

# Introduction

As you know, AngularJS is the most popular JavaScript framework to develop single page applications. You can also use Angular for **Insert**, **Update**, **Delete** and **Retrieve** operations. This tip will demonstrate how to use Angular with MVC5 and WebAPI2 for CRUD (**Create**, **Read**, **Update**, **Delete**) operations. Many experienced developers will find this tip very basic, but since the tip is written from the perspective of beginners, I've tried to keep things simple.


## Background

In my previous [article](#), I demonstrated CRUD operations with KnockoutJS. Here, I will be using the same database, design and contents, but instead of Knockout, I will be using AngularJS and operations will be handled in WebAPI.

## Using the Code

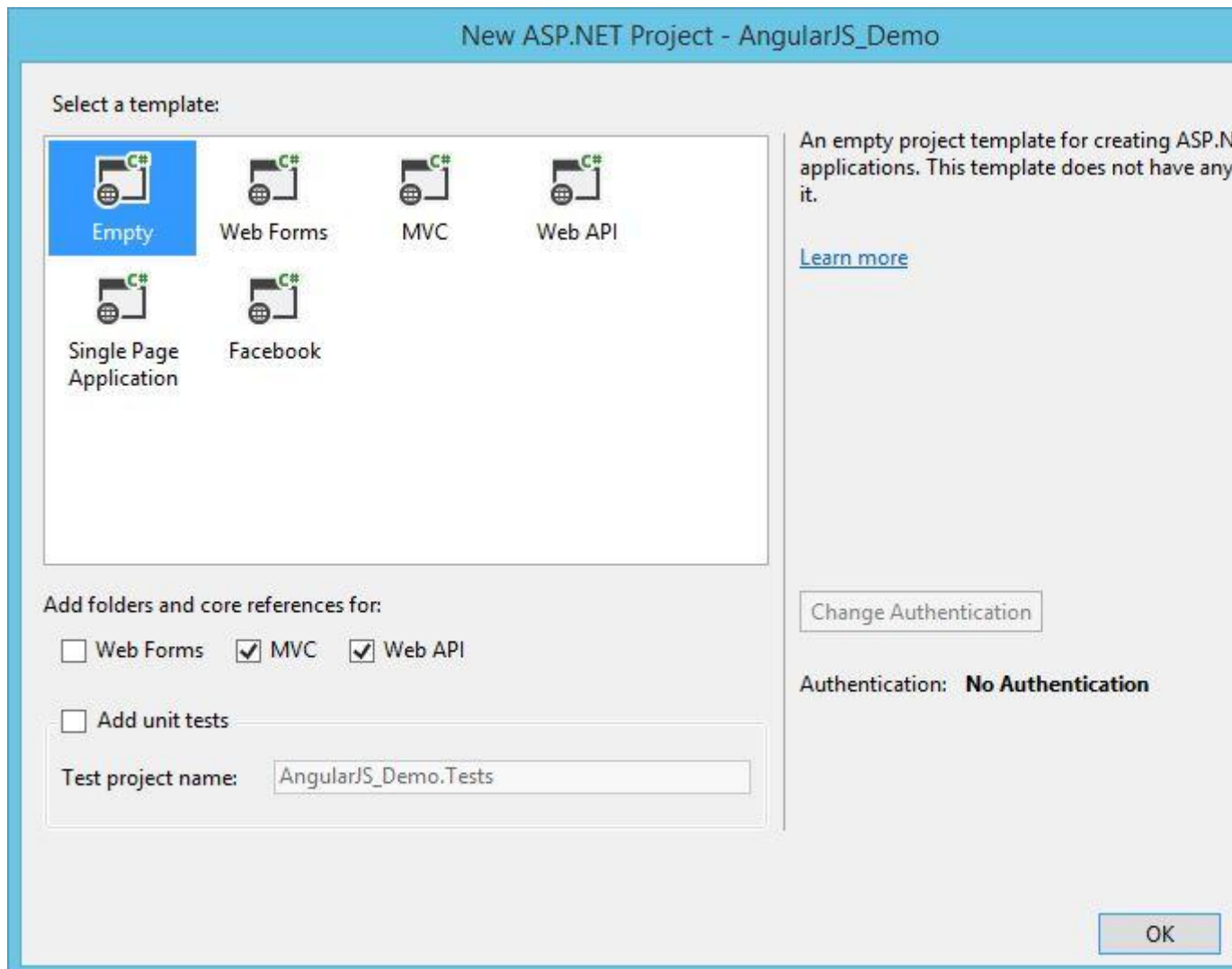
Let's begin !!!

Create '**TblProductList**' table with this schema.

Column Name	Data Type	Allow Nulls
 Id	int	<input type="checkbox"/>
Name	varchar(20)	<input checked="" type="checkbox"/>
Category	varchar(20)	<input checked="" type="checkbox"/>
Price	decimal(10, 2)	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

Create a new project in ASP.NET MVC 5 and name it as you prefer and select empty project template.

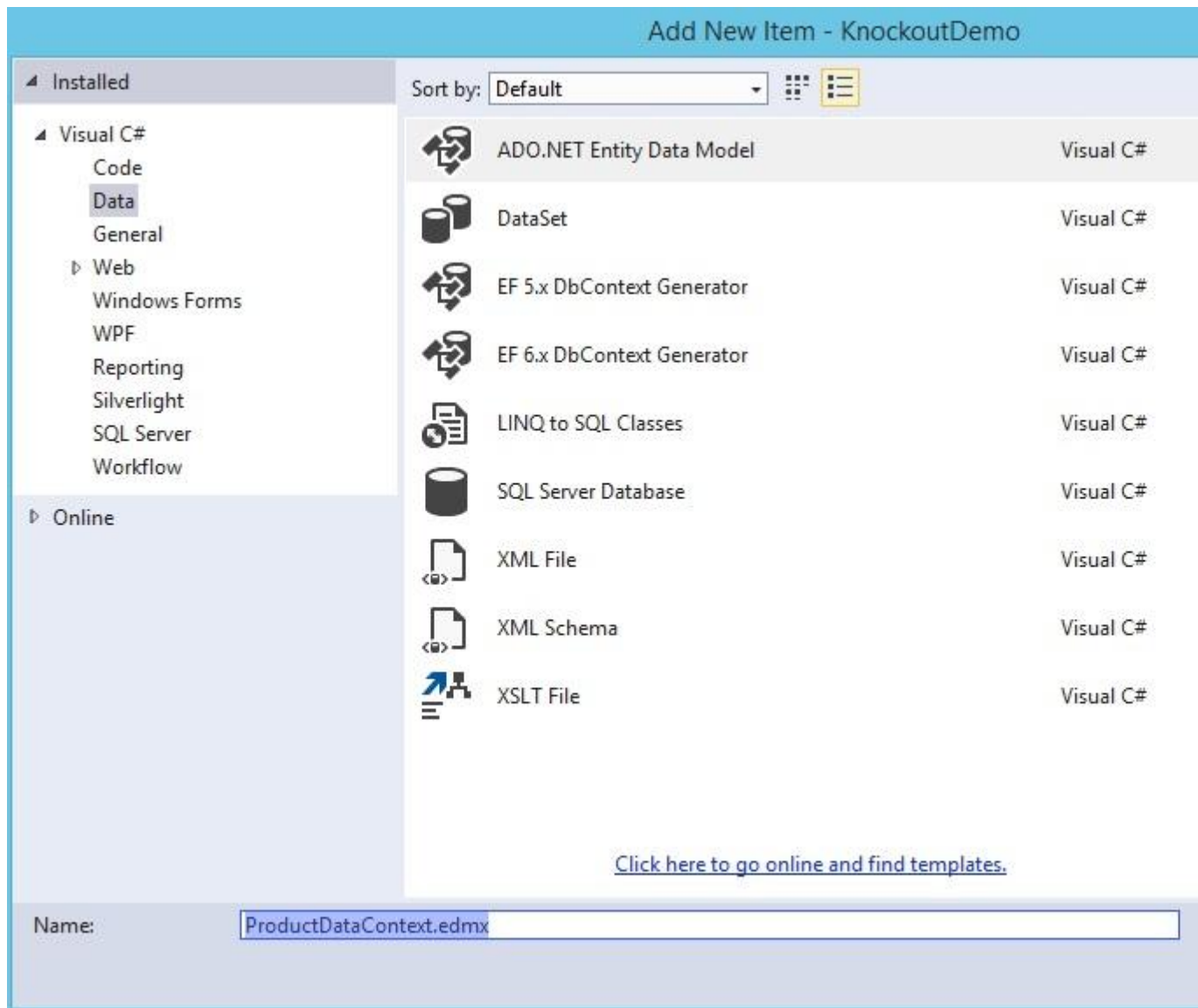
Tick MVC and Web API under *Add* folders and core references for:



Install Entity Framework 6, JQuery and AngularJS in your project using NuGet Package Manager.

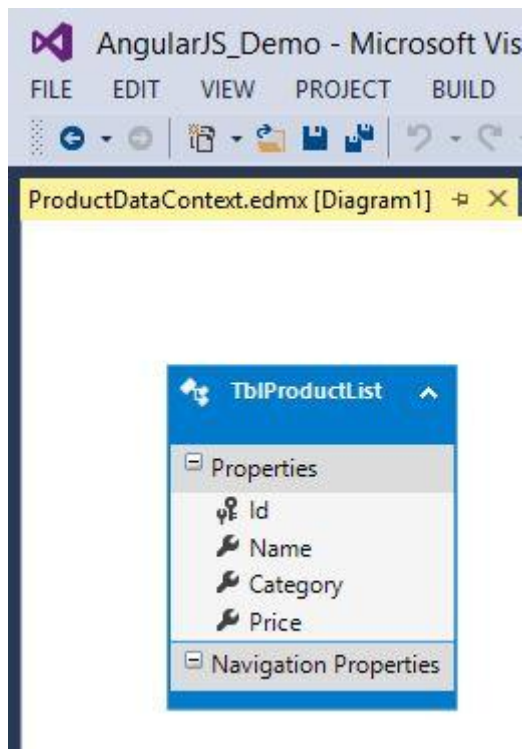
You can also download jquery.js and angular.js from their official website and paste it in 'Scripts' folder of your project.

Right click on *Model* folder and add a new ADO.NET Entity Data Model. Name it as '*ProductDataContext.edmx*'.



Choose 'Generate from Database' and configure the connection string as per your SQL server.

After generating the model, you will get the entity of **TblProductList**.



Create new folder '*Interface*' in root directory. Add a new class '*IProductRepository.cs*'.

[Hide](#) [Copy Code](#)

```
interface IProductRepository
{
    IEnumerable<TblProductList> GetAll();
    TblProductList Get(int id);
    TblProductList Add(TblProductList item);
    bool Update(TblProductList item);
    bool Delete(int id);
}
```

Create new folder '*Repositories*' in root directory. Add a new class '*ProductRepository.cs*'. Implement the methods to **Create, Read, Update, Delete** using Entity Framework.

[Hide](#) [Shrink](#) [Copy Code](#)

```
public class ProductRepository : IProductRepository
{
    ProductDBEntities ProductDB = new ProductDBEntities();

    public IEnumerable<TblProductList> GetAll()
    {
        // TO DO : Code to get the list of all the records in database
        return ProductDB.TblProductLists;
    }

    public TblProductList Get(int id)
    {
        // TO DO : Code to find a record in database
        return ProductDB.TblProductLists.Find(id);
    }

    public TblProductList Add(TblProductList item)
    {

```

```

        if (item == null)
        {
            throw new ArgumentNullException("item");
        }

        // TO DO : Code to save record into database
        ProductDB.TblProductLists.Add(item);
        ProductDB.SaveChanges();
        return item;
    }

    public bool Update(TblProductList item)
    {
        if (item == null)
        {
            throw new ArgumentNullException("item");
        }

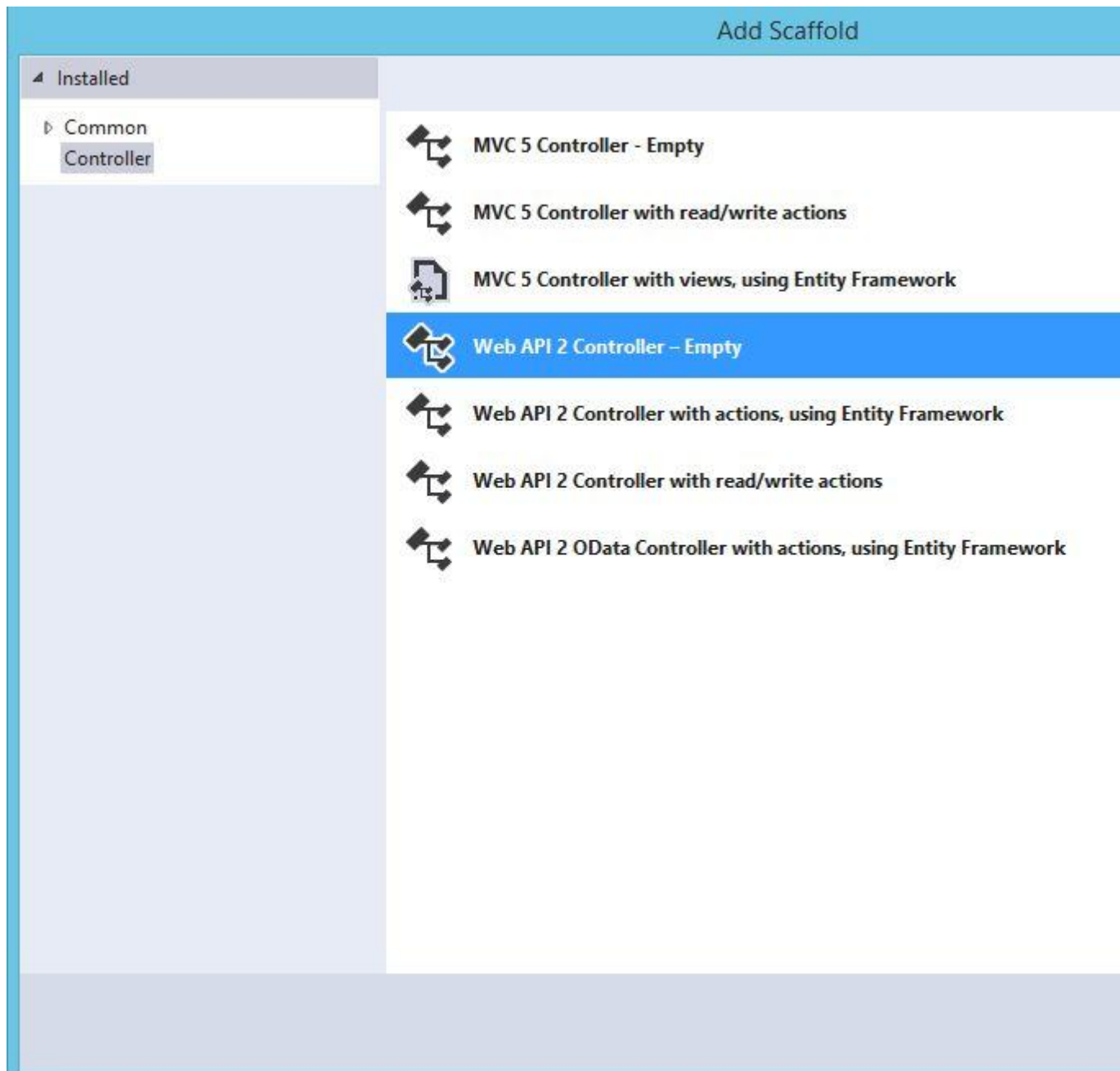
        // TO DO : Code to update record into database
        var products = ProductDB.TblProductLists.Single(a => a.Id == item.Id);
        products.Name = item.Name;
        products.Category = item.Category;
        products.Price = item.Price;
        ProductDB.SaveChanges();

        return true;
    }

    public bool Delete(int id)
    {
        // TO DO : Code to remove the records from database
        TblProductList products = ProductDB.TblProductLists.Find(id);
        ProductDB.TblProductLists.Remove(products);
        ProductDB.SaveChanges();
        return true;
    }
}

```

Right click on *Controllers* folder and add new WebAPI 2 Empty Controller  
'ProductController.cs':



Hide Shrink ▲ Copy Code

```
public class ProductController : ApiController
{
    static readonly IProductRepository repository = new ProductRepository();

    public IEnumerable GetAllProducts()
    {
        return repository.GetAll();
    }

    public TblProductList PostProduct(TblProductList item)
    {
        return repository.Add(item);
    }

    public IEnumerable PutProduct(int id, TblProductList product)
    {

```

```

        product.Id = id;
        if (repository.Update(product))
        {
            return repository.GetAll();
        }
        else
        {
            return null;
        }
    }

    public bool DeleteProduct(int id)
    {
        if (repository.Delete(id))
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}

```

Right click on *Controllers* folder and add new Controller '*HomeController.cs*':

Hide Copy Code

```

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

```

Right click on **ActionResult Product()** and add a view '*Product.cshtml*'.

Hide Shrink ▲ Copy Code

```

@{
    ViewBag.Title = "Products List";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
@section scripts {

    <link href="~/Content/CustomStyle.css" rel="stylesheet" />
    <script src="~/Scripts/jquery-1.10.2.min.js"></script>
    <script src="~/Scripts/angular.js"></script>
    <script src="~/Scripts/AngularDemo.js"></script>
}
<div ng-app="demoModule" id="body">
    <div ng-controller="demoCtrl">
        <h2>AngularJS CRUD Operations with MVC5 WebAPI</h2>

        <h3>List of Products</h3>

        <table ng-cloak>
            <thead>
                <tr>
                    <th style="display: none;">ID</th>
                    <th>Name</th>
                    <th>Category</th>

```

```

        <th>Price</th>
        <th>Actions</th>
    </tr>
</thead>
<tbody>
    <tr ng-repeat="items in productsData">
        <td hidden="hidden">{{items.Id}}</td>
        <td>{{items.Name}}</td>
        <td>{{items.Category}}</td>
        <td>{{items.Price | currency:'&#8377;':2}}</td>
        <td>
            <button ng-model="$scope.Product"
            ng-click="edit(productsData[$index])">Edit</button>
            <button ng-click="delete($index)">Delete</button>
        </td>
    </tr>
</tbody>
<tfoot>
    <tr>
        <td colspan="6">
            <hr />
        </td>
    </tr>
    <tr>
        <td>Total :</td>
        <td></td>
        <td><label ng-bind="total() |
        currency:'&#8377;':2"></label></td>
        <td></td>
    </tr>
</tfoot>
</table>
<br />
<div style="border-top: solid 2px #282828; width: 430px; height: 10px">
</div>

<div ng-show="Product.Id != '' ">
    <div>
        <h2>Update Product</h2>
    </div>
    <div hidden="hidden">
        <label for="id">Id</label>
        <input type="text" data-ng-model="Product.Id" />
    </div>
    <div>
        <label for="name">Name</label>
        <input type="text" data-ng-model="Product.Name" />
    </div>

    <div>
        <label for="category">Category</label>
        <input type="text" data-ng-model="Product.Category" />
    </div>

    <div>
        <label for="price">Price</label>
        <input type="text" data-ng-model="Product.Price" />
    </div>
    <br />
    <div>
        <button data-ng-click="update()">Update</button>
        <button data-ng-click="cancel()">Cancel</button>
    </div>
</div>

<div ng-hide="Product.Id != '' ">

```



```

        <div>
            <h2>Add New Product</h2>
        </div>
        <div>
            <label for="name">Name</label>
            <input type="text" data-ng-model="Product.Name" />
        </div>

        <div>
            <label for="category">Category</label>
            <input type="text" data-ng-model="Product.Category" />
        </div>

        <div>
            <label for="price">Price</label>
            <input type="text" data-ng-model="Product.Price" />
        </div>
        <br />
        <div>
            <button data-ng-click="save()">Save</button>
            <button data-ng-click="clear()">Clear</button>
        </div>
    </div>
</div>
</div>

```

Create a new JavaScript file 'AngularDemo.js' in *Scripts* folder to implement CRUD operations using Angular code.

Hide Shrink ▲ Copy Code

```

// Defining angularjs module
var app = angular.module('demoModule', []);

// Defining angularjs Controller and injecting ProductsService
app.controller('demoCtrl', function ($scope, $http, ProductsService) {

    $scope.productsData = null;
    // Fetching records from the factory created at the bottom of the script file
    ProductsService.GetAllRecords().then(function (d) {
        $scope.productsData = d.data; // Success
    }, function () {
        alert('Error Occured !!!'); // Failed
    });

    // Calculate Total of Price After Initialization
    $scope.total = function () {
        var total = 0;
        angular.forEach($scope.productsData, function (item) {
            total += item.Price;
        })
        return total;
    }

    $scope.Product = {
        Id: '',
        Name: '',
        Price: '',
        Category: ''
    };

    // Reset product details
    $scope.clear = function () {
        $scope.Product.Id = '';
    }

```

```

        $scope.Product.Name = '';
        $scope.Product.Price = '';
        $scope.Product.Category = '';
    }

    //Add New Item
    $scope.save = function () {
        if ($scope.Product.Name != "" &&
            $scope.Product.Price != "" && $scope.Product.Category != "") {
            // Call Http request using $.ajax

            //$.ajax({
            //    type: 'POST',
            //    contentType: 'application/json; charset=utf-8',
            //    data: JSON.stringify($scope.Product),
            //    url: 'api/Product/PostProduct',
            //    success: function (data, status) {
            //        $scope.$apply(function () {
            //            $scope.productsData.push(data);
            //            alert("Product Added Successfully !!!");
            //            $scope.clear();
            //        });
            //    },
            //    error: function (status) { }
            //});

            // or you can call Http request using $http
            $http({
                method: 'POST',
                url: 'api/Product/PostProduct/',
                data: $scope.Product
            }).then(function successCallback(response) {
                // this callback will be called asynchronously
                // when the response is available
                $scope.productsData.push(response.data);
                $scope.clear();
                alert("Product Added Successfully !!!");
            }, function errorCallback(response) {
                // called asynchronously if an error occurs
                // or server returns response with an error status.
                alert("Error : " + response.data.ExceptionMessage);
            });
        }
        else {
            alert('Please Enter All the Values !!');
        }
    };

    // Edit product details
    $scope.edit = function (data) {
        $scope.Product = { Id: data.Id, Name: data.Name, Price: data.Price, Category:
data.Category };
    }

    // Cancel product details
    $scope.cancel = function () {
        $scope.clear();
    }

    // Update product details
    $scope.update = function () {
        if ($scope.Product.Name != "" &&
            $scope.Product.Price != "" && $scope.Product.Category != "") {
            $http({
                method: 'PUT',
                url: 'api/Product/PutProduct/' + $scope.Product.Id,

```

```

        data: $scope.Product
    }).then(function successCallback(response) {
        $scope.productsData = response.data;
        $scope.clear();
        alert("Product Updated Successfully !!!");
    }, function errorCallback(response) {
        alert("Error : " + response.data.ExceptionMessage);
    });
}
else {
    alert('Please Enter All the Values !!!');
}
};

// Delete product details
$scope.delete = function (index) {
    $http({
        method: 'DELETE',
        url: 'api/Product/DeleteProduct/' + $scope.productsData[index].Id,
    }).then(function successCallback(response) {
        $scope.productsData.splice(index, 1);
        alert("Product Deleted Successfully !!!");
    }, function errorCallback(response) {
        alert("Error : " + response.data.ExceptionMessage);
    });
};

});

// Here I have created a factory which is a popular way to create and configure
// services.
// You may also create the factories in another script file which is best practice.

app.factory('ProductsService', function ($http) {
    var fac = {};
    fac.GetAllRecords = function () {
        return $http.get('api/Product/GetAllProducts');
    }
    return fac;
});

```

## Note

You can use `$.ajax` from jQuery script or `$http` from angular.js script to make HTTP request.

It's better to use `$http` because using `$.ajax` is also forcing us to use `$scope.apply` which is not needed if you use `$http`.

You can also use `$resource` which is considered to be best practice for doing CRUD operations in RESTful Web API. This tutorial is for beginners, so I tried to keep things simple by using `$http`.

As per the architecture perspective, instead of controller, you can call `$http` request for **POST**, **PUT**, **DELETE** in our custom factory like I have done for `$http.get()`, so that our controller will look clean and exhibits proper Separation of Concern.

Now, add a Layout view '`_Layout.cshtml`'.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>@ViewBag.Title</title>
  <meta name="viewport" content="width=device-width" />
</head>
<body>
  <div id="body">
    @RenderSection("featured", required: false)
    <section class="content-wrapper main-content clear-fix">
      @RenderBody()
    </section>
  </div>

  @RenderSection("scripts", required: false)
</body>
</html>

```

Add stylesheet '*CustomStyle.css*' to improve the look and feel of the page.

Hide Shrink ▲ Copy Code

```

[ng\:cloak], [ng-cloak], [data-ng-cloak], [x-ng-cloak], .ng-cloak, .x-ng-cloak {
  display: none !important;
}

body {
  margin: 20px;
  font-family: "Arial", "Helvetica", sans-serif;
}

label {
  width: 80px;
  display: inline-block;
}

button {
  display: inline-block;
  outline: none;
  cursor: pointer;
  text-align: center;
  text-decoration: none;
  padding: .4em 1.1em .4em;
  color: #fef4e9;
  border: solid 1px #006fb9;
  background: #1276bb;
}

button:hover {
  text-decoration: none;
  background: #282828;
  border: solid 1px #000;
}

table {
  padding-top: 1em;
}

thead, tfoot {
  font-weight: 600;
}

th, td {

```

```

padding: .1em .5em;
text-align: left;
}

td li, td ul {
    margin: 0;
    padding: 0;
}

td li {
    display: inline;
}

td li::after {
    content: ',';
}

td li:last-child::after {
    content: '';
}

```

## Note

Here, I have defined style for **ng-cloak** directive used in our '*Product.cshtml*'. According to [AngularJS Documentation](#).

The **ngCloak** directive is used to prevent the Angular html template from being briefly displayed by the browser in its raw (uncompiled) form while your application is loading. Use this directive to avoid the undesirable flicker effect caused by the html template display.

The reason for adding the class is because the **ng-cloak** directive is parsed after the html has been displayed, so there is always the possibility that your JS thread dies and still displays anything like `{{something here}}`

Now, change the default controller and action in *Route.Config.cs*.

Hide Copy Code

```

routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{id}",
    defaults: new { controller = "Product", action = "Product", id =
    UrlParameter.Optional }
);

```

And also change the default **routeTemplate** to add action in *WebApiConfig.cs*.

Hide Copy Code

```

config.Routes.MapHttpRoute(
    name: "DefaultApi",
    routeTemplate: "api/{controller}/{action}/{id}",
    defaults: new { id = RouteParameter.Optional }
);

```

Hit Ctrl+F5.

That's it !!!



## AngularJS CRUD Operations with MVC5 WebAPI

### List of Products

Name	Category	Price	Actions	
Iphone 5s	Phone	₹33,425.32	Edit	Delete
Dell E5440	Laptop	₹45,632.21	Edit	Delete
Lumia 730	Phone	₹14,532.52	Edit	Delete
Total :		₹93,590.05		

### Add New Product

Name	<input type="text"/>
Category	<input type="text"/>
Price	<input type="text"/>

## AngularJS CRUD Operations with MVC5 WebAPI

### List of Products

Name	Category	Price	Actions	
Iphone 5s	Phone	₹33,425.32	Edit	Delete
Dell E5440	Laptop	₹45,632.21	Edit	Delete
Lumia 730	Phone	₹14,532.52	Edit	Delete
Total :		₹93,590.05		

### Update Product

Name	<input type="text" value="Lumia 730"/>
Category	<input type="text" value="Phone"/>
Price	<input type="text" value="14532.52"/>

Results		Messages		
	Id	Name	Category	Price
1	1	Iphone 5s	Phone	33425.32
2	2	Dell E5440	Laptop	45632.21
3	4	Lumia 730	Phone	14532.52

Congratulations!!! Now, you have successfully implemented CRUD operations in ASP.NET MVC 5 using WebAPI 2 using AngularJS.

Please comment for any queries.

## Source Code

I have uploaded a sample project with SQL scripts, in case you need them. Don't forget to change the server name in **ConnectionString** of *Web.Config*.

Happy coding! :)



# CRUD en ASP.NET MVC 5 utilizando WebAPI 2 con AngularJS



**Indresh Prajapati**, 22 Feb 2016 [CPOL](#)



4,79 (33 votos)



Califica esto: [vote 1](#)[vote 2](#)[vote 3](#)[vote 4](#)[vote 5](#)

Esta sugerencia ayudará a los principiantes a implementar operaciones CRUD en ASP.NET MVC 5 utilizando WebAPI 2 con lenguaje de scripting como AngularJS y Database como MS SQL 2008R2.

- [Descargar demo - 3.5 MB](#)

## Introducción

Como usted sabe, AngularJS es el framework JavaScript más popular para desarrollar aplicaciones de una sola página. También puede utilizar Angular para operaciones **Insert**, **Update**, **Delete** y **Retrieve**. Esta sugerencia demostrará cómo usar Angular con MVC5 y WebAPI2 para operaciones CRUD ( **Create**, **Read**, Actualizar, **Delete** ). Muchos desarrolladores experimentados encontrarán este consejo muy básico, pero como la punta está escrita desde la perspectiva de los principiantes, he tratado de mantener las cosas simples.


## Fondo

En mi [artículo](#) anterior, he demostrado operaciones CRUD con KnockoutJS. Aquí, usaré la misma base de datos, diseño y contenido, pero en vez de Knockout, usaré AngularJS y las operaciones serán manejadas en WebAPI.

## Uso del código

Vamos a empezar !!!

Cree la tabla ' **TblProductList** ' con este esquema.


	Column Name	Data Type	Allow Nulls
	<b>Id</b>	int	<input type="checkbox"/>
	Name	varchar(20)	<input checked="" type="checkbox"/>
	Category	varchar(20)	<input checked="" type="checkbox"/>
	Price	decimal(10, 2)	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

Cree un nuevo proyecto en ASP.NET MVC 5 y asígnele el nombre que prefiera y seleccione una plantilla de proyecto vacía.


Marque MVC y API Web en *Agregar carpetas y referencias principales* para:

New ASP.NET Project - AngularJS\_Demo


Select a template:




Empty




Web Forms




MVC



Web API



Single Page Application



Facebook

Add folders and core references for:

☐ Web Forms
☒ MVC
☒ Web API

☐ Add unit tests

Test project name:

An empty project template for creating ASP.NET applications. This template does not have any it.

[Learn more](#)

Change Authentication

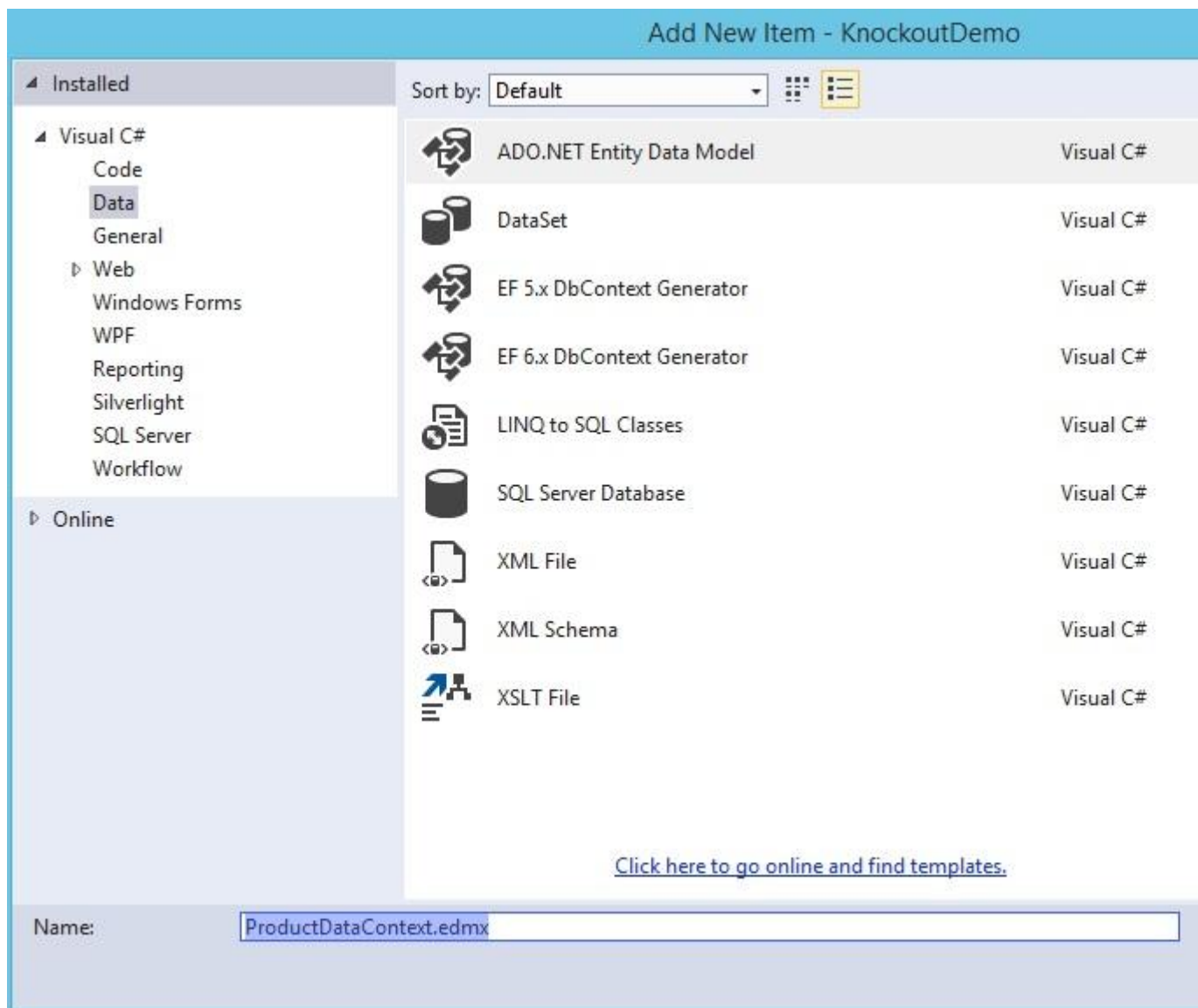
Authentication: **No Authentication**

OK

Instale Entity Framework 6, JQuery y AngularJS en su proyecto utilizando NuGet Package Manager.

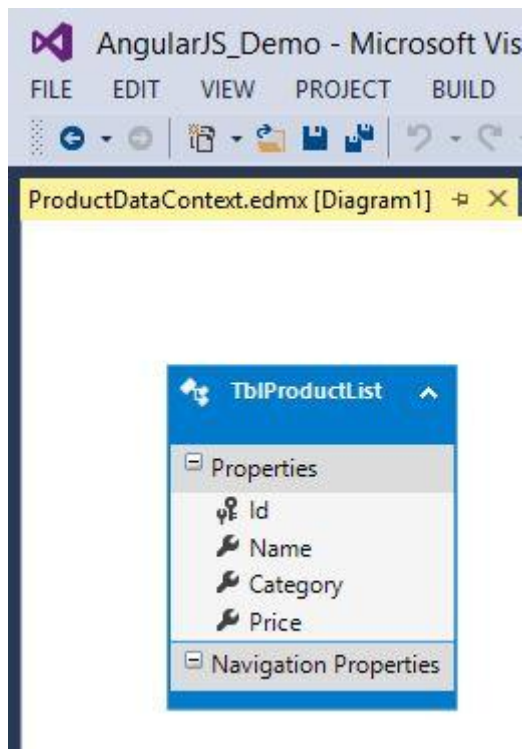
También puede descargar jquery.js y angular.js desde su sitio web oficial y pegarlo en la carpeta ' *Scripts* ' de su proyecto.

Haga clic con el botón derecho del ratón en la carpeta *Modelo* y agregue un nuevo modelo de datos de entidad ADO.NET. *Denomínelo* como " *ProductDataContext.edmx* ".



Elija 'Generar de la base de datos' y configure la cadena de conexión según su servidor SQL.

Después de generar el modelo, obtendrá la entidad de **TblProductList**.



Crear una nueva carpeta 'Interface' en el directorio raíz. Agregue una nueva clase 'IProductRepository.cs'.

Hide Copy Code

```
Interfaz IProductRepository
{
    IEnumerable <TblProductList> GetAll ();
    TblProductList Get ( int id);
    TblProductList Agregar (elemento TblProductList);
    Actualización de bool (artículo de TblProductList);
    Bool Eliminar ( int id);
}
```

Crear nueva carpeta 'Repositorios' en el directorio raíz . Agregue una nueva clase 'ProductRepository.cs' . Implementar los métodos para **Create** , **Read** , actualizar, **Delete** utilizando Entity Framework.

Hide Shrink ▲ Copy Code

```
Clase pública ProductRepository: IProductRepository
{
    ProductDBEntities ProductDB = new ProductDBEntities ();

    Public IEnumerable <TblProductList> GetAll ()
    {
        // TO DO: Código para obtener La Lista de todos Los registros de La base
de datos
        Return ProductDB.TblProductLists;
    }

    Public TblProductList Get ( int id)
    {
        // TO DO: Código para encontrar un registro en La base de datos
    }
}
```

```

        Return ProductDB.TblProductLists.Find (id);
    }

    Public TblProductList Add (TblProductList item)
    {
        If (item == null )
        {
            Throw new ArgumentNullException ( " item" );
        }

        // PARA HACER: Código para guardar el registro en la base de datos
        ProductDB.TblProductLists.Add (item);
        ProductDB.SaveChanges ();
        Elemento de devolución;
    }

    Public bool Update (elemento TblProductList)
    {
        If (item == null )
        {
            Throw new ArgumentNullException ( " item" );
        }

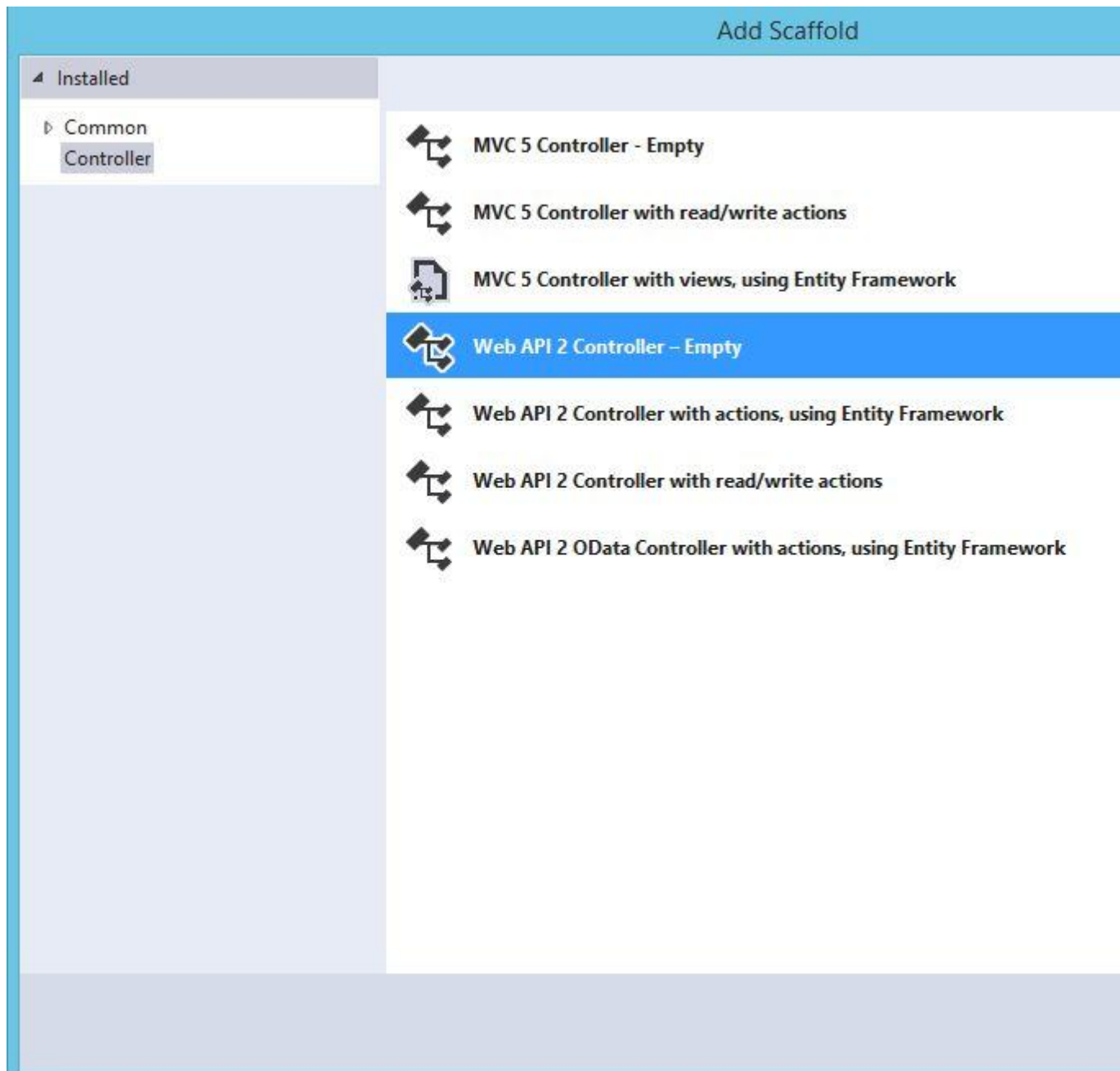
        // PARA HACER: Código para actualizar el registro en la base de datos
        Var products = ProductDB.TblProductLists. Soltero (a => a.Id ==
item.Id);
        Products.Name = item.Name;
        Products.Category = item.Category;
        Products.Price = item.Price;
        ProductDB.SaveChanges ();

        Devuelve verdadero ;
    }

    Public bool Eliminar ( int id)
    {
        // TO DO: Código para eliminar Los registros de la base de datos
        TblProductList productos = ProductDB.TblProductLists.Find (id);
        ProductDB.TblProductLists.Remove (productos);
        ProductDB.SaveChanges ();
        Devuelve verdadero ;
    }
}

```

Haga clic con el botón derecho del ratón en la carpeta *Controladores* y agregue el nuevo Controlador Vacío de WebAPI 2 ' *ProductController.cs* ':



Hide Shrink ▲ Copy Code

```
Clase pública ProductController: ApiController
{
    Static readonly IProductRepository repository = nuevo ProductRepository ();

    Public IEnumerable GetAllProducts ()
    {
        Return repository.GetAll ();
    }

    Public TblProductList PostProduct (artículo TblProductList)
    {
        Return repository.Add (item);
    }

    Public IEnumerable PutProduct ( int id, producto TblProductList)
```

```

        {
            Product.Id = id;
            If (repository.Update (product))
            {
                Return repository.GetAll ();
            }
            más
            {
                Return null ;
            }
        }

        Public bool EliminarProducto ( int id)
        {
            If (repository.Delete (id))
            {
                Devuelve verdadero ;
            }
            más
            {
                Return false ;
            }
        }
    }
}

```

Haga clic con el botón derecho en la carpeta *Controladores* y agregue el nuevo Controlador 'HomeController.cs':

Hide Copy Code

```

Clase pública HomeController: Controlador
{
    Public ActionResult Producto ()
    {
        Return View ();
    }
}

```

Haga clic derecho en **ActionResult Product()** y agregue una vista 'Product.cshtml'.

Hide Shrink ▲ Copy Code

```

Todos los derechos reservados
ViewBag.Title = "Lista de productos";
Layout = "~/ Views / Shared / _Layout.cshtml";
}
Scripts @section {

    < Link href = " ~ / Content / CustomStyle.css" rel = " stylesheet" / >
    < Script src = " ~ / Scripts / jquery-1.10.2.min.js" > </ script>
    < Script src = " ~ / Scripts / angular.js" > </ script>
    < Script src = " ~ / Scripts / AngularDemo.js" > </ script>
}
< Div ng-app = " demoModule" id = " body" >
    < Div ng-controller = " demoCtrl" >
        < H2 > Operaciones AngularJS CRUD con MVC5 WebAPI < / h2 >

        < H3 > Lista de productos < / h3 >

        < Table ng-cloak >
            < Thead >

```

```

        < Tr >
            < Th style = " display: none;" > ID < / th >
            < Th > Nombre < / th >
            < Th > Categoría < / th >
            < Th > Precio < / th >
            < Th > Acciones < / th >
        < / Tr >
    < / Thead >
    < Tbody >
        < Tr ng-repeat = " artículos en productsData" >
            < Td hidden = " hidden" > {{items.Id}} < / td >
            < Td > {{items.Name}} < / td >
            < Td > {{items.Category}} < / td >
            < Td > {{items.Price | Moneda: '& # 8377;': 2}} < / td >
            < Td >
                < Button ng-model = " $ scope.Product"
                Ng-click = " edit (productsData [$ index])" > Editar < / button >
                < Button ng-click = " delete ($ index)" > Eliminar < / button >
            < / Td >
        < / Tr >
    < / Tbody >
    < Tfoot >
        < Tr >
            < Td colspan = " 6" >
                < Hr / >
            < / Td >
        < / Tr >
        < Tr >
            < Td > Total: < / td >
            < Td > < / td >
            < Td > < label ng-bind = " total () |
                Moneda: '& # 8377;': 2 " > < / label > < / td >
            < Td > < / td >
        < / Tr >
    < / Tfoot >
< / Table >
< Br / >
< Div style = " border-top: sólido 2px # 282828; anchura: 430px; altura:
10px" > < / div >

< Div ng-show = " Product.Id! = ''" >
    < Div >
        < H2 > Actualizar producto < / h2 >
    < / Div >
    < Div hidden = " hidden" >
        < Label for = " id" > Id < / label >
        < Input type = " text" datos-ng-modelo = " Product.Id" / >
    < / Div >
    < Div >
        < Label for = " name" > Nombre < / label >
        < Input type = " text" data-ng-model = " Product.Name" / >
    < / Div >

    < Div >
        < Label for = " category" > Categoría < / label >
        < Input type = " text" data-ng-model = " Producto.Categoría" / >
    < / Div >

    < Div >
        < Label for = " price" > Precio < / label >
        < Input type = " text" data-ng-model = " Producto.Precio" / >
    < / Div >
    < Br / >
    < Div >
        < Button data-ng-click = " update ()" > Actualizar < / button >
        < Button data-ng-click = " cancel ()" > Cancelar < / button >

```



```

        < / Div >
    < / Div >

    < Div ng-hide = " Product.Id! = ''" >
        < Div >
            < H2 > Añadir nuevo producto < / h2 >
        < / Div >
        < Div >
            < Label for = " name" > Nombre < / label >
            < Input type = " text" data-ng-model = " Product.Name" / >
        < / Div >

        < Div >
            < Label for = " category" > Categoría < / label >
            < Input type = " text" data-ng-model = " Producto.Categoría" / >
        < / Div >

        < Div >
            < Label for = " price" > Precio < / label >
            < Input type = " text" data-ng-model = " Producto.Precio" / >
        < / Div >
        < Br />
        < Div >
            < Button data-ng-click = " save ()" > Guardar < / button >
            < Button data-ng-click = " clear ()" > Borrar < / button >
        < / Div >
    < / Div >
< / Div >
< / Div >

```

Cree un nuevo archivo JavaScript ' *AngularDemo.js* ' en la carpeta *Scripts* para implementar operaciones CRUD usando código Angular.

Hide Shrink ▲ Copy Code

```

// Definición del módulo angularjs
Var app = angular.module ( ' demoModule' , []);

// Definición de ángulos Controlador e inyección ProductosServicio
App.controller ( ' demoCtrl' , function ($ scope, $ http, ProductsService) {

    $ Scope.productsData = null ;
    // Recuperación de registros de fábrica creados en la parte inferior del archivo
de script
    ProductsService.GetAllRecords (). Entonces ( function (d) {
        $ Scope.productsData = d.data; // Éxito
    }, function () {
        Alert ( ' Error Occured !!!' ); Falló
    });

    // Calcular el precio total después de la inicialización
    $ Scope.total = function () {
        Var total = 0 ;
        Angular.forEach ($ scope.productsData, function (item) {
            Total + = item.Price;
        })
        Retorno total;
    }

    $ Scope.Product = {
        Id: ' ' ,
        Nombre: ' ' ,
        Precio: ' ' ,
    }
}

```

```

        Categoría: ' '
    };

    // Restablecer detalles del producto
    $ Scope.clear = function () {
        $ Scope.Product.Id = ' ' ;
        $ Scope.Product.Name = ' ' ;
        $ Scope.Product.Price = ' ' ;
        $ Scope.Product.Category = ' ' ;
    }

    // Añadir nuevo elemento
    $ Scope.save = function () {
        If ($ scope.Product.Name! = " " &&
            $ Scope.Product.Price! = " " && $ scope.Product.Category! = " " ) {
            // Solicitud Http de llamada usando $ .ajax

            // $ .ajax ({
            // tipo: 'POST',
            // contentType: 'application / json; Charset = utf-8 ',
            // data: JSON.stringify ($ scope.Product),
            // url: 'api / Product / PostProduct',
            // éxito: función (datos, estado) {
            // $ scope. $ Apply (function () {
            // $ scope.productsData.push (data);
            // alert ("Producto añadido con éxito!");
            // $ scope.clear ();
            // });
            // },
            // error: function (status) {}
            // });

            // o puede llamar a la solicitud Http usando $ http
            $ Http ({
                Método: ' POST' ,
                Url: ' api / Product / PostProduct / ' ,
                Datos: $ scope.Product
            }). Entonces ( function successCallback (response) {
                // este callback se llamará asincrónicamente
                // cuando la respuesta está disponible
                $ Scope.productsData.push (response.data);
                $ Scope.clear ();
                Alerta ( " Producto añadido con éxito!" );
            }, function errorCallback (respuesta) {
                // se llama de forma asincrónica si se produce un error
                // o el servidor devuelve la respuesta con un estado de error.
                Alert ( " Error:" + response.data.ExceptionMessage);
            });
        }
        Else {
            Alerta ( ' Por favor, ingrese todos los valores!' );
        }
    };

    // Editar detalles del producto
    $ Scope.edit = function (data) {
        $ Scope.Product = {Id: data.Id, Name: data.Name, Price: data.Price,
            Category: data.Category};
    }

    // Cancelar detalles del producto
    $ Scope.cancel = function () {
        $ Scope.clear ();
    }

    // Actualizar los detalles del producto

```

```

$ Scope.update = function () {
    If ($ scope.Product.Name! = " " &&
        $ Scope.Product.Price! = " " && $ scope.Product.Category! = " " ) {
        $ Http ({
            Método: ' PUT' ,
            Url: ' api / Product / PutProduct /' + $ scope.Product.Id,
            Datos: $ scope.Product
        }). Entonces ( function successCallback (response) {
            $ Scope.productsData = response.data;
            $ Scope.clear ();
            Alerta ( " Producto actualizado con éxito!" );
        }, function errorCallback (respuesta) {
            Alert ( " Error:" + response.data.ExceptionMessage);
        });
    }
    Else {
        Alerta ( ' Por favor, ingrese todos los valores!' );
    }
};

// Borrar detalles del producto
$ Alcance. Delete = function (índice) {
    $ Http ({
        Método: ' DELETE' ,
        Url: ' api / Product / DeleteProduct /' + $ scope.productsData [índice]
        .Id,
    }). Entonces ( function successCallback (response) {
        $ Scope.productsData.splice (índice, 1 );
        Alerta ( " Producto eliminado con éxito!" );
    }, function errorCallback (respuesta) {
        Alert ( " Error:" + response.data.ExceptionMessage);
    });
};

});

// Aquí he creado una fábrica que es una forma popular de crear y configurar
servicios.
// También puede crear las fábricas en otro archivo de script que sea la mejor
práctica.

App.factory ( ' ProductsService' , function ($ http) {
    Var fac = {};
    Fac.GetAllRecords = function () {
        Return $ http.get ( ' api / Product / GetAllProducts' );
    }
    Return fac;
});

```

## Nota

Puede usar `$.ajax` de la `$.ajax` de comandos jQuery o `$http` de la secuencia de comandos angular.js para realizar la solicitud HTTP.

Es mejor usar `$http` porque usar `$.ajax` también nos obliga a usar `$scope.apply` que no es `$scope.apply` si usas `$http` .

También puede usar `$resource` que se considera la mejor práctica para realizar operaciones CRUD en la API Web RESTful. Este tutorial es para principiantes, así que traté de mantener las cosas simples usando `$http` .

Según la perspectiva de la arquitectura, en vez del regulador, usted puede llamar la petición de `$http` para **POST**, **PUT**, **DELETE** en nuestra fábrica de encargo como he hecho para `$http.get()`, de modo que nuestro regulador parecerá limpio y exhibe la separación apropiada de Preocupación.

Ahora, agregue una vista de diseño '`_Layout.cshtml`'.

Hide Copy Code

```
< ! DOCTYPE html >
< Html lang = " en" >
< Head >
  < Meta charset = " utf-8" / >
  < Title > @ ViewBag.Title < / title >
  < Meta name = " viewport" content = " width = device-width" / >
< / Head >
< Body >
  < Div id = " body" >
    @RenderSection ("featured", requiere: false)
    < Section class = " content-wrapper contenido principal clear-fix" >
      @RenderBody ()
    < / Section >
  < / Div >

  @RenderSection ("scripts", requiere: false)
< / Body >
< / Html >
```

Agregue la hoja de estilos '`CustomStyle.css`' para mejorar la *apariencia* de la página.

Hide Shrink ▲ Copy Code

```
[Ng \: capa], [ng-capa], [data-ng-cloak], [x-ng-capa],. Ng-cloak,. X-ng-cloak {
  Display : ninguno ! Importante ;
}

Cuerpo {
  Margen : 20px;
  Font-family : "Arial", "Helvetica", sans-serif;
}

Etiqueta {
  Ancho : 80px;
  Display : inline-block;
}

Botón {
  Display : inline-block;
  Esquema : ninguno;
  Cursor : puntero;
  Text-align : center;
  Text-decoration : ninguno;
  Relleno : .4em 1.1em .4em;
  Color : # fef4e9;
  Frontera : sólido 1px # 006fb9;
  Fondo : # 1276bb;
}

Botón: hover {
  Text-decoration : ninguno;
  Fondo : # 282828;
```

```

        Frontera : sólido 1px # 000;
    }

    Tabla {
        Padding-top : 1em;
    }

    Tead { sustantivo } tfoot {
        Peso de la fuente : 600;
    }

    Th , td {
        Relleno : .1em .5em;
        Text-align : left;
    }

    Td li , td ul {
        Margen : 0;
        Relleno : 0;
    }

    Td li {
        Display : en línea;
    }

    Td li :: after {
        Contenido : ',';
    }

    Td li: last-child :: después de {
        Contenido : '';
    }

```

## Nota

Aquí, he definido el estilo de **ng-cloak** directiva **ng-cloak** utilizada en nuestro 'Product.cshtml'.

De acuerdo con la [documentación de AngularJS](#).

La **ngCloak** se utiliza para evitar que la plantilla html Angular se muestre brevemente por el navegador en su forma cruda (sin compilar) mientras la aplicación se está cargando. Utilice esta directiva para evitar el efecto de parpadeo indeseable causado por la visualización de la plantilla html.

La razón para añadir la clase es porque la directiva **ng-cloak** se analiza después de que se ha mostrado el html, por lo que siempre existe la posibilidad de que su hilo JS muera y todavía muestra algo similar `{{something here}}`

Ahora, cambie el controlador y la acción *predeterminados* en *Route.Config.cs*.

Hide Copy Code

```

Routes.MapRoute (
    Nombre: " Default" ,
    Url: " {controller} / {action} / {id}" ,
    Defaults: new {controller = " Producto" , action = " Producto" , id =
    UrlParameter.Optional}
);

```

Y también cambia el `routeTemplate` por defecto para añadir acción en `WebApiConfig.cs` .

[Hide](#) [Copy Code](#)

```
Config.Routes.MapHttpRoute (
    Nombre: " DefaultApi" ,
    RouteTemplate: " api / {controller} / {action} / {id}" ,
    Defaults: new {id = RouteParameter.Optional}
);
```

Pulse Ctrl + F5.

Eso es !!!



## AngularJS CRUD Operations with MVC5 WebAPI

### List of Products

Name	Category	Price	Actions	
Iphone 5s	Phone	₹33,425.32	Edit	Delete
Dell E5440	Laptop	₹45,632.21	Edit	Delete
Lumia 730	Phone	₹14,532.52	Edit	Delete

---

**Total :** ₹93,590.05

---

### Add New Product

Name	<input type="text"/>
Category	<input type="text"/>
Price	<input type="text"/>

---

## AngularJS CRUD Operations with MVC5 WebAPI

### List of Products

Name	Category	Price	Actions	
Iphone 5s	Phone	₹33,425.32	Edit	Delete
Dell E5440	Laptop	₹45,632.21	Edit	Delete
Lumia 730	Phone	₹14,532.52	Edit	Delete
Total :		₹93,590.05		

### Update Product

Name	<input type="text" value="Lumia 730"/>
Category	<input type="text" value="Phone"/>
Price	<input type="text" value="14532.52"/>

Results		Messages		
	Id	Name	Category	Price
1	1	Iphone 5s	Phone	33425.32
2	2	Dell E5440	Laptop	45632.21
3	4	Lumia 730	Phone	14532.52

!!!Felicitaciones!!! Ahora, ha implementado con éxito las operaciones CRUD en ASP.NET MVC 5 utilizando WebAPI 2 utilizando AngularJS.

Por favor, comente cualquier consulta.

## Código fuente

He subido un proyecto de ejemplo con secuencias de comandos SQL, en caso de que las necesite. No olvide cambiar el nombre del servidor en **ConnectionString** de *Web.Config*.

Codificación feliz :)