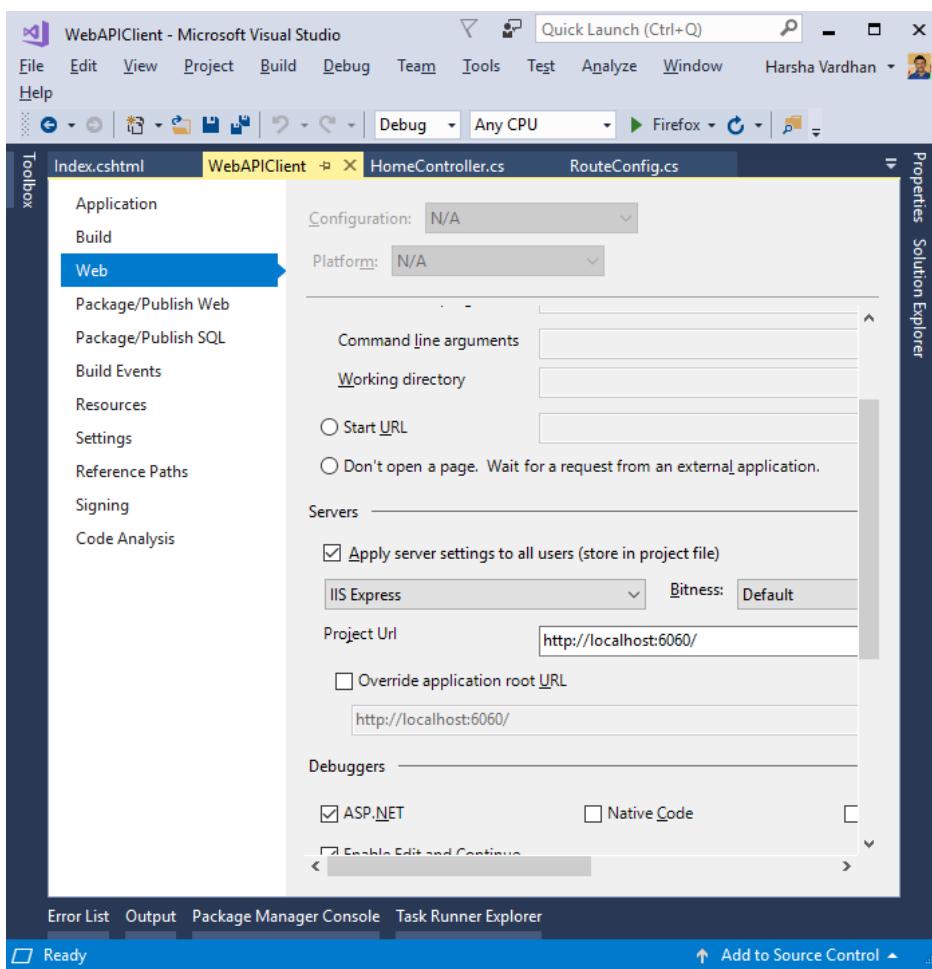
```

<html>
<head>
    <title>CORS</title>
</head>
<body>
    <h1>CORS</h1>
    <input type="button" id="button1" value="Get Data from Web API" />
    <table id="table1" border="1">
        <tr>
            <th>Emp ID</th>
            <th>Emp Name</th>
            <th>Salary</th>
        </tr>
    </table>
    <script src="~/Scripts/jquery-3.3.1.js"></script>
    <script>
        $("#button1").click(fum);
        function fum()
        {
            $.ajax({ url: 'http://localhost:5050/api/Employees', type: 'GET',
            success: fun2, error: fun3 });
        }
        function fun2(response)
        {
            for (var i = 0; i < response.length; i++)
            {
                var emp = response[i];
                var temp = "<tr><td>" + emp.EmpID + "</td><td>" +
                emp.EmpName + "</td><td>" + emp.Salary + "</td></tr>";
                $("#table1").append(temp);
            }
        }
        function fun3(xhr)
        {
            alert(xhr.responseText);
        }
    </script>
</body>
</html>

```

**Project Properties**

- Right click on the project (WebAPIClient) and click on "Properties". Click on "Web". Enter the project url as "http://localhost:6060/".

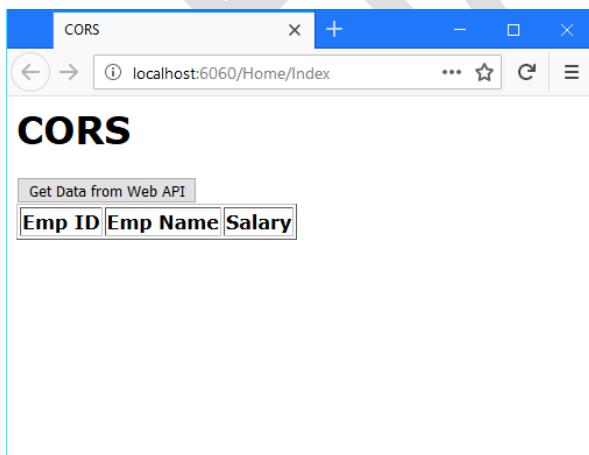


- Close the "WebAPIClient" properties window.

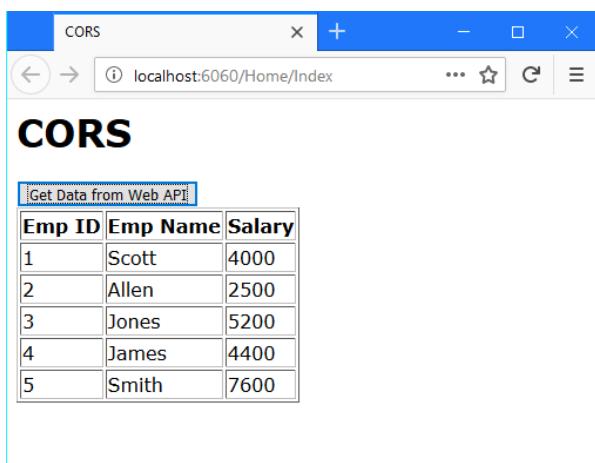
### Running the application

- Go to "Debug" menu - "Start Without Debugging".
- Type "http://localhost:6060/Home/Index".

Output:

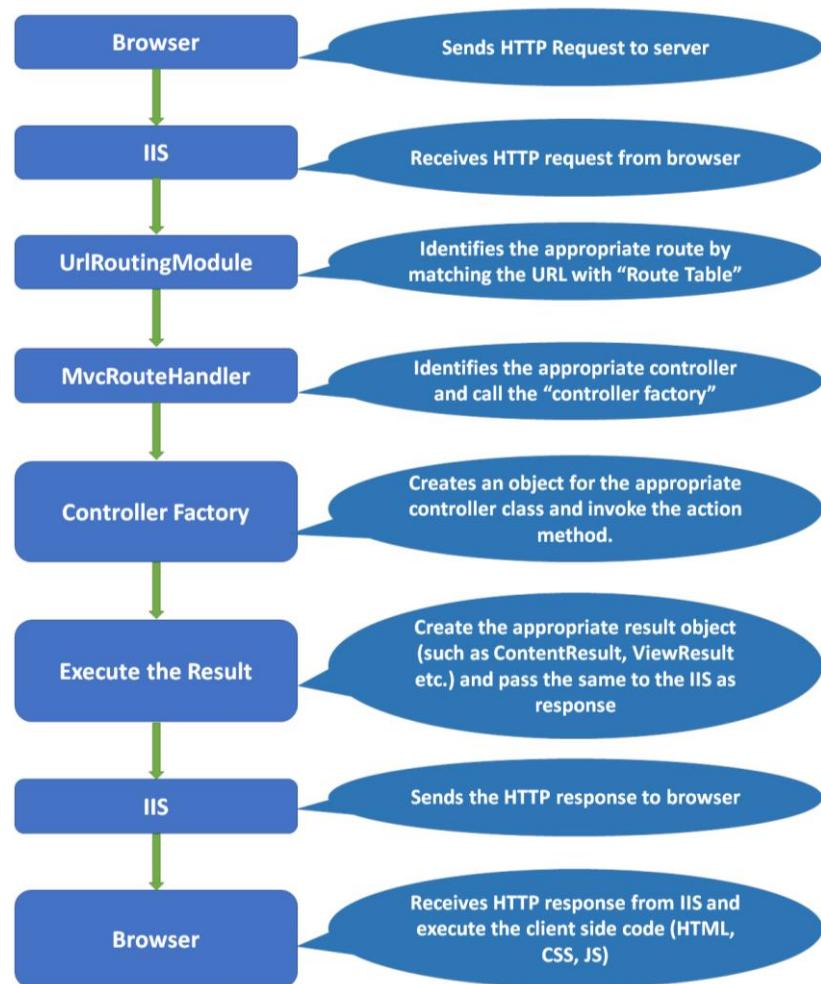


Click on "Get Data from Web API".



## REQUEST LIFE CYCLE

- This diagram explains the complete process, how the request is processed in ASP.NET MVC.



### UNIT TESTING

#### What is Unit Testing

- "Unit Testing" is a concept of testing the controllers / models, whether it is working correctly or not. As a part of unit testing, we will send some dummy inputs to the controller / model and we can get outputs (results) from the controller / model. Then we will check whether it is giving correct output as per the input or not. There are many unit testing frameworks to perform unit testing:
  - MS UnitTesting
  - xUnit
  - NUnit
- MS UnitTesting is the most easy unit testing framework, which is done with "Assert" class.

#### The "Assert" class

- The "Assert" class provides essential methods for unit testing.
  - Microsoft.VisualStudio.TestTools.UnitTesting.Assert.AreEqual(expected, actual)  
Checks whether expected value and actual value are same.
  - Microsoft.VisualStudio.TestTools.UnitTesting.Assert.AreNotEqual(expected, actual)  
Checks whether expected value and actual value are not same.
  - Microsoft.VisualStudio.TestTools.UnitTesting.Assert.IsNull(actual)  
Checks whether the value is null.
  - Microsoft.VisualStudio.TestTools.UnitTesting.Assert.IsNotNull(actual)  
Checks whether the value is not null.
  - Microsoft.VisualStudio.TestTools.UnitTesting.Assert.IsInstanceOfType(actual, typeof(classname))  
Checks whether the value is an object of the specific class.

### Basic Unit Testing - Example

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "BasicUnitTesting". Type the location as "C:\Mvc". Type the solution name as "BasicUnitTesting". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Check the checkbox "Add unit tests". Click on OK.

#### Creating Calculator.cs

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Calculator.cs". Click on "Add".

#### Code for "Calculator.cs"

```
using System;

namespace BasicUnitTesting.Models
{
    public class Calculator
    {
        public double Add(double a, double b)
        {
            double c;
            c = a + b;
            return c;
        }
    }
}
```

#### Creating CalculatorTest.cs

- Right click on "UnitTest1.cs" file and click on "Rename". Type the filename as "CalcualtorTest.cs" and press Enter.

#### Code for "CalculatorTest.cs"

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
```

```

using BasicUnitTesting.Models;

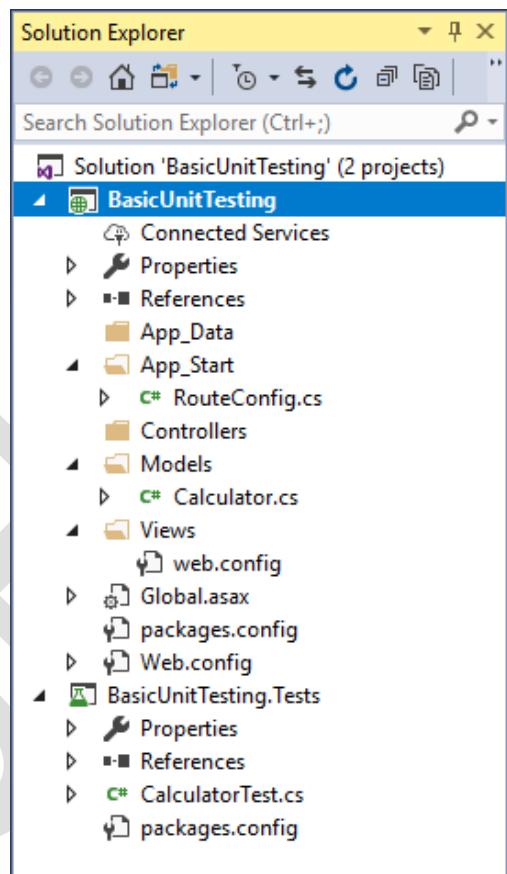
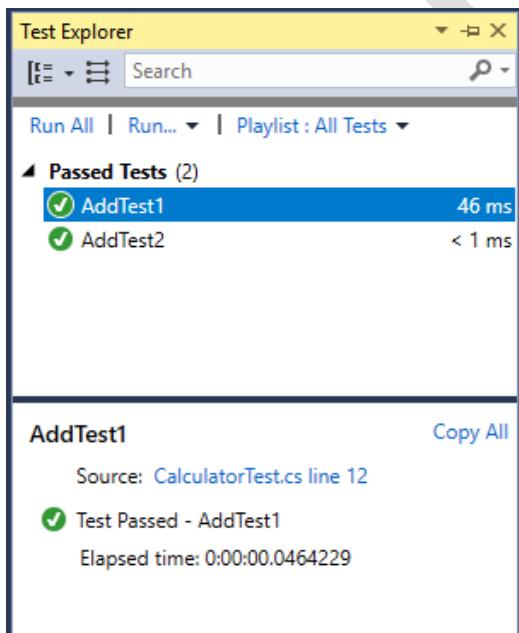
namespace BasicUnitTesting.Tests
{
    [TestClass]
    public class CalculatorTest
    {
        [TestMethod]
        public void AddTest1()
        {
            Calculator c = new Calculator();
            double actualresult = c.Add(10, 20);
            double expectedresult = 30;
            Assert.AreEqual(expectedresult, actualresult);
        }

        [TestMethod]
        public void AddTest2()
        {
            Calculator c = new Calculator();
            double actualresult = c.Add(10.81, 20.12);
            double expectedresult = 30.93;
            Assert.AreEqual(expectedresult, actualresult);
        }
    }
}

```

#### Running the application

- Go to "Test" menu - "Run" - "All Tests".
- Output:



## View Name Testing - Example

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "ViewNameTesting". Type the location as "C:\Mvc". Type the solution name as "ViewNameTesting". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Check the checkbox "Add unit tests". Click on OK.

## Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Select the default project as "ViewNameTesting.Tests".
- Type the following command in "Package Manager Console" and press Enter.  
`install-package Microsoft.AspNet.Mvc`

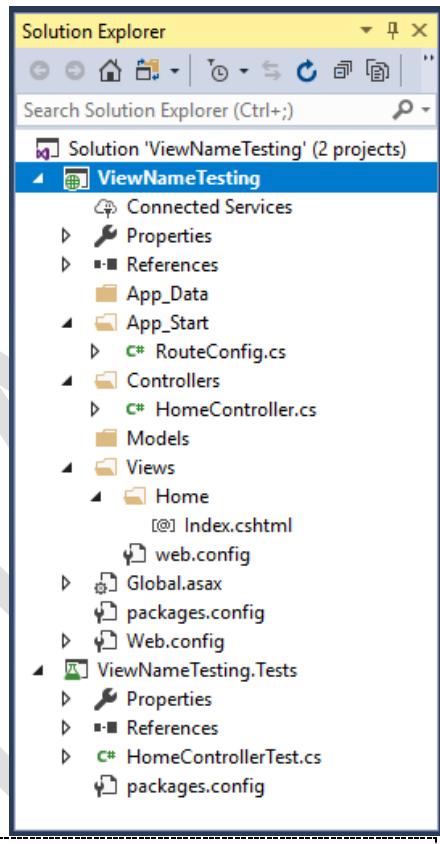
## Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

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

namespace ViewNameTesting.Controllers
{
    public class HomeController : Controller
    {
        public ViewResult Index()
        {
            return View("Index");
        }
    }
}
```



## Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### Code for "Views\Home\Index.cshtml"

```
<html>
<head>
    <title>Index</title>
</head>
<body>
    <h1>Index</h1>
</body>
</html>
```

## Creating HomeControllerTest.cs

- Right click on "UnitTest1.cs" file and click on "Rename". Type the filename as "HomeControllerTest.cs" and press Enter.

### Code for "HomeControllerTest.cs"

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System.Web.Mvc;
using ViewNameTesting.Controllers;

namespace ViewNameTesting.Tests
{
    [TestClass]
    public class HomeControllerTest
    {
        [TestMethod]
        public void IndexTest()
        {
            //Arrange

```

```

HomeController home = new HomeController();

//Act
ViewResult vr = home.Index();

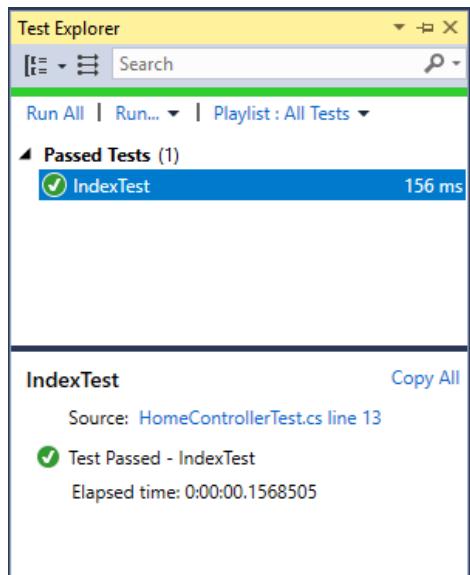
//Assert
string expectedresult = "Index";
Assert.AreEqual(expectedresult, vr.ViewName);
}
}
}

```

#### Running the application

- Go to "Test" menu - "Run" - "All Tests".

Output:



## View Bag Testing - Example

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "ViewBagTesting". Type the location as "C:\Mvc". Type the solution name as "ViewBagTesting". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Check the checkbox "Add unit tests". Click on OK.

#### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Select the default project as "ViewBagTesting.Tests". Type the following command in "Package Manager Console" and press Enter.

```
install-package Microsoft.AspNet.Mvc
```

#### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### Code for "HomeController.cs"

```

using System;
using System.Web.Mvc;

namespace ViewBagTesting.Controllers
{

```

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        int n=5;
        int sq=n * n;
        ViewBag.square=sq;
        return View();
    }
}
```

### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### Code for "Views\Home\Index.cshtml"

```
<html>
<head>
    <title>Index</title>
</head>
<body>
    <p>@ViewBag.square</p>
</body>
</html>
```

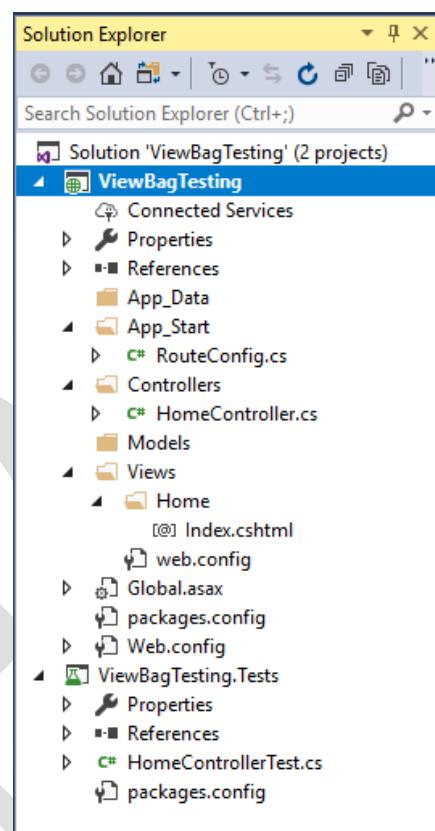
### Creating HomeControllerTest.cs

- Right click on "UnitTest1.cs" file and click on "Rename". Type the filename as "HomeControllerTest.cs" and press Enter.

### Code for "HomeControllerTest.cs"

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System.Web.Mvc;
using ViewBagTesting.Controllers;

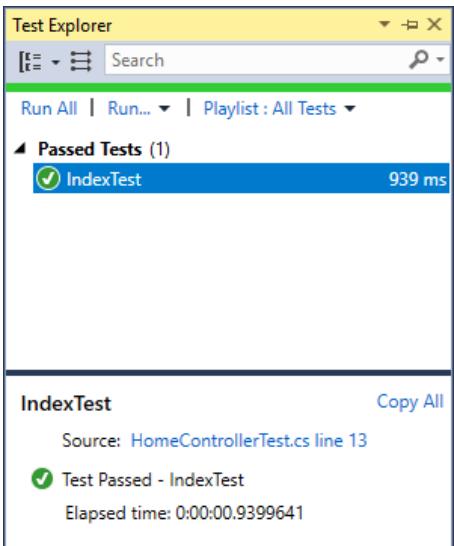
namespace ViewBagTesting.Tests
{
    [TestClass]
    public class HomeControllerTest
    {
        [TestMethod]
        public void IndexTest()
        {
            HomeController home = new HomeController();
            ViewResult vr = home.Index() as ViewResult;
            Assert.AreEqual(25, vr.ViewBag.square);
        }
    }
}
```



### Running the application

- Go to "Test" menu - "Run" - "All Tests".

Output:



### Model Testing - Example

#### **Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "ModelTesting". Type the location as "C:\Mvc". Type the solution name as "ModelTesting". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Check the checkbox "Add unit tests". Click on OK.

#### **Creating Database**

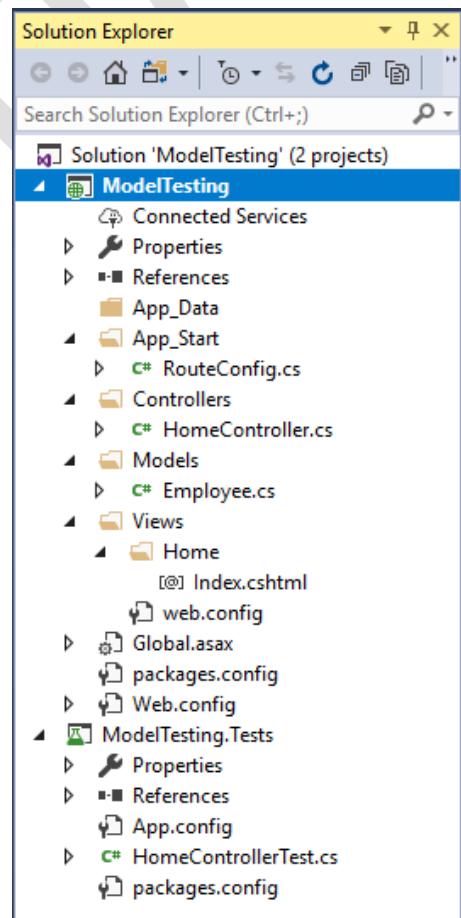
- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```
create database company
go
use company
go
create table Employees(
    EmpID int primary key,
    EmpName nvarchar(max),
    Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

#### **Installing Packages**

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Select the default project as "ModelTesting.Tests". Type the following command in "Package Manager Console" and press Enter.  
`install-package Microsoft.AspNet.Mvc`  
`install-package EntityFramework`
- Select the default project as "ModelTesting". Type the following command in "Package Manager Console" and press Enter.  
`install-package EntityFramework`



### Creating Employee.cs

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

### Code for "Employee.cs"

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;
using System.Collections.Generic;

namespace ModelTesting.Models
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }

        public override bool Equals(object obj)
        {
            Employee other = obj as Employee;
            return (this.EmpID == other.EmpID) && (this.EmpName == other.EmpName) && (this.Salary == other.Salary);
        }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }

        public DbSet<Employee> Employees { get; set; }
    }
}
```

### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

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

namespace ModelTesting.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            CompanyDbContext db = new CompanyDbContext();
            List<Employee> emps = db.Employees.ToList();
            return View(emps);
        }
    }
}
```

### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### Code for "Views\Home\Index.cshtml"

```
<html>
<head>
<title>Index</title>
</head>
<body>
<h1>Index</h1>
</body>
</html>
```

### Creating HomeControllerTest.cs

- Right click on "UnitTest1.cs" file and click on "Rename". Type the filename as "HomeControllerTest.cs" and press Enter.

### Code for "HomeControllerTest.cs"

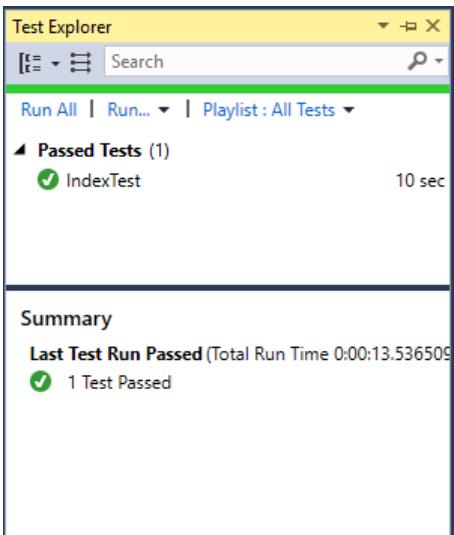
```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System.Web.Mvc;
using ModelTesting.Controllers;
using ModelTesting.Models;
using System.Collections.Generic;
using System.Linq;

namespace ModelTesting.Tests
{
    [TestClass]
    public class HomeControllerTest
    {
        [TestMethod]
        public void IndexTest()
        {
            HomeController home = new HomeController();
            ViewResult vr = home.Index() as ViewResult;
            List<Employee> actualresult = vr.Model as List<Employee>;
            List<Employee> expectedresult = new List<Employee>()
            {
                new Employee() { EmpID = 1, EmpName = "Scott", Salary = 4000 },
                new Employee() { EmpID = 2, EmpName = "Allen", Salary = 2500 },
                new Employee() { EmpID = 3, EmpName = "Jones", Salary = 5200 },
                new Employee() { EmpID = 4, EmpName = "James", Salary = 4400 },
                new Employee() { EmpID = 5, EmpName = "Smith", Salary = 7600 }
            };
            //Assert.AreEqual(expectedresult[0], actualresult[0]);
            Assert.IsTrue(expectedresult.SequenceEqual(actualresult));
        }
    }
}
```

### Running the application

- Go to "Test" menu - "Run" - "All Tests".

Output:



## Repository Testing - Example

**Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "RepositoryTesting". Type the location as "C:\Mvc". Type the solution name as "RepositoryTesting". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Check the checkbox "Add unit tests". Click on OK.

**Creating Database**

- Open SQL Server Management Studio. Select "Windows Authentication". Click on "Connect". Ignore this step if "company" database already exists. Click on "New Query". Type the following code:

```
create database company
go
use company
go
create table Employees(EmpID int primary key, EmpName nvarchar(max),
Salary decimal)
go
insert into Employees values(1, 'Scott', 4000)
insert into Employees values(2, 'Allen', 2500)
insert into Employees values(3, 'Jones', 5200)
insert into Employees values(4, 'James', 4400)
insert into Employees values(5, 'Smith', 7600)
go
```

- Click on "Execute" button. It shows "Query executed successfully" in the status bar.

**Installing Packages**

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Select the default project as "RepositoryTesting.Tests". Type the following command in "Package Manager Console" and press Enter.  
`install-package Microsoft.AspNet.Mvc`  
`install-package EntityFramework`
- Select the default project as "RepositoryTesting ". Type the following command in "Package Manager Console" and press Enter.  
`install-package EntityFramework`

The screenshot shows the Solution Explorer window with two projects: 'RepositoryTesting' and 'RepositoryTesting.Tests'. The 'RepositoryTesting' project contains files for Connected Services, Properties, References, App\_Data, App\_Start (with RouteConfig.cs), Controllers (HomeController.cs), Models (Employee.cs), RepositoryLayer (EmployeesRepository.cs), Views (Home with Index.cshtml and web.config), Global.asax, packages.config, and Web.config. The 'RepositoryTesting.Tests' project contains files for Properties, References, App.config, HomeControllerTest.cs, and packages.config.

### Creating Employee.cs

- Right click on "Models" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Employee.cs". Click on "Add".

### Code for "Employee.cs"

```
using System;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;

namespace RepositoryTesting.Models
{
    public class Employee
    {
        [Key]
        public int EmpID { get; set; }
        public string EmpName { get; set; }
        public decimal Salary { get; set; }

        public override bool Equals(object obj)
        {
            Employee other = obj as Employee;
            return (this.EmpID == other.EmpID) && (this.EmpName == other.EmpName) && (this.Salary == other.Salary);
        }
    }

    public class CompanyDbContext : DbContext
    {
        public CompanyDbContext() : base(@"data source=localhost\sqlexpress; integrated security=yes; initial catalog=company")
        {
        }

        public DbSet<Employee> Employees { get; set; }
    }
}
```

### Creating EmployeesRepository.cs

- Right click on the project (RepositoryTesting), and click on "Add" - "New Folder". Type the folder name as "RepositoryLayer" and press Enter. Right click on "RepositoryLayer" folder and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "EmployeesRepository.cs". Click on "Add".

### Code for "EmployeesRepository.cs"

```
using System;
using System.Collections.Generic;
using System.Linq;
using RepositoryTesting.Models;

namespace RepositoryTesting.RepositoryLayer
{
    public interface IEmployeesRepository
    {
        List<Employee> GetEmployees();
    }

    public class EmployeesRepository : IEmployeesRepository
    {
        public List<Employee> GetEmployees()
        {
            CompanyDbContext db = new CompanyDbContext();
            List<Employee> emps = db.Employees.ToList();
            return emps;
        }
    }
}
```

### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Mvc;
using RepositoryTesting.RepositoryLayer;
using RepositoryTesting.Models;

namespace RepositoryTesting.Controllers
{
    public class HomeController : Controller
    {
        IEmployeesRepository r;

        public HomeController()
        {
            this.r = new EmployeesRepository();
        }

        public ActionResult Index()
        {
            List<Employee> emps = r.GetEmployees();
            return View(emps);
        }
    }
}
```

### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### Code for "Views\Home\Index.cshtml"

```
<html>
    <head>
        <title>Index</title>
    </head>
    <body>
        <h5>Index</h5>
    </body>
</html>
```

### Creating HomeControllerTest.cs

- Right click on "UnitTest1.cs" file and click on "Rename". Type the filename as "HomeControllerTest.cs" and press Enter.

### Code for "HomeControllerTest.cs"

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using RepositoryTesting.Controllers;
using RepositoryTesting.Models;
using System.Collections.Generic;
using System.Web.Mvc;
using System.Linq;

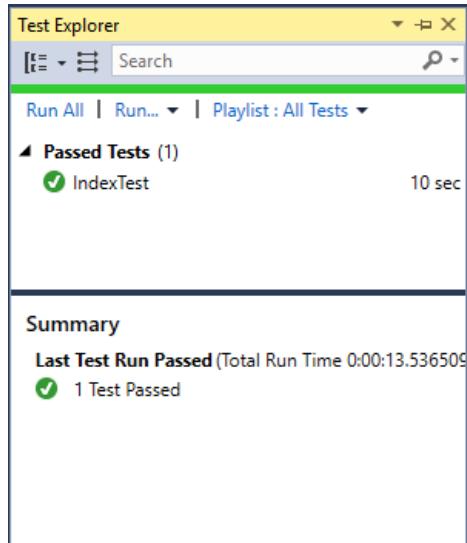
namespace RepositoryTesting.Tests
{
    [TestClass]
    public class HomeControllerTest
    {
        [TestMethod]
    
```

```
public void IndexTest()
{
    HomeController home = new HomeController();
    ViewResult vr = home.Index() as ViewResult;
    List<Employee> actualresult = vr.Model as List<Employee>;
    List<Employee> expectedresult = new List<Employee>()
    {
        new Employee() { EmpID = 1, EmpName = "Scott", Salary = 4000 },
        new Employee() { EmpID = 2, EmpName = "Allen", Salary = 2500 },
        new Employee() { EmpID = 3, EmpName = "Jones", Salary = 5200 },
        new Employee() { EmpID = 4, EmpName = "James", Salary = 4400 },
        new Employee() { EmpID = 5, EmpName = "Smith", Salary = 7600 }
    };
    Assert.IsTrue(expectedresult.SequenceEqual(actualresult));
}
}
```

### Running the application

- Go to "Test" menu - "Run" - "All Tests".

Output:



## DEPLOYMENT

### Deployment to IIS

#### What is Deployment

- Deployment is a process of uploading the project into server.
- After deployment, the user can access the website from a remote system, through browser.
- On the server, no need of installing Visual Studio. It is enough to install IIS (Internet Information Server) only.

### Deployment into IIS - Example

#### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "App1". Type the location as "C:\Mvc". Type the solution name as "App1". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

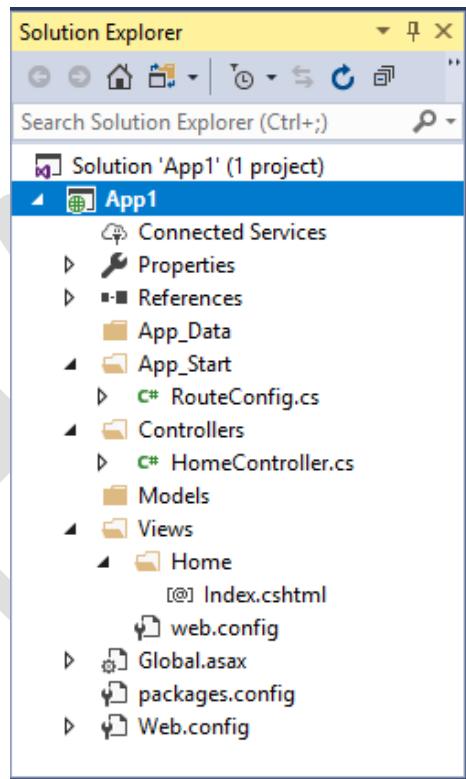
### **Creating HomeController.cs**

- Open Solution Explorer.
- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### **Code for "HomeController.cs"**

```
using System;
using System.Web.Mvc;

namespace App1.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
```



### **Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

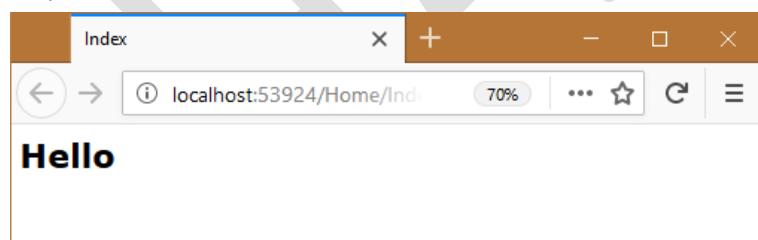
### **Code for "Views\Home\Index.cshtml"**

```
<html>
  <head>
    <title>Index</title>
  </head>
  <body>
    <h1>Hello</h1>
  </body>
</html>
```

### **Running the application**

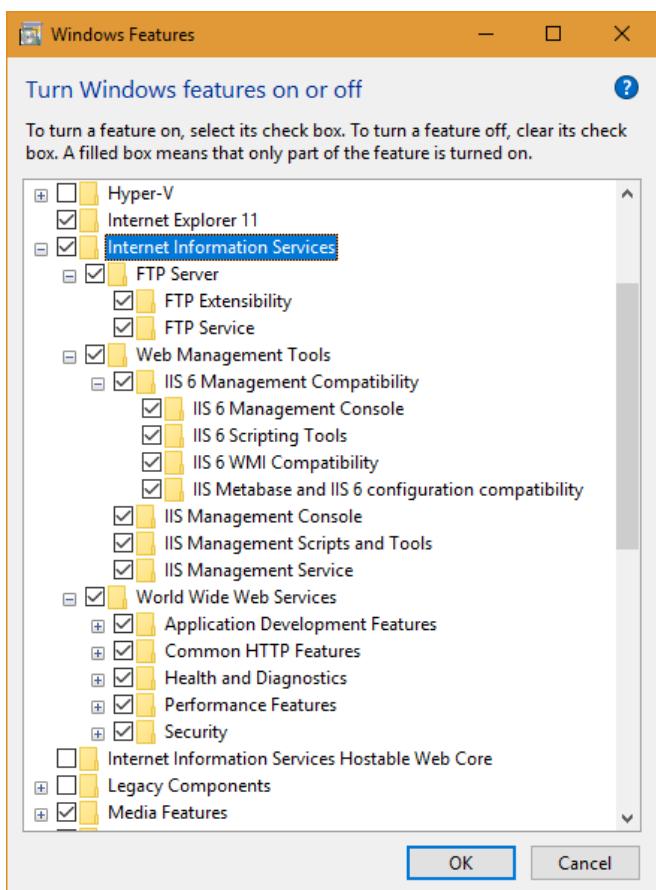
- Press "F5" to run the application.
- Type "http://localhost:portnumber/Home/Index".

Output:



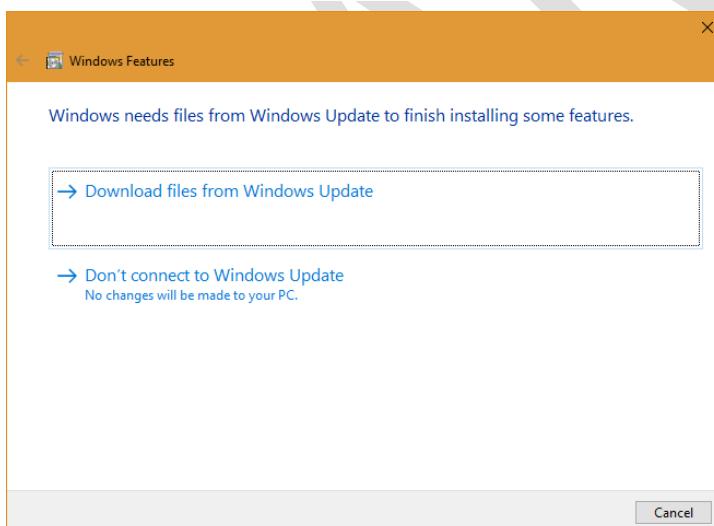
### **Installing IIS**

- Make sure you have internet connection. Go to Control Panel > Programs > Turn Windows features on or off > Check all the checkboxes of "Internet Information Services".

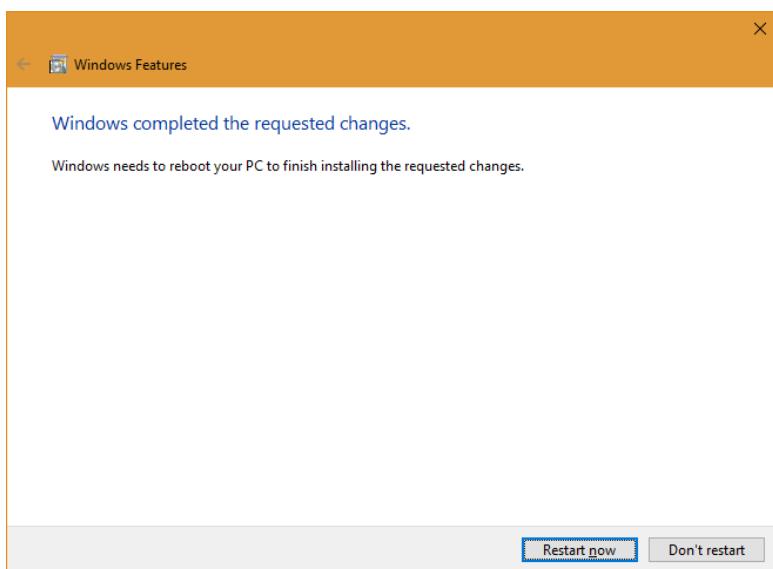


Click on OK.

Click on "Download files from Windows Update".



After finishing the installation, it shows the following screen:

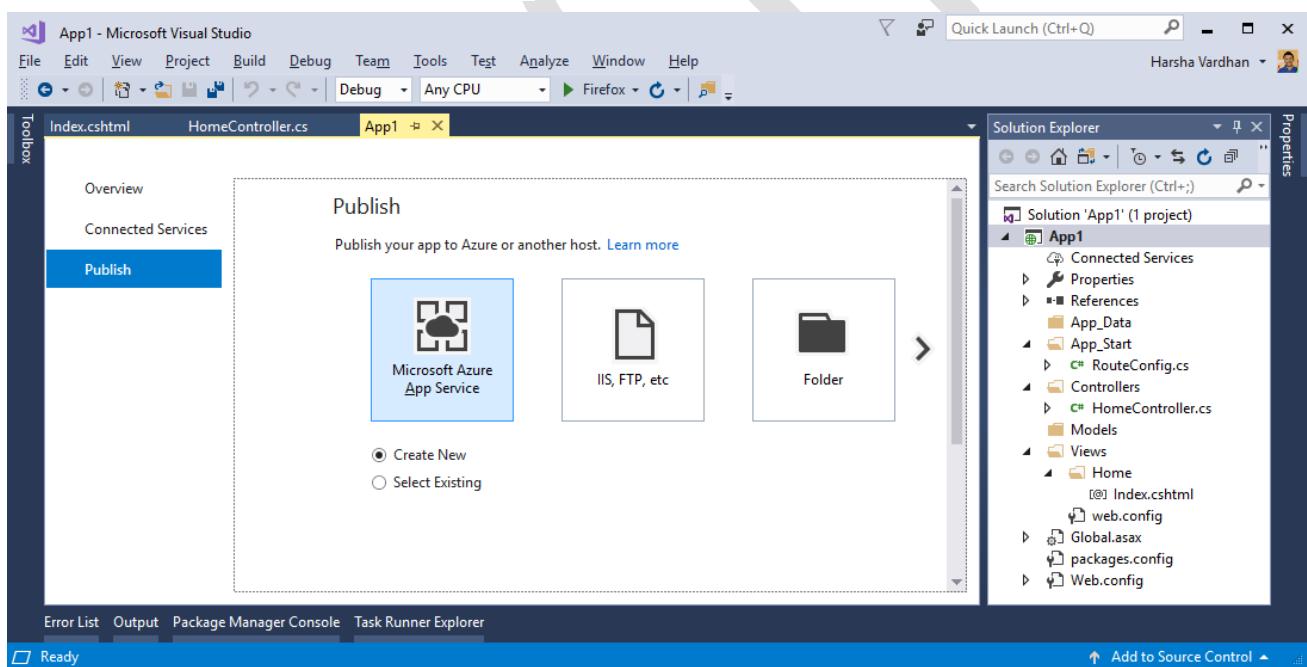


Click on "Restart now".

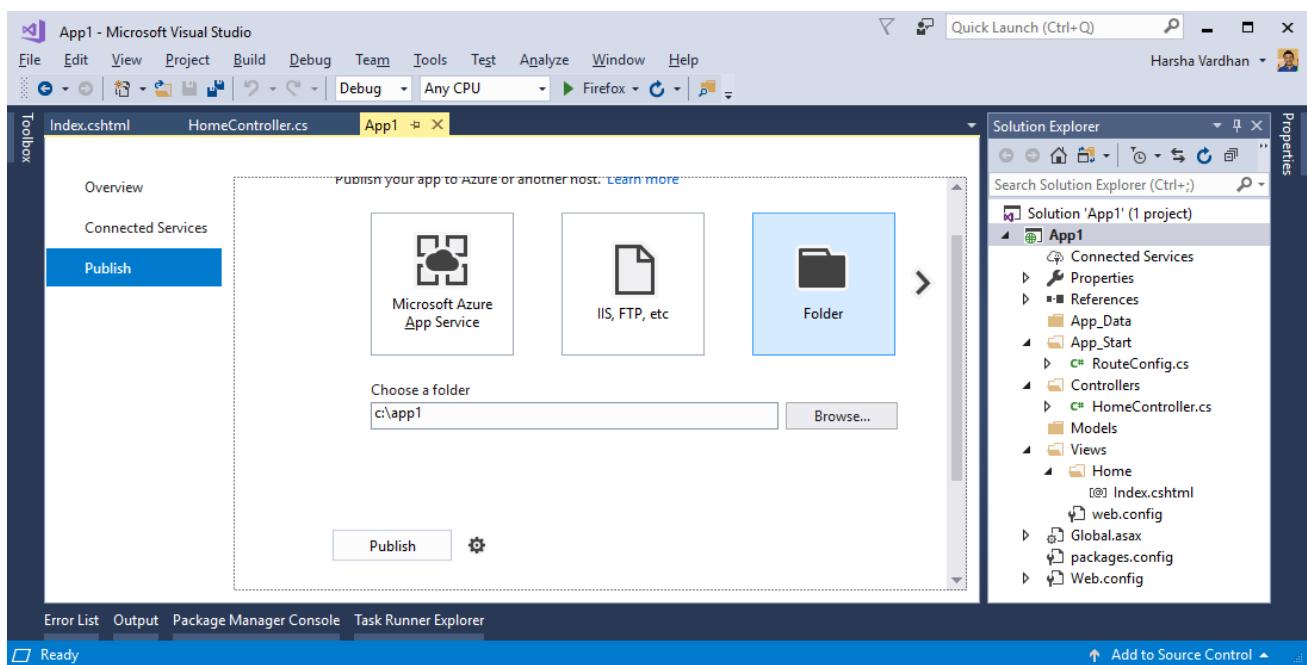
It restarts your system.

## Publishing the application

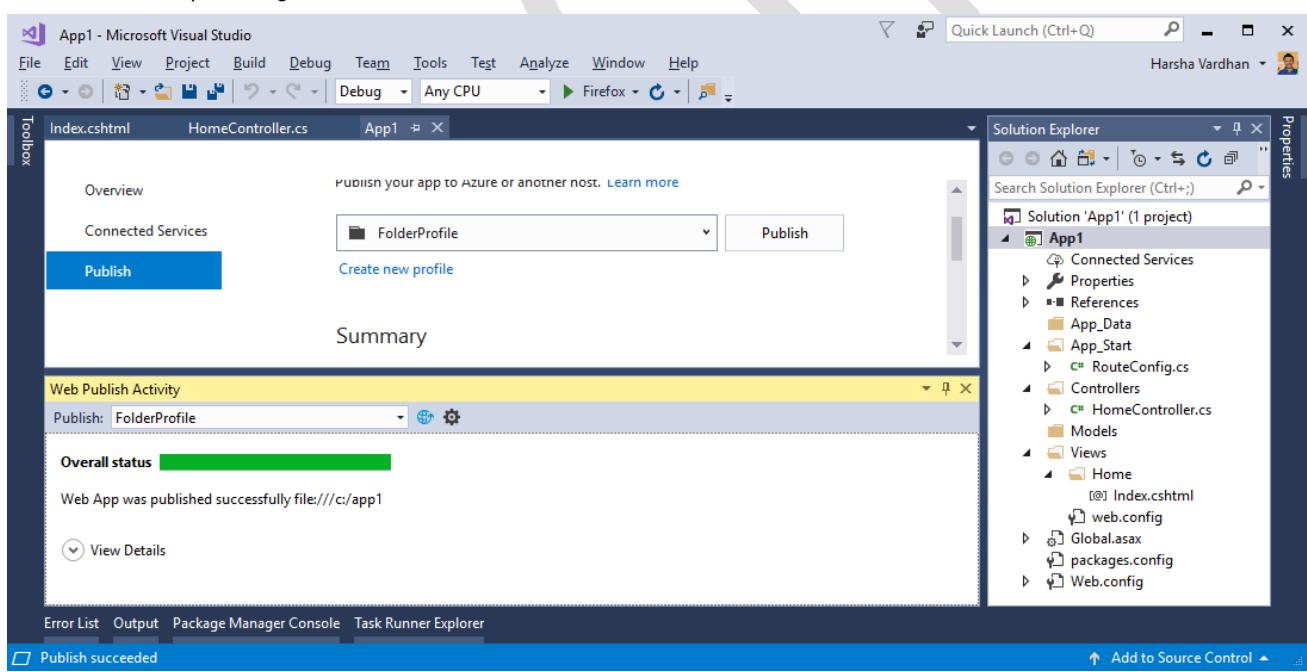
- Right click on the project (App1) and click on "Publish".



- Click on "Folder".
- Enter the folder as "c:\app1".

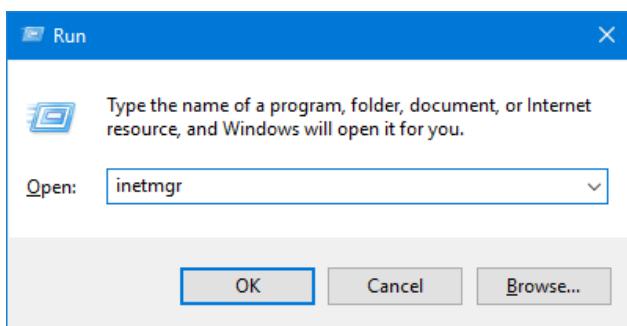


- Click on "Publish".
- After few seconds, publishing will be finished.

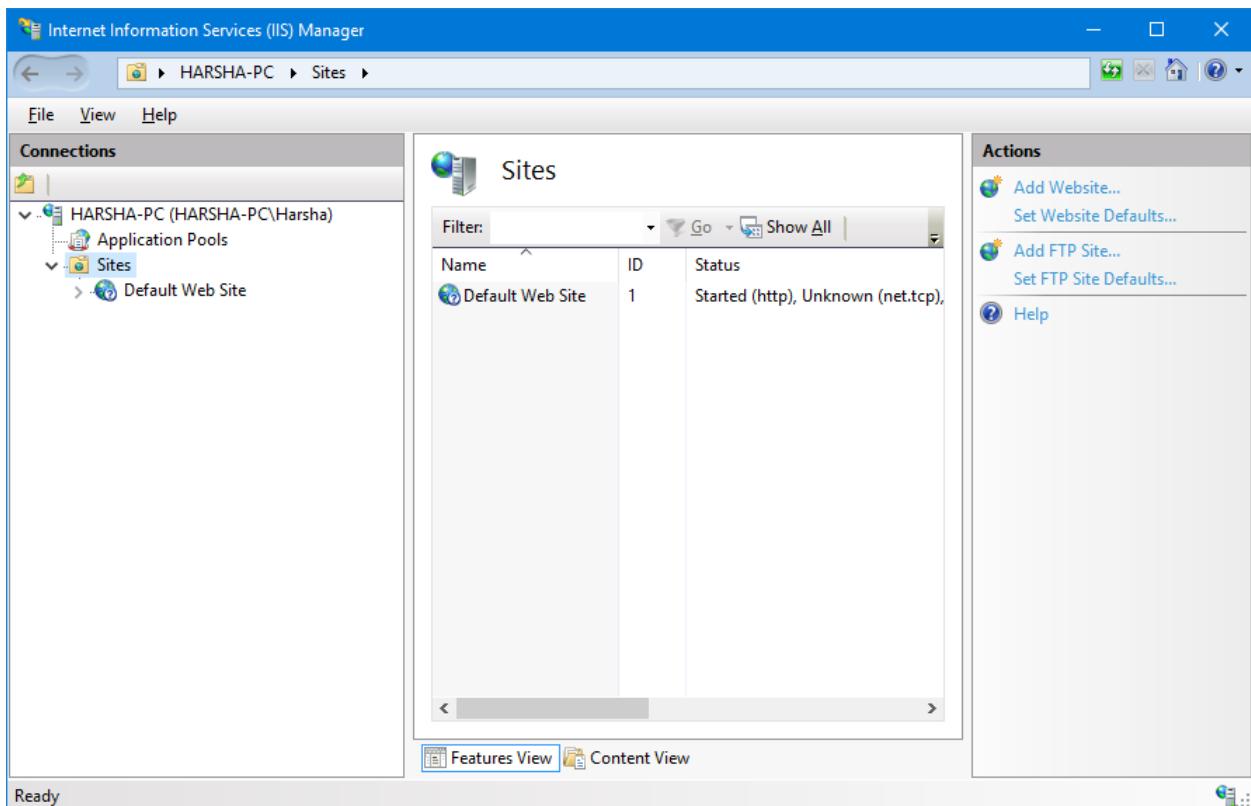


### Deploying the application into IIS

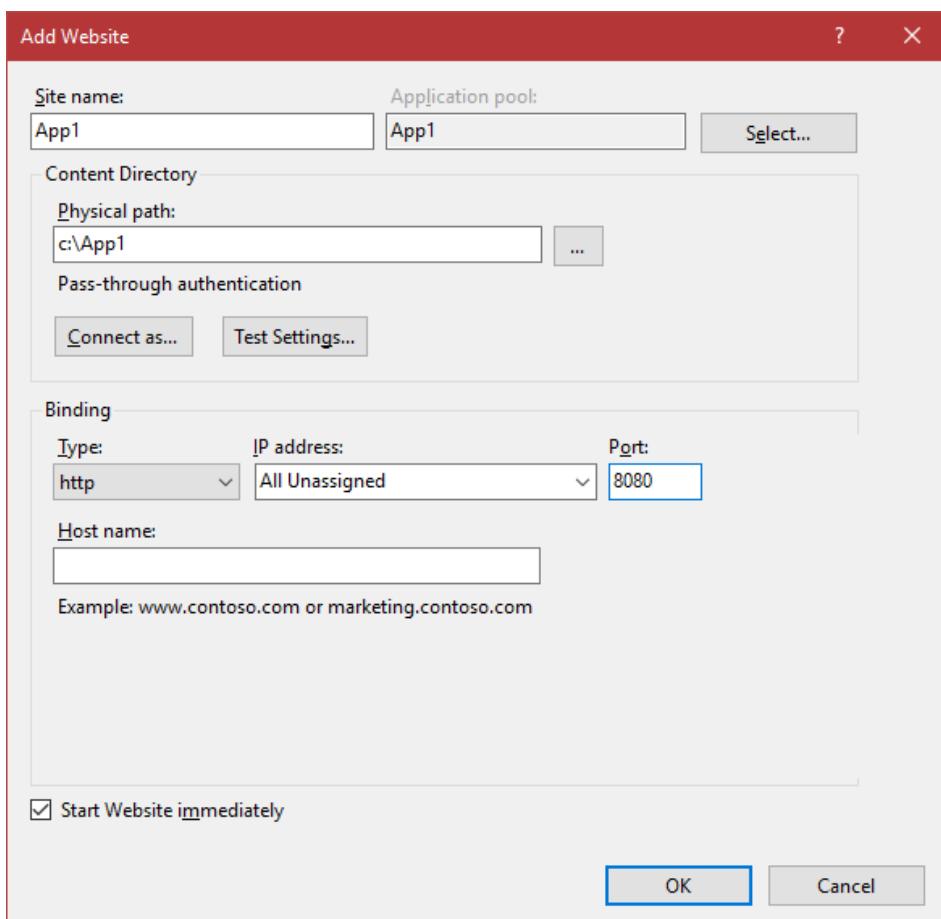
- Press "Windows+R" to open "Run" window.
- Type "inetmgr" and press Enter.



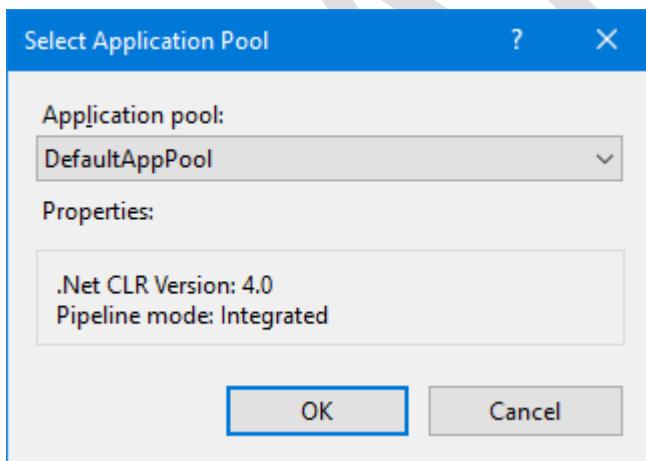
- It shows "Internet Information Services (IIS) Manager" window.



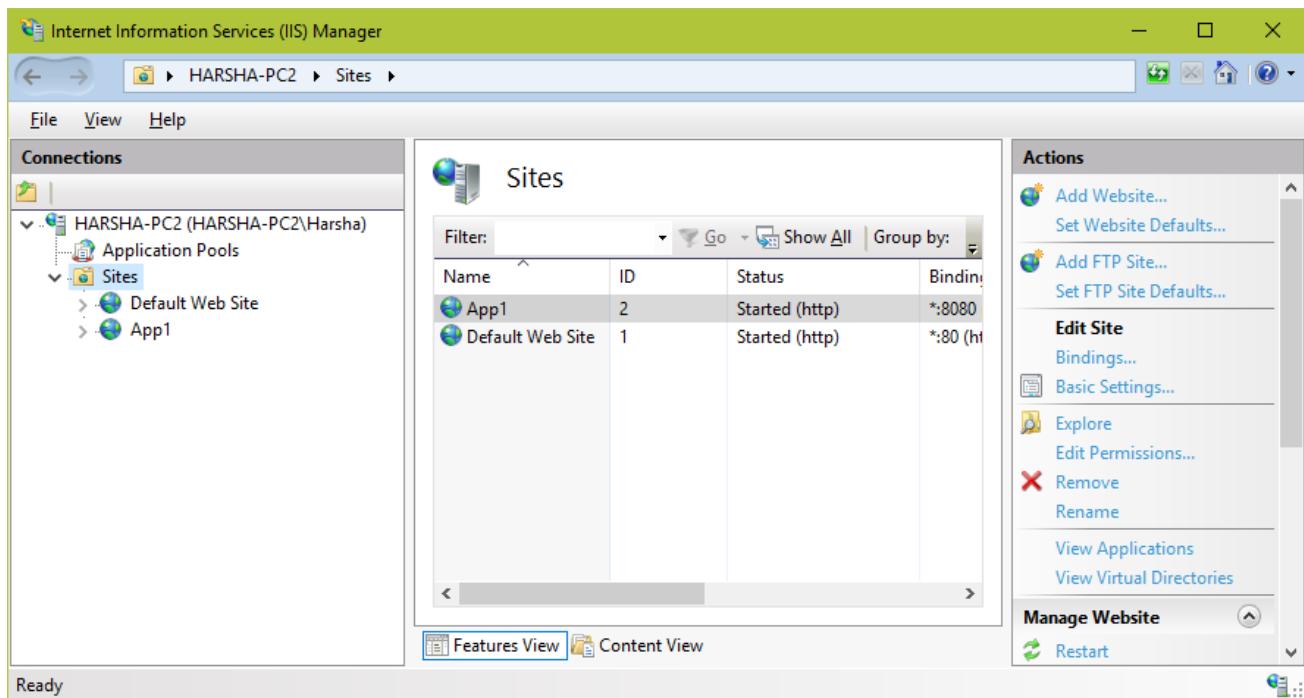
- Right click on "Sites" – "Add Web Site".
- Enter the site name as "App1".
- Select the physical application path as "C:\App1".
- Enter the port no as "8080".



- Click on "Select" button near "Application pool".
- Select "DefaultAppPool". Make sure you have ".NET CLR Version: 4.0", as shown below:



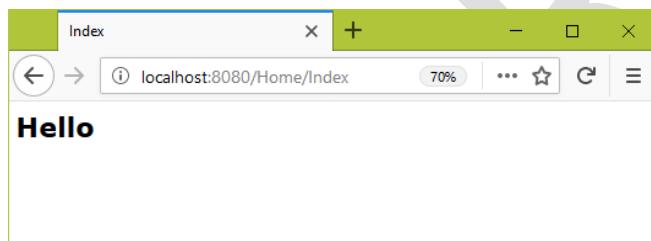
- Click on OK.
- Click on OK again.
- The IIS website (App1) is ready.



### Running the application

- Press "F5" to run the application.
- Type "http://localhost:8080/Home/Index".

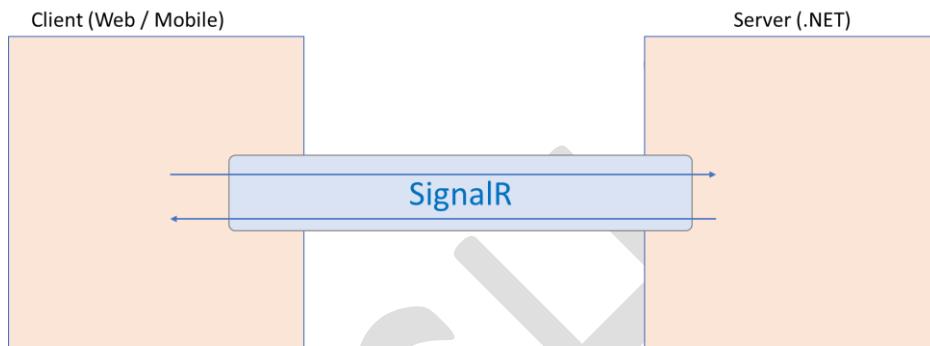
Output:



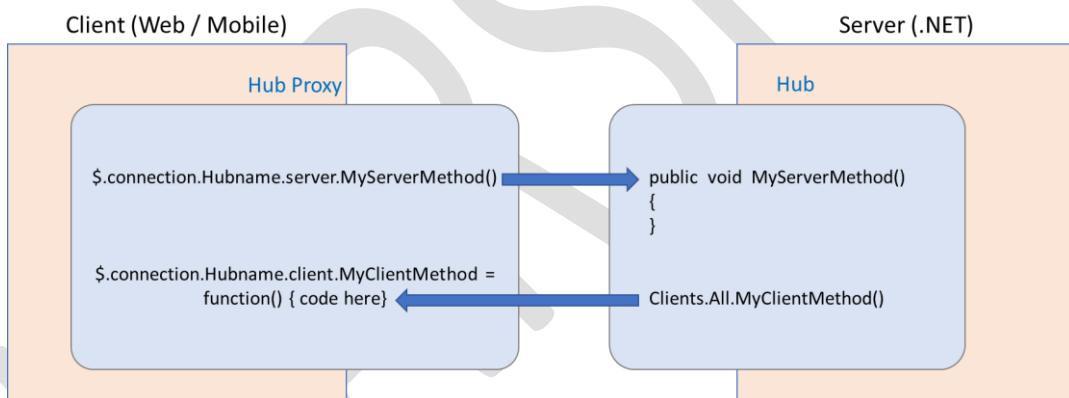
## SIGNALR

### Introduction to SignalR

- SignalR is used to create chatting applications (realtime messenger applications). SignalR supports "server push" or "broadcasting" functionality. That means, server can send data to client, even without needing client send request to server.
- SignalR maintains the open persistent connection between client and server, which acts as a tunnel between client and server, using which they can communicate each other, without needing request / response pattern. SignalR supports RPC (Remote Procedure Call) pattern, using which the server or client can call a method of opposite at any point of time.
- Realtime applications of SignalR:
  - Chatting applications
  - Task monitoring applications



#### Execution Model of SignalR



- "Hub" is a class that inherits from "Microsoft.AspNet.SignalR.Hub" class. The methods of hub class can be called from the client. The server can call client methods using `Clients.All.Methodname()` syntax. The client can call the server's method using `$connection.Hubname.server.Methodname()` syntax.

#### SignalR Underlying Transports

- SignalR is an abstraction layer over some of the transports that supports real-time communication between client and server. A SignalR connection starts as HTTP, and is then promoted to a WebSocket connection if it is supported by the client. If WebSocket is not supported by the client, it uses best possible transport which is supported by the client.
- SignalR mainly supports 3 transports:
  - WebSocket
  - Server Sent Events
  - Forever Frame

#### Web Socket

- It is supported by Web / Mobile Apps. WebSocket is the only transport that supports real duplex communication between client and server. That means client can send data to server; server can send data to client at any point of time, rather than using "Request-Response Model". WebSocket requires the server to be using Windows 8 or later with .NET Framework 4.5; and client to be using latest version of Google Chrome, Firefox or Edge browser. If these requirements are not met, SignalR will automatically attempt to use other transports to make its connections.

### Server Sent Events

- It is supported by all browsers except Internet Explorer. It is also known as "EventSource". Client (browser) sends one request to server and server keep on sending continuous responses to the client, for every fixed interval time (Ex: 5 seconds).

### Forever Frame

- It is supported by Internet Explorer browser. It is hidden IFrame which makes a request to the server that doesn't complete. The server then continually sends script to the client which is immediately executed, providing a one-way realtime connection from server to client. Client needs to create a separate HTTP request connection to the server, for each data that is to be sent to the browser.

### Microsoft.AspNet.SignalR Package

- In order to use SignalR, we have to install Microsoft.AspNet.SignalR package at both server side and client side.
- Installing package:  
install-package Microsoft.AspNet.SignalR

### Map SignalR JavaScript Files At Server

```
using Microsoft.Owin;
using Owin;
using SignalRExample;

namespace SignalRExample
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            app.MapSignalR();
        }
    }
}
```

### Create Hub At Server

```
using System;
using Microsoft.AspNet.SignalR;

namespace SignalRExample
{
    public class MyHub : Hub
    {
    }
}
```

### Client Side API

- To call Server Method from Client, use the following ways:
  - Connect to Server
  - Call Server's Method
  - Define Client Method to be called by Server

### Connect to Server

```
$connection.hub.start().done(
    function()
    {
    }
)
.fail(
    function()
    {
```

```
    }  
);
```

### Call Server's Method

```
$connection.Hubname.ServerMethod();
```

### Define Client Method to be Called by Server

```
$connection.Hubname.client.ClientMethod = function ()  
{  
};
```

### Server Side API

- To call Client Method from Server, use the following ways:
  - Call All Client's Method
  - Call Specific Client's Method
  - Call Current Client's Method

### Call All Client's Method

```
Clients.All.ClientMethod();
```

### Call Specific Client's Method

```
Clients.Client(connectionId).ClientMethod();
```

### Call Current Client's Method

```
Clients.Caller.ClientMethod();
```

### Receiving Return Value of Server Method at Client

```
$connection.Hubname.ServerMethod().done(  
    function(returndata)  
    {  
    }  
)  
.fail(  
    function()  
    {  
    }  
);
```

### Specify HubName Attribute

```
[HubName("hubname here")]  
public class MyHub : Hub  
{  
}
```

### Specify HubMethodName Attribute

```
[HubMethodName("hub method here")]  
public void Methodname()  
{  
}
```

### Enable Logging

```
$connection.hub.logging = true;
```

## SignalR - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "SignalRExample". Type the location as "C:\Mvc". Type the solution name as "SignalRExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package Microsoft.AspNet.SignalR
```

### Creating Startup.cs

- Right click on the project (SignalRExample) and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Startup.cs". Click on "Add".

#### Code for "Startup.cs"

```
using Microsoft.Owin;
using Owin;
using SignalRExample;

namespace SignalRExample
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            app.MapSignalR();
        }
    }
}
```

### Creating MyHub.cs

- Right click on the project (SignalRExample) and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "MyHub.cs". Click on "Add".

#### Code for "MyHub.cs"

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using Microsoft.AspNet.SignalR;

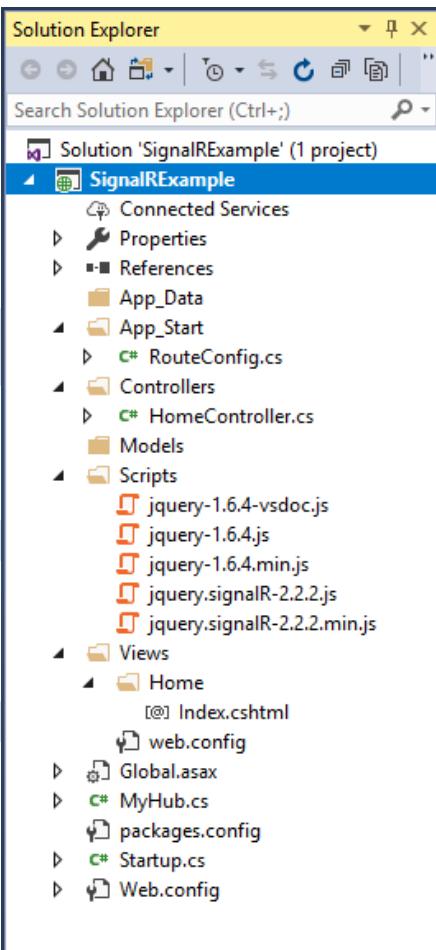
namespace SignalRExample
{
    public class MyHub : Hub
    {
    }
}
```

### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### Code for "HomeController.cs"

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



```

namespace SignalRExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}

```

#### **Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### **Code for "Views\Home\Index.cshtml"**

```

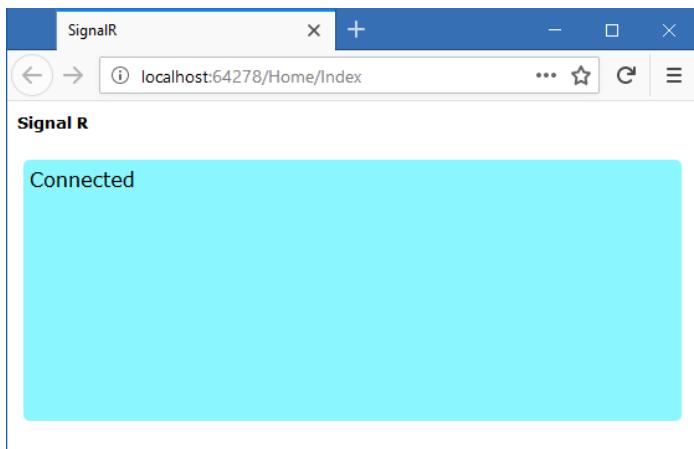
<html>
    <head>
        <title>SignalR</title>
        <style>
            #div1
            {
                background-color: #8af7fe;
                height: 200px;
                border-radius: 5px;
                margin: 5px;
                padding: 5px;
                overflow: auto;
            }
        </style>
    </head>
    <body>
        <h5>Signal R</h5>
        <div id="div1">
        </div>

        <script src="~/Scripts/jquery-1.6.4.js"></script>
        <script src="~/Scripts/jquery.signalR-2.2.2.js"></script>
        <script src="~/signalr/js"></script>
        <script>
            $.connection.hub.start().done(
                function ()
                {
                    document.getElementById("div1").innerHTML += "Connected<br>";
                }
            )
            .fail(
                function ()
                {
                    document.getElementById("div1").innerHTML += "Failed<br>";
                }
            );
        </script>
    </body>
</html>

```

#### **Running the application**

- Go to "Debug" menu and click on "Start Debugging".
  - Type "http://localhost:portnumber/Home/Index".
- Output:



## Method Invocation - Example

**Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name as "SignalRExample". Type the location as "C:\Mvc". Type the solution name as "SignalRExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

**Installing Packages**

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.  
`install-package Microsoft.AspNet.SignalR`

**Creating Startup.cs**

- Right click on the project (SignalRExample) and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Startup.cs". Click on "Add".

**Code for "Startup.cs"**

```
using Microsoft.Owin;
using Owin;
using SignalRExample;

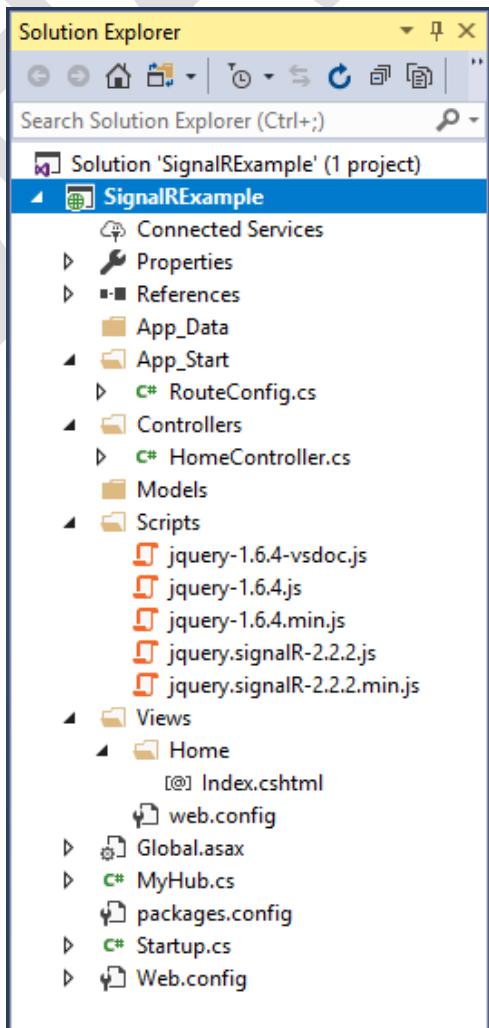
namespace SignalRExample
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            app.MapSignalR();
        }
    }
}
```

**Creating MyHub.cs**

- Right click on the project (SignalRExample) and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "MyHub.cs". Click on "Add".

**Code for "MyHub.cs"**

```
using System;
using Microsoft.AspNet.SignalR;
```



```
public class MyHub : Hub
{
    public void Announce(string message)
    {
        Clients.All.Announce(message);
    }
}
```

**Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

**Code for "HomeController.cs"**

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

namespace SignalRExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

**Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Index.cshtml"**

```
<html>
    <head>
        <title>SignalR</title>
        <style>
            #div1
            {
                background-color: #8af7fe;
                height: 200px;
                border-radius: 5px;
                margin: 5px;
                padding: 5px;
                overflow: auto;
            }
        </style>
    </head>
    <body>
        <h5>Signal R</h5>
        <div id="div1">
        </div>

        <script src="~/Scripts/jquery-1.6.4.js"></script>
        <script src="~/Scripts/jquery.signalR-2.2.2.js"></script>
        <script src="~/signalr/js"></script>
        <script>
            $.connection.hub.start().done(
                function ()
```

```

{
    document.getElementById("div1").innerHTML+="Connected<br>";
    $.connection.myHub.server.announce("Hai How Are You");
}
.fail(
    function ()
{
    document.getElementById("div1").innerHTML+="Failed<br>";
}
);

$.connection.myHub.client.announce=function (message)
{
    document.getElementById("div1").innerHTML+= message + "<br>";
};

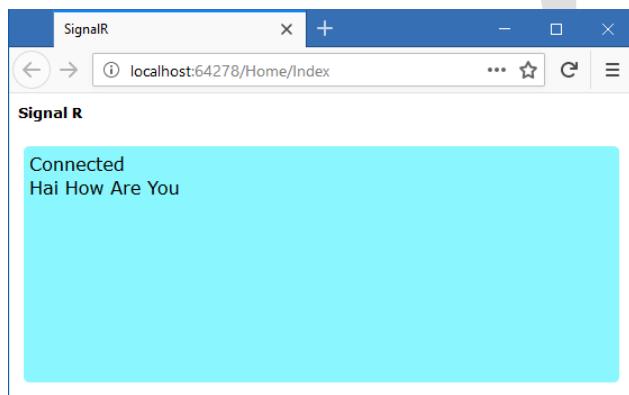
</script>
</body>
</html>

```

**Running the application**

- Go to "Debug" menu and click on "Start Debugging".
- Type "http://localhost:portnumber/Home/Index".

Output:

**Group Chat - Example****Creating Project**

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "SignalRExample". Type the location as "C:\Mvc". Type the solution name as "SignalRExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

**Installing Packages**

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package Microsoft.AspNet.SignalR
```

**Creating Startup.cs**

- Right click on the project (SignalRExample) and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Startup.cs". Click on "Add".

**Code for "Startup.cs"**

```
using Microsoft.Owin;
using Owin;
using SignalRExample;

namespace SignalRExample
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            app.MapSignalR();
        }
    }
}
```

**Creating MyHub.cs**

- Right click on the project (SignalRExample) and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "MyHub.cs". Click on "Add".

**Code for "MyHub.cs"**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using Microsoft.AspNet.SignalR;

namespace SignalRExample
{
    public class MyHub : Hub
    {
        public void Connect(string Username)
        {
            Clients.All.Announce(Username + " Connected");
        }

        public void SendMessage(string From, string Message)
        {
            Clients.All.Announce(From + ":" + Message);
        }
    }
}
```

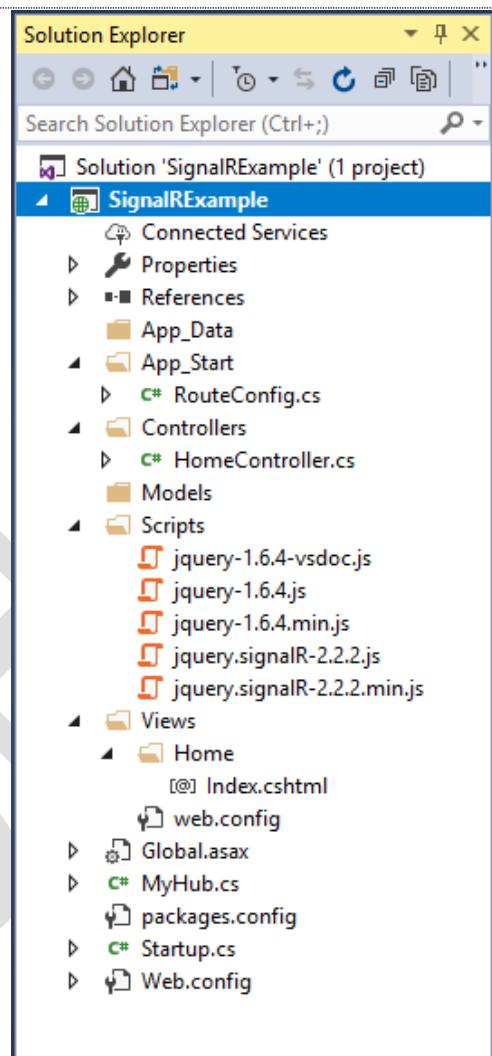
**Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

**Code for "HomeController.cs"**

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

namespace SignalRExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
```



```

    }
}

```

#### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### Code for "Views\Home\Index.cshtml"

```

<html>
  <head>
    <title>SignalR</title>
    <style>
      #div1
      {
        background-color: #8af7fe;
        height: 200px;
        border-radius: 5px;
        margin: 5px;
        padding: 5px;
        overflow: auto;
      }
    </style>
  </head>
  <body>
    <h5>Signal R</h5>
    <div id="connectdiv">
      <input type="text" id="txt1" placeholder="Username" />
      <input type="button" value="Connect" id="button1" />
    </div>
    <div id="div1">
    </div>
    <div id="messagediv" style="display:none">
      <input type="text" id="txt2" placeholder="Message" />
      <input type="button" value="Send" id="button2" />
    </div>

    <script src="~/Scripts/jquery-1.6.4.js"></script>
    <script src="~/Scripts/jquery.signalR-2.2.2.js"></script>
    <script src="~/signalr/js"></script>
    <script>
      $("#button1").click(function ()
      {
        $.connection.hub.start().done(
          function ()
          {
            var username = $("#txt1").val();
            $.connection.myHub.server.connect(username);
            $("#txt1").attr("disabled", "disabled");
            $("#button1").hide();
            $("#messagediv").show();
          }
        )
        .fail(
          function ()
          {
            $("#div1").append("Failed<br>");
          }
        );
      });

      $.connection.myHub.client.announce = function (message)
      {

```

```

    $("#div1").append(message + "<br>");
};

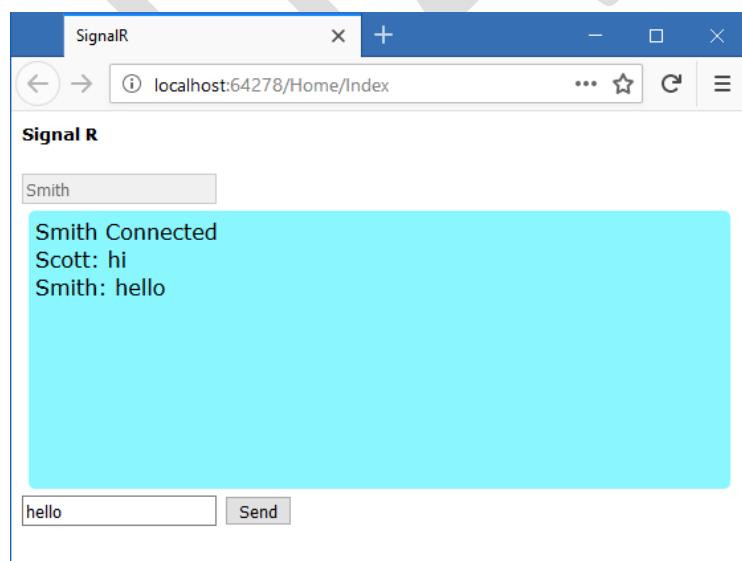
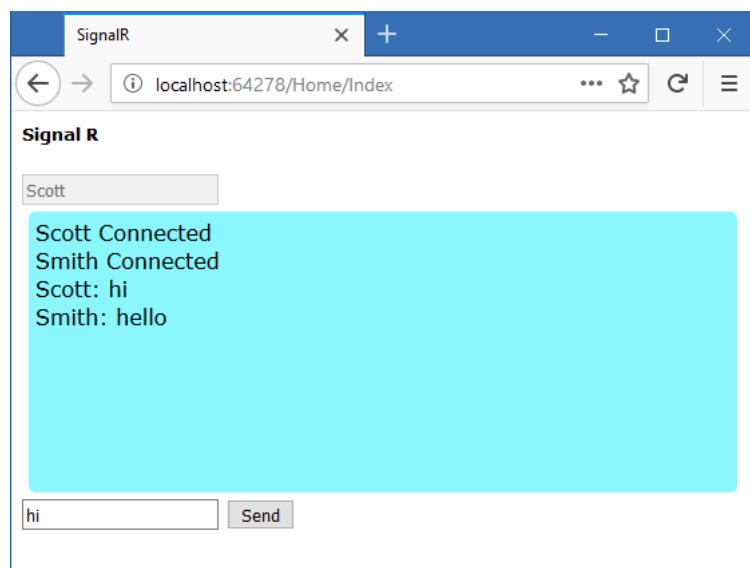
$("#button2").click(function () {
    var from = $("#txt1").val();
    var message = $("#txt2").val();
    $.connection.myHub.server.sendMessage(from, message);
    $("#txt2").val("").focus();
});
</script>
</body>
</html>

```

#### Running the application

- Go to "Debug" menu and click on "Start Debugging".
- Type "http://localhost:portnumber/Home/Index".

Output:



## Private Chat - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name as "SignalRExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package Microsoft.AspNet.SignalR
```

### Creating Startup.cs

- Right click on the project (SignalRExample) and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Startup.cs". Click on "Add".

#### Code for "Startup.cs"

```
using Microsoft.Owin;
using Owin;
using SignalRExample;

namespace SignalRExample
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            app.MapSignalR();
        }
    }
}
```

### Creating MyHub.cs

- Right click on the project (SignalRExample) and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "MyHub.cs". Click on "Add".

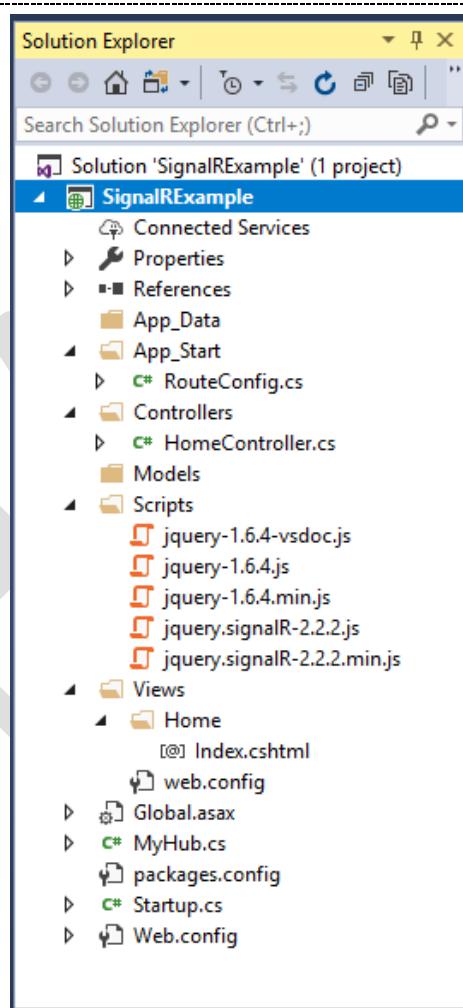
#### Code for "MyHub.cs"

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using Microsoft.AspNet.SignalR;

namespace SignalRExample
{
    public class MyHub : Hub
    {
        public static Dictionary<string, string> AllClients = new Dictionary<string, string>();

        public void Connect(string Username)
        {
            AllClients.Add(Username, Context.ConnectionId);
            Clients.All.Announce(Username + " Connected");
        }

        public void SendMessage(string From, string To, string Message)
        {
            Clients.Client(AllClients[To]).DisplayMessage(From + ":" + Message);
            Clients.Client(AllClients[From]).DisplayMessage(From + ":" + Message);
        }
    }
}
```



```

    }
}
}

```

#### **Creating HomeController.cs**

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### **Code for "HomeController.cs"**

```

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

namespace SignalRExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}

```

#### **Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

#### **Code for "Views\Home\Index.cshtml"**

```

<html>
<head>
    <title>SignalR</title>
    <style>
        #div1
        {
            background-color: #8af7fe;
            height: 200px;
            border-radius: 5px;
            margin: 5px;
            padding: 5px;
            overflow: auto;
        }
    </style>
</head>
<body>
    <h5>Signal R</h5>
    <div id="connectdiv">
        <input type="text" id="txt1" placeholder="Username" />
        <input type="button" value="Connect" id="button1" />
    </div>
    <div id="div1">
    </div>
    <div id="messagediv" style="display:none">
        <input type="text" id="txt2" placeholder="To" />
        <input type="text" id="txt3" placeholder="Message" />
        <input type="button" value="Send" id="button2" />
    </div>
<script src="~/Scripts/jquery-1.6.4.js"></script>
<script src="~/Scripts/jquery.signalR-2.2.2.js"></script>

```

```

<script src="~/signalr/js"></script>
<script>
$("#button1").click(function ()
{
    $.connection.hub.start().done(
        function ()
        {
            var username = $("#txt1").val();
            $.connection.myHub.server.connect(username);
            $("#txt1").attr("disabled", "disabled");
            $("#button1").hide();
            $("#messagediv").show();
        }
    )
    .fail(
        function ()
    {
        $("#div1").append("Failed<br>");
    }
);
});

$.connection.myHub.client.announce = function (message)
{
    $("#div1").append(message + "<br>");
};

$("#button2").click(function ()
{
    var from = $("#txt1").val();
    var to = $("#txt2").val();
    var message = $("#txt3").val();
    $.connection.myHub.server.sendMessage(from, to, message);
    $("#txt3").val("").focus();
});
};

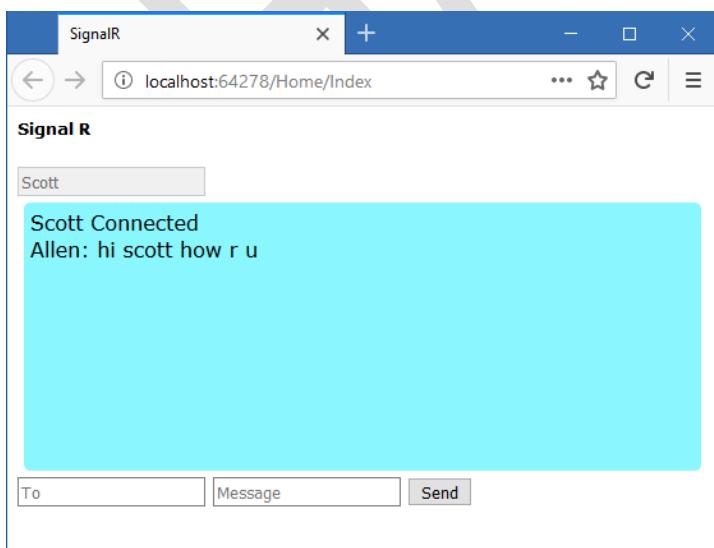
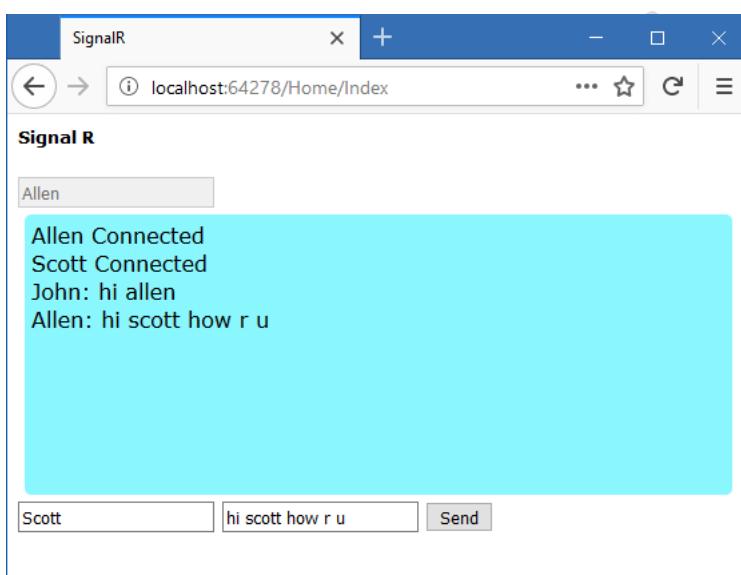
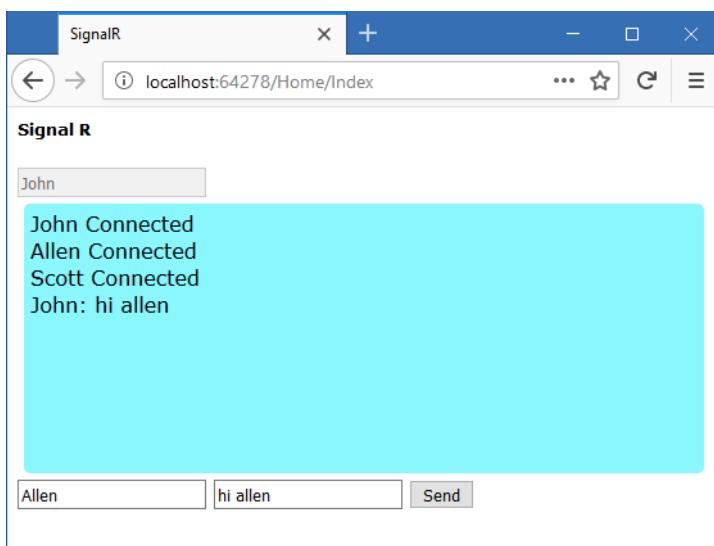
$.connection.myHub.client.displayMessage = function (message)
{
    $("#div1").append(message + "<br>");
};
</script>
</body>
</html>

```

#### Running the application

- Go to "Debug" menu and click on "Start Debugging".
- Type "http://localhost:portnumber/Home/Index".

Output:



## Method Returns - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name as "SignalRExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened. Type the following command in "Package Manager Console" and press Enter.

```
install-package Microsoft.AspNet.SignalR
```

### Creating Startup.cs

- Right click on the project (SignalRExample) and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Startup.cs". Click on "Add".

#### Code for "Startup.cs"

```
using Microsoft.Owin;
using Owin;
using SignalRExample;

namespace SignalRExample
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            app.MapSignalR();
        }
    }
}
```

### Creating MyHub.cs

- Right click on the project (SignalRExample) and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "MyHub.cs". Click on "Add".

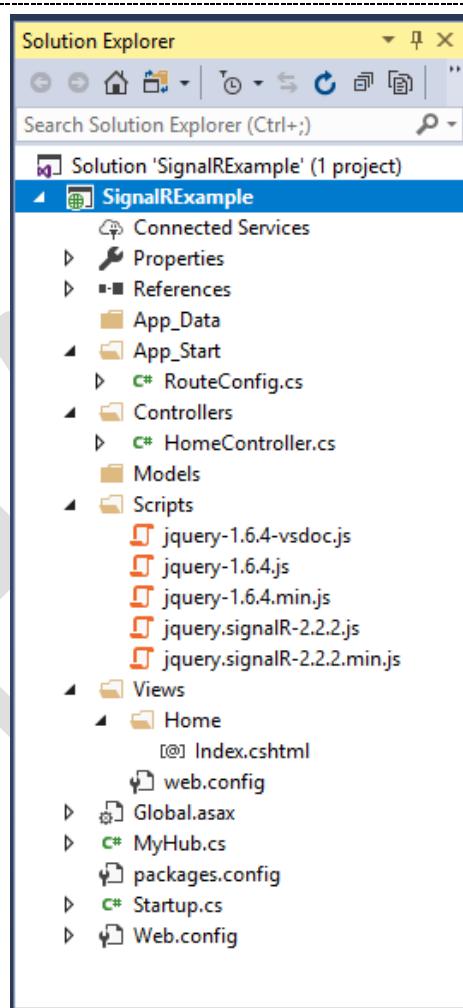
#### Code for "MyHub.cs"

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using Microsoft.AspNet.SignalR;

namespace SignalRExample
{
    public class MyHub : Hub
    {
        public string SendMessage(string message)
        {
            Clients.All.Announce(message);
            return "Message Received";
        }
    }
}
```

### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".



**Code for "HomeController.cs"**

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

namespace SignalRExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

**Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Index.cshtml"**

```
<html>
<head>
<title>SignalR</title>
<style>
#div1
{
    background-color: #8af7fe;
    height: 200px;
    border-radius: 5px;
    margin: 5px;
    padding: 5px;
    overflow: auto;
}
</style>
</head>
<body>
<h5>Signal R</h5>
<div id="div1">
</div>

<script src="~/Scripts/jquery-1.6.4.js"></script>
<script src="~/Scripts/jquery.signalR-2.2.2.js"></script>
<script src="~/signalr/js"></script>
<script>
$.connection.hub.start().done(
    function ()
    {
        document.getElementById("div1").innerHTML+= "Connected<br>";
        $.connection.myHub.server.sendMessage("Hai How Are You").done(
            function (data)
            {
                document.getElementById("div1").innerHTML+= data + "<br>";
            }
        )
        .error(
            function (error)
            {
                document.getElementById("div1").innerHTML+= error + "<br>";
            }
        );
    }
);</script>
```

```
        }
    );
}
.fail(
    function (error)
{
    document.getElementById("div1").innerHTML+=error + "<br>";
}
);

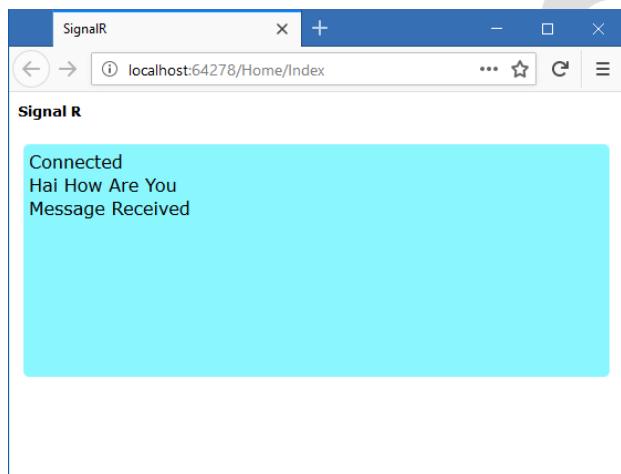
$.connection.myHub.client.announce=function (message)
{
    document.getElementById("div1").innerHTML+=message + "<br>";
};

</script>
</body>
</html>
```

### Running the application

- Go to "Debug" menu and click on "Start Debugging".
- Type "http://localhost:portnumber/Home/Index".

Output:



## HubName and HubMethodName - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name "SignalRExample". Type the location as "C:\Mvc". Type the solution name as "SignalRExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console". "Package Manager Console" window will be opened.
- Type the following command in "Package Manager Console" and press Enter.  
`install-package Microsoft.AspNet.SignalR`

### Creating Startup.cs

- Right click on the project (SignalRExample) and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Startup.cs". Click on "Add".

### Code for "Startup.cs"

```
using Microsoft.Owin;
```

```

using Owin;
using SignalRExample;

namespace SignalRExample
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            app.MapSignalR();
        }
    }
}

```

#### Creating MyHub.cs

- Right click on the project (SignalRExample) and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "MyHub.cs". Click on "Add".

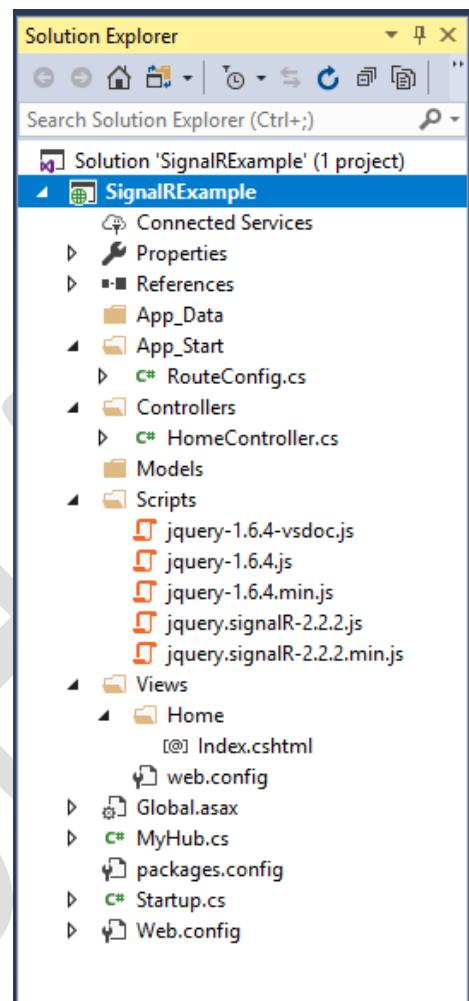
#### Code for "MyHub.cs"

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using Microsoft.AspNet.SignalR;
using Microsoft.AspNet.SignalR.Hubs;

namespace SignalRExample
{
    [HubName("Chat")]
    public class MyHub : Hub
    {
        [HubMethodName("Promote")]
        public void Announce(string message)
        {
            Clients.All.Announce(message);
        }
    }
}

```



#### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

#### Code for "HomeController.cs"

```

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

namespace SignalRExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}

```

**Creating Index.cshtml**

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

**Code for "Views\Home\Index.cshtml"**

```

<html>
  <head>
    <title>SignalR</title>
    <style>
      #div1
      {
        background-color: #8af7fe;
        height: 200px;
        border-radius: 5px;
        margin: 5px;
        padding: 5px;
        overflow: auto;
      }
    </style>
  </head>
  <body>
    <h5>Signal R</h5>
    <div id="div1">
    </div>

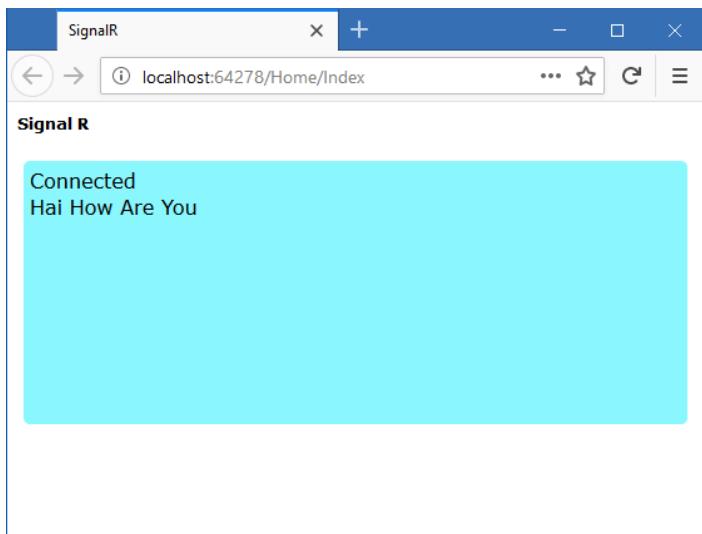
    <script src="~/Scripts/jquery-1.6.4.js"></script>
    <script src="~/Scripts/jquery.signalR-2.2.2.js"></script>
    <script src="~/signalr/js"></script>
    <script>
      $.connection.hub.start().done(
        function ()
        {
          document.getElementById("div1").innerHTML+= "Connected<br>";
          $.connection.Chat.server.Promote("Hai How Are You");
        }
      )
      .fail(
        function ()
        {
          document.getElementById("div1").innerHTML+= "Failed<br>";
        }
      );
    <$.connection.Chat.client.announce = function (message)>
    <{
      document.getElementById("div1").innerHTML+= message + "<br>";
    }>
    </script>
  </body>
</html>

```

**Running the application**

- Go to "Debug" menu and click on "Start Debugging".
- Type "http://localhost:portnumber/Home/Index".

Output:



## Logging - Example

### Creating Project

- Open Visual Studio 2017. Go to "File" - "New" - "Project". Select ".NET Framework 4.7". Select "Visual C#". Select "ASP.NET Web Application (.NET Framework)". Type the project name as "SignalRExample". Type the location as "C:\Mvc". Type the solution name as "SignalRExample". Click on OK. Select "Empty". Check the checkbox "MVC". Uncheck the checkbox "Enable Docker support". Uncheck the checkbox "Add unit tests". Click on OK.

### Installing Packages

- Go to "Tools" menu - "NuGet Package Manager" - "Package Manager Console".
- "Package Manager Console" window will be opened.
- Type the following command in "Package Manager Console" and press Enter.  
`install-package Microsoft.AspNet.SignalR`

### Creating Startup.cs

- Right click on the project (SignalRExample) and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "Startup.cs". Click on "Add".

#### Code for "Startup.cs"

```
using Microsoft.Owin;
using Owin;
using SignalRExample;

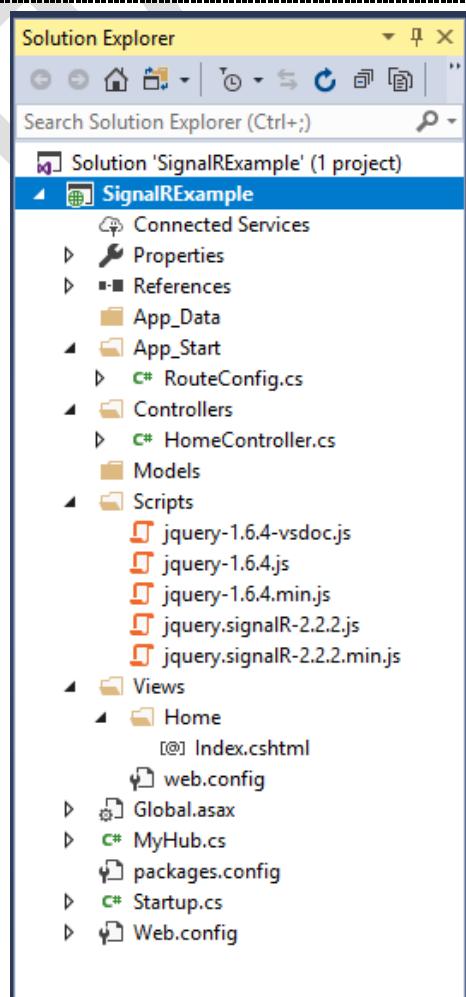
namespace SignalRExample
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            app.MapSignalR();
        }
    }
}
```

### Creating MyHub.cs

- Right click on the project (SignalRExample) and click on "Add" - "New Item". Click on "Code" - "Class". Type the file name as "MyHub.cs". Click on "Add".

#### Code for "MyHub.cs"

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



```
using System.Linq;
using System.Web;
using Microsoft.AspNet.SignalR;

namespace SignalRExample
{
    public class MyHub : Hub
    {
        public void Announce(string message)
        {
            Clients.All.Announce(message);
        }
    }
}
```

### Creating HomeController.cs

- Right click on "Controllers" folder and click on "Add" - "Controller". Select "MVC 5 Controller - Empty". Click on "Add". Type the controller name as "HomeController". Click on "Add".

### Code for "HomeController.cs"

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

namespace SignalRExample.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

### Creating Index.cshtml

- Right click on "Views\Home" folder and click on "Add" - "View". Type the view name as "Index". Select the template "Empty (without model)". Uncheck all the checkboxes. Click on "Add".

### Code for "Views\Home\Index.cshtml"

```
<html>
<head>
    <title>SignalR</title>
    <style>
        #div1
        {
            background-color: #8af7fe;
            height: 200px;
            border-radius: 5px;
            margin: 5px;
            padding: 5px;
            overflow: auto;
        }
    </style>
</head>
<body>
    <h5>Signal R</h5>
    <div id="div1">
    </div>

```

```

<script src="~/Scripts/jquery-1.6.4.js"></script>
<script src="~/Scripts/jquery.signalR-2.2.2.js"></script>
<script src="~/signalr/js"></script>
<script>
    $.connection.hub.start().done(
        function ()
    {
        $.connection.hub.logging=true;
        document.getElementById("div1").innerHTML+="Connected<br>";
        $.connection.myHub.server.announce("Hai How Are You");
    }
)
.fail(
    function ()
{
    document.getElementById("div1").innerHTML+="Failed<br>";
}
);
$.connection.myHub.client.announce=function (message)
{
    document.getElementById("div1").innerHTML+=message + "<br>";
};
</script>
</body>
</html>

```

#### Running the application

- Go to "Debug" menu and click on "Start Debugging".
- Type "http://localhost:portnumber/Home/Index".

#### Output:

