

Create A Talent Restful Service

This tutorial shows how to support CRUD operations in an HTTP service using ASP.NET Web API.

Software versions used in the tutorial

CRUD stands for "Create, Read, Update, and Delete," which are the four basic database operations. Many HTTP services also model CRUD operations through REST or REST-like APIs.

In this tutorial, you will build a very simple web API to manage a list of products. Each product will contain a name, price, and category (such as "toys" or "hardware"), plus a product ID.

The products API will expose following methods.

Action	HTTP method	Relative URI
Get a list of all talents	GET	/api/talents
Get a talent by ID	GET	/api/talents/ <i>id</i>
Create a new talent	POST	/api/talents
Update a talent	PUT	/api/talents/ <i>id</i>
Delete a talent	DELETE	/api/talents/ <i>id</i>

Notice that some of the URIs include the talent ID in path. For example, to get the product whose ID is 28, the client sends a GET request for <http://hostname/api/talents/2>.

For example: <http://csc123.azurewebsites.net/api/talents/2>

Resources

The products API defines URIs for two resource types:

Resource	URI
The list of all the talents.	<code>/api/talents</code>
An individual talent.	<code>/api/talents/<i>id</i></code>

Methods

The four main HTTP methods (GET, PUT, POST, and DELETE) can be mapped to CRUD operations as follows:

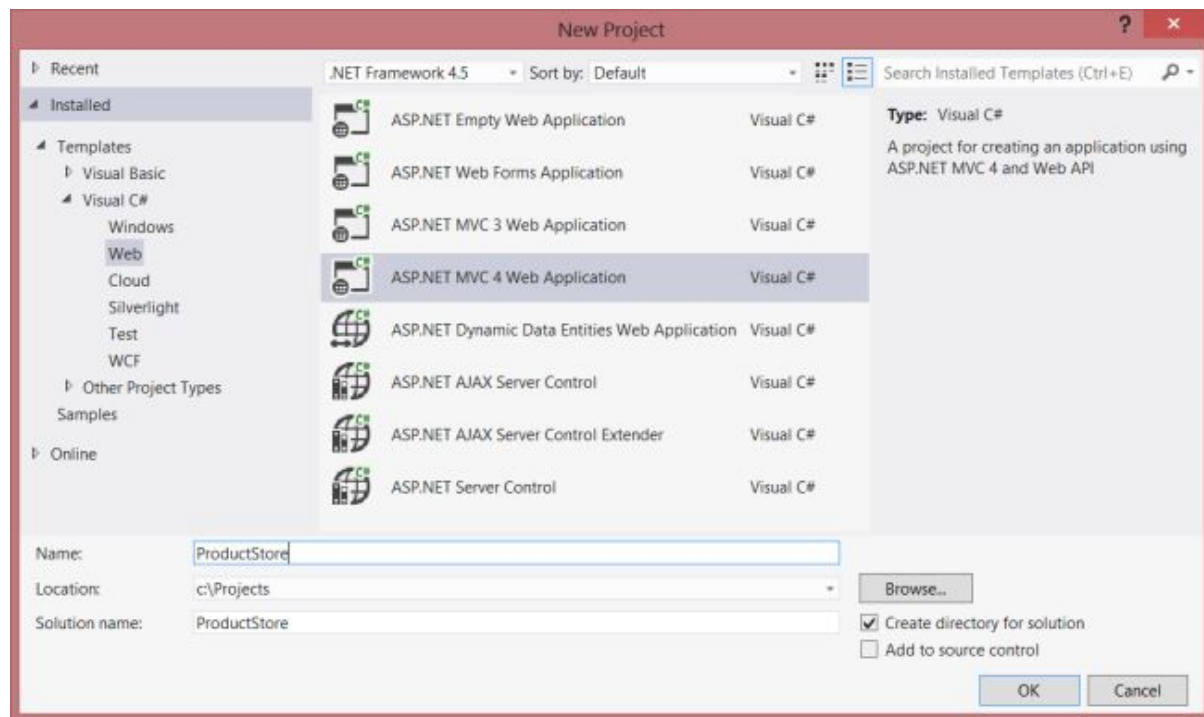
- GET retrieves the representation of the resource at a specified URI. GET should have no side effects on the server.
- PUT updates a resource at a specified URI. PUT can also be used to create a new resource at a specified URI, if the server allows clients to specify new URIs. For this tutorial, the API will not support creation through PUT.
- POST creates a new resource. The server assigns the URI for the new object and returns this URI as part of the response message.
- DELETE deletes a resource at a specified URI.

Note: The PUT method replaces the entire product entity. That is, the client is expected to send a complete representation of the updated product. If you want to support partial updates, the PATCH method is preferred. This tutorial does not implement PATCH.

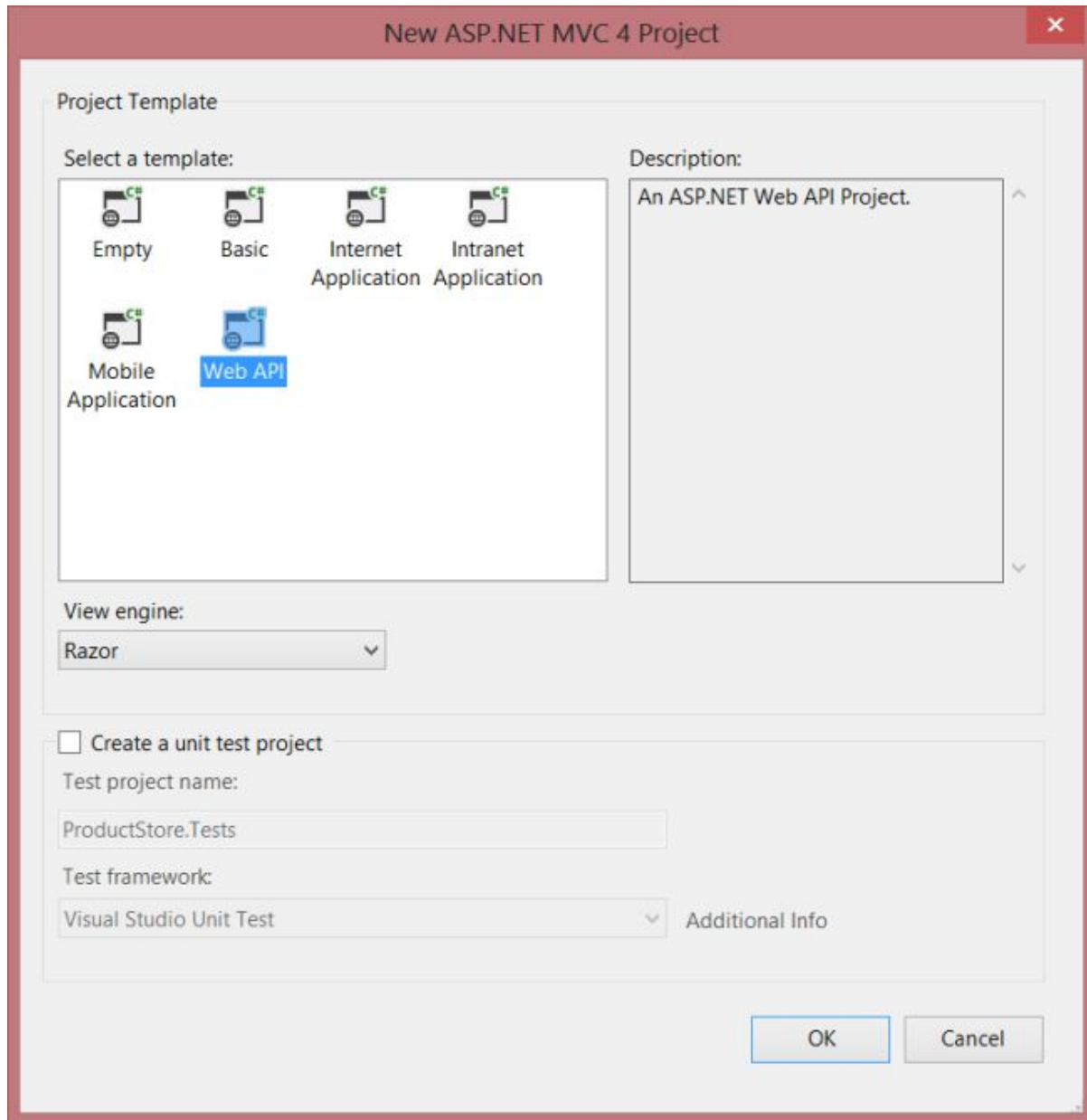
Create a New Web API Project

Start by running Visual Studio and select **New Project** from the **Start** page. Or, from the **File** menu, select **New** and then **Project**.

In the **Templates** pane, select **Installed Templates** and expand the **Visual C#** node. Under **Visual C#**, select **Web**. In the list of project templates, select **ASP.NET MVC 4 Web Application**. Name the project "ProductStore" and click **OK**.



In the **New ASP.NET MVC 4 Project** dialog, select **Web API** and click **OK**.

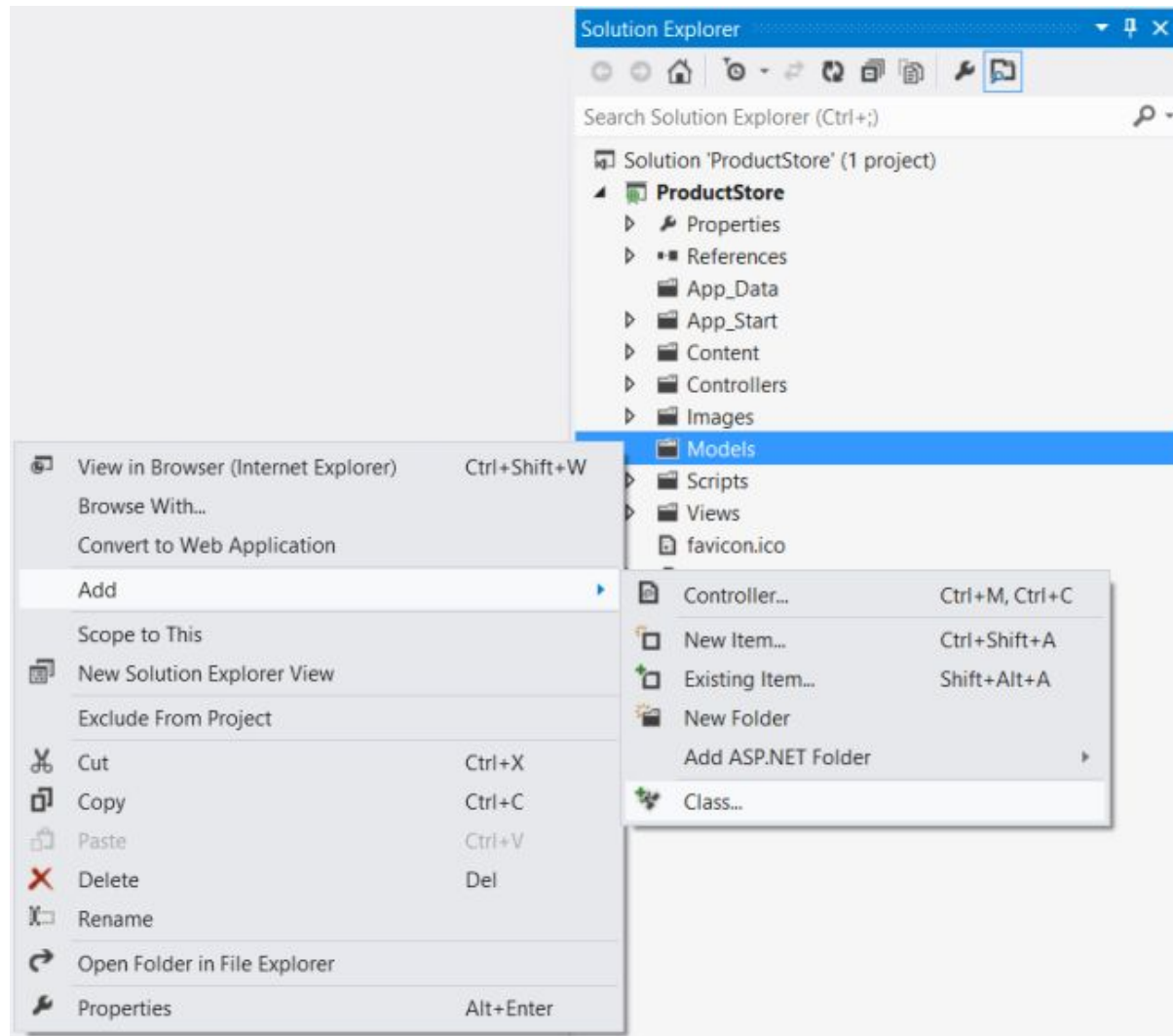


Adding a Model

A *model* is an object that represents the data in your application. In ASP.NET Web API, you can use strongly typed CLR objects as models, and they will automatically be serialized to XML or JSON for the client.

For the ProductStore API, our data consists of products, so we'll create a new class named **Talent**

If Solution Explorer is not already visible, click the **View** menu and select **Solution Explorer**. In Solution Explorer, right-click the **Models** folder. From the context menu, select **Add**, then select **Class**. Name the class "**Talent**".



Add the following properties to the **Talent** class.

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Web;
```

```
namespace ProductStore.Models
```

```

{

    public class Talent

    {

        public int Id { get; set; }

        public string Name { get; set; }

        public string ShortName { get; set; }

        public string Reknown { get; set; }

        public string Bio { get; set; }

    }

}

```

Adding a Repository

We need to store a collection of products. It's a good idea to separate the collection from our service implementation. That way, we can change the backing store without rewriting the service class. This type of design is called the *repository* pattern. Start by defining a generic interface for the repository.

In Solution Explorer, right-click the **Models** folder. Select **Add**, then select **New Item**.

Now add another class to the Models folder, named "TalentRepository." Add the following implementation:

```

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

```

```

namespace ProductStore.Models
{
    public class TalentRepository
    {
        private List<Talent> talents = new List<Talent>();

        private int _nextId = 1;

        public TalentRepository()
        {
            talents.Add(new Talent
            {
                Name = "Barot Bellingham",
                ShortName = "Barot_Bellingham",
                Reknown = "Royal Academy of Painting and Sculpture",
                Bio = "Barot has just finished his final year at The Royal Academy of
Painting and Sculpture, where he excelled in glass etching paintings and portraiture.
Hailed as one of the most diverse artists of his generation, Barot is equally as
skilled with watercolors as he is with oils, and is just as well-balanced in
different subject areas. Barot's collection entitled \"The Un-Collection\" will adorn
the walls of Gilbert Hall, depicting his range of skills and sensibilities - all of
them, uniquely Barot, yet undeniably different"
            });

            talents.Add(new Talent
            {
                Id = 1,
                Name = "Jonathan G. Ferrar II",
                ShortName = "Jonathan_Ferrar",
                Reknown = "Artist to Watch in 2012",
                Bio = "The Artist to Watch in 2012 by the London Review, Johnathan
has already sold one of the highest priced-commissions paid to an art student, ever

```

on record. The piece, entitled Gratitude Resort, a work in oil and mixed media, was sold for \$750,000 and Jonathan donated all the proceeds to Art for Peace, an organization that provides college art scholarships for creative children in developing nations"

```
    });

    talents.Add(new Talent
    {
        Id = 2,

        Name = "Hillary Hewitt Goldwynn-Post",

        ShortName = "Hillary_Goldwynn",

        Reknown = "New York University",

        Bio = "Hillary is a sophomore art sculpture student at New York University, and has already won all the major international prizes for new sculptors, including the Divinity Circle, the International Sculptor's Medal, and the Academy of Paris Award. Hillary's CAC exhibit features 25 abstract watercolor paintings that contain only water images including waves, deep sea, and river."

    });

    talents.Add(new Talent
    {
        Id = 3,

        Name = "Hassum Harrod",

        ShortName = "Hassum_Harrod",

        Reknown = "Art College in New Dehli",

        Bio = "The Art College in New Dehli has sponsored Hassum on scholarship for his entire undergraduate career at the university, seeing great promise in his contemporary paintings of landscapes - that use equal parts muted and vibrant tones, and are almost a contradiction in art. Hassum will be speaking on \"The use and absence of color in modern art\" during Thursday's agenda."

    });
}
```



```
public IEnumerable<Talent> GetAll()
```

```
{
```

```
    return talents;
```

```
}
```

```
public Talent Get(int id)
```

```
{
```

```
    return talents.Find(p => p.Id == id);
```

```
}
```

```
public Talent Add(Talent item)
```

```
{
```

```
    if (item == null)
```

```
    {
```

```
        throw new ArgumentNullException("item");
```

```
    }
```

```
    item.Id = _nextId++;
```

```
    talents.Add(item);
```

```
    return item;
```

```
}
```

```
public void Remove(int id)
```

```
{
```

```
    talents.RemoveAll(p => p.Id == id);
```

```

    }

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

        int index = talents.FindIndex(p => p.Id == item.Id);

        if (index == -1)
        {
            return false;
        }

        talents.RemoveAt(index);

        talents.Add(item);

        return true;
    }
}

} //end of the controller class

```

The repository keeps the list in local memory. This is OK for a tutorial, but in a real application, you would store the data externally, either a database or in cloud storage. The repository pattern will make it easier to change the implementation later.

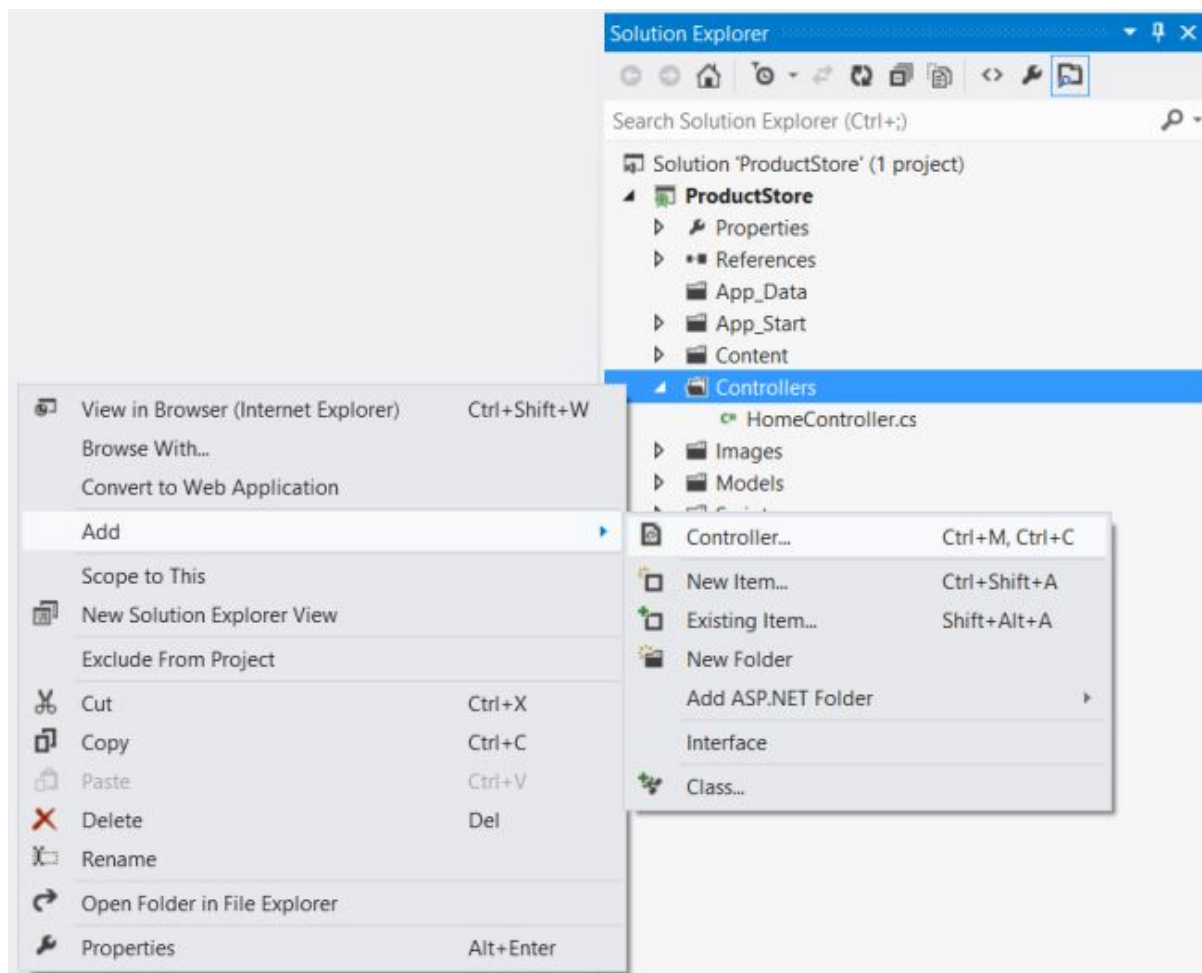
Adding a Web API Controller

If you have worked with ASP.NET MVC, then you are already familiar with controllers. In ASP.NET Web API, a *controller* is a class that handles HTTP requests from the client. The New Project wizard created two controllers for you when it created the project. To see them, expand the Controllers folder in Solution Explorer.

- HomeController is a traditional ASP.NET MVC controller. It is responsible for serving HTML pages for the site, and is not directly related to our web API.
- ValuesController is an example WebAPI controller.

Go ahead and delete ValuesController, by right-clicking the file in Solution Explorer and selecting **Delete**. Now add a new controller, as follows:

In **Solution Explorer**, right-click the the Controllers folder. Select **Add** and then select **Controller**.



In the **Add Controller** wizard, name the controller "ProductsController". In the **Template** drop-down list, select **Empty API Controller**. Then click **Add**.

It is not necessary to put your controllers into a folder named Controllers. The folder name is not important; it is simply a convenient way to organize your source files.

The **Add Controller** wizard will create a file named `ProductsController.cs` in the `Controllers` folder. If this file is not open already, double-click the file to open it. Add the following **using** statement:

```
using ProductStore.Models;
```

Add a field that holds an **IPProductRepository** instance.

```
public class TalentsController : ApiController
{
    static readonly TalentRepository repository = new TalentRepository();
}
```

Calling `new ProductRepository()` in the controller is not the best design, because it ties the controller to a particular implementation of **IPProductRepository**. For a better approach, see [Using the Web API Dependency Resolver](#).

Getting a Resource

The ProductStore API will expose several "read" actions as HTTP GET methods. Each action will correspond to a method in the **ProductsController** class.

Action	HTTP method	Relative URI
Get a list of all products	GET	/api/products
Get a product by ID	GET	/api/products/ <i>id</i>

```
using ProductStore.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Http;
```

```
using System.Web.Http;

using System.Web.Http.Cors;

namespace ProductStore.Controllers
{
    public class TalentsController : ApiController
    {
        static readonly TalentRepository repository = new TalentRepository();

        [EnableCors(origins: "*", headers: "*", methods: "*")]

        [Route("api/talents")]

        public IEnumerable<Talent> GetAllTalents()
        {
            return repository.GetAll();
        }

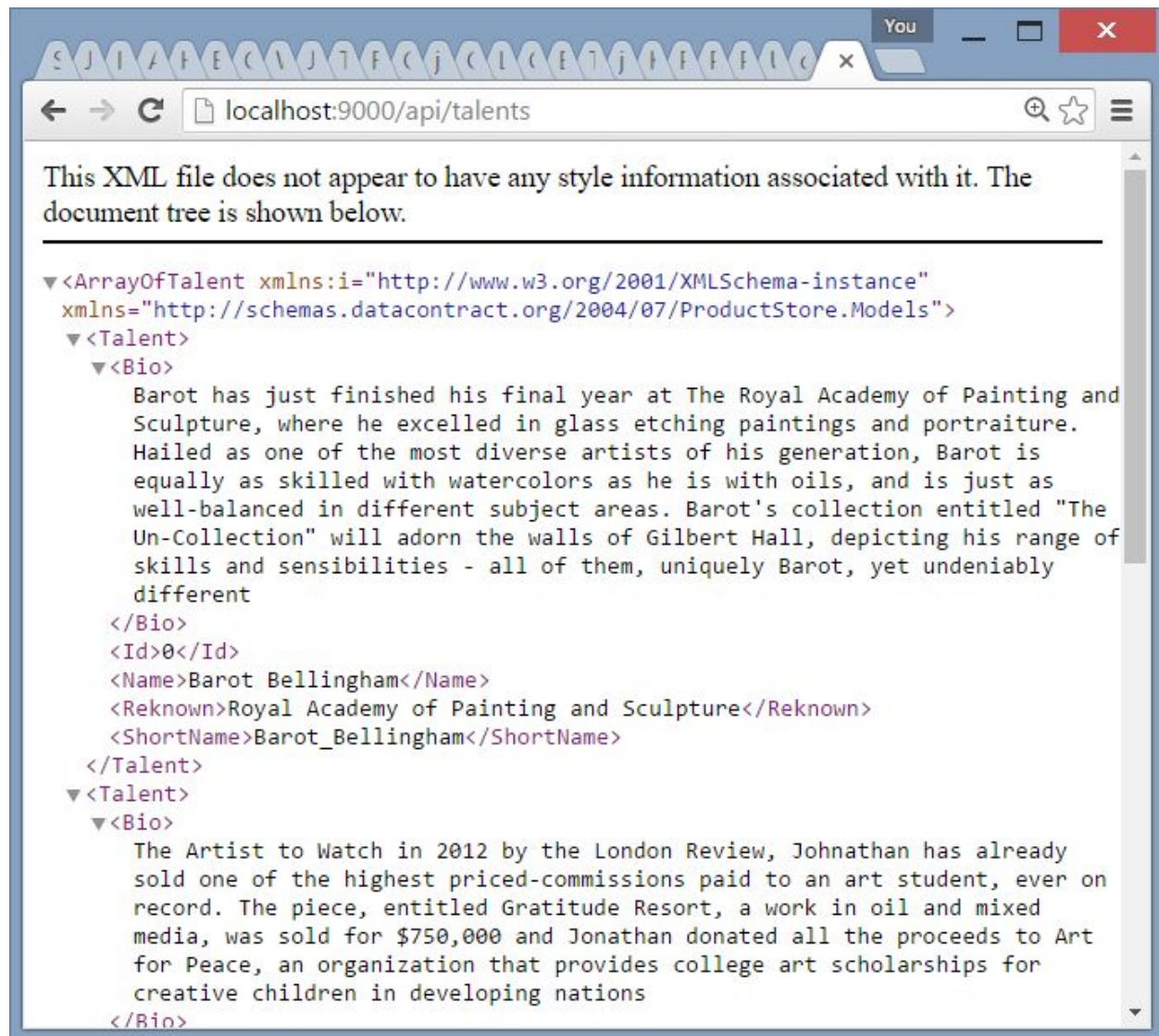
        [Route("api/talents/{id:int}")]

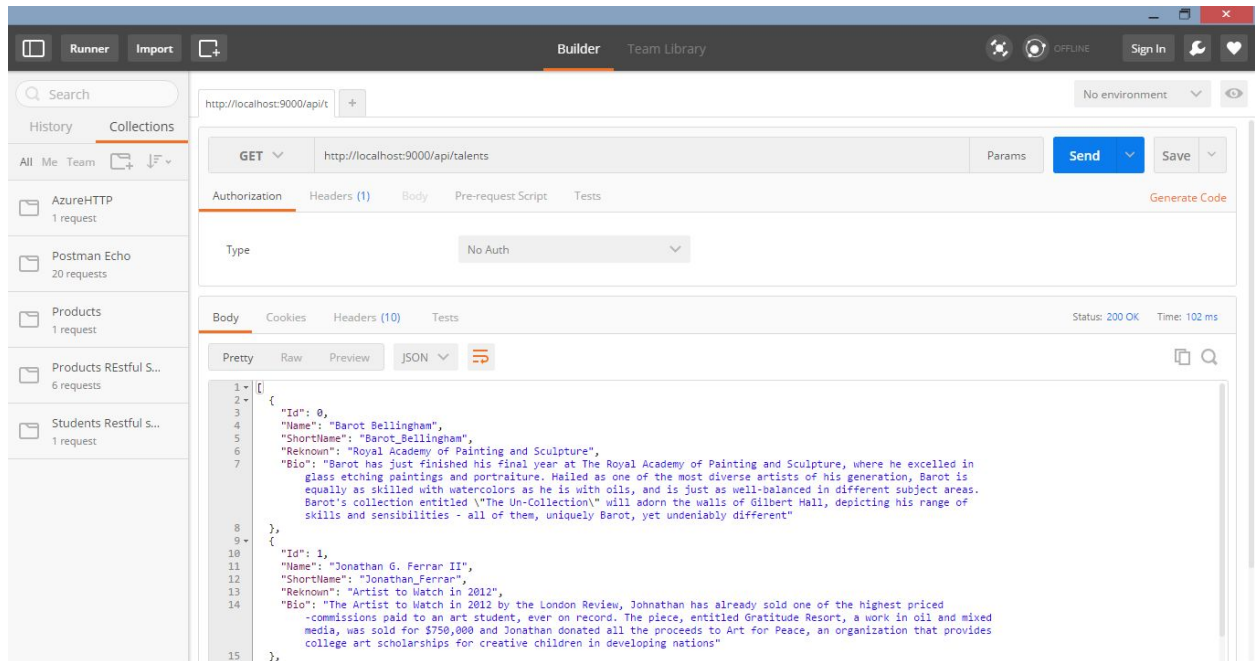
        public Talent GetTalent(int id)
        {
            Talent item = repository.Get(id);

            if (item == null)
            {
                throw new HttpResponseException(HttpStatusCode.NotFound);
            }

            return item;
        }
    }
}
```

Testing your Restful Service





E

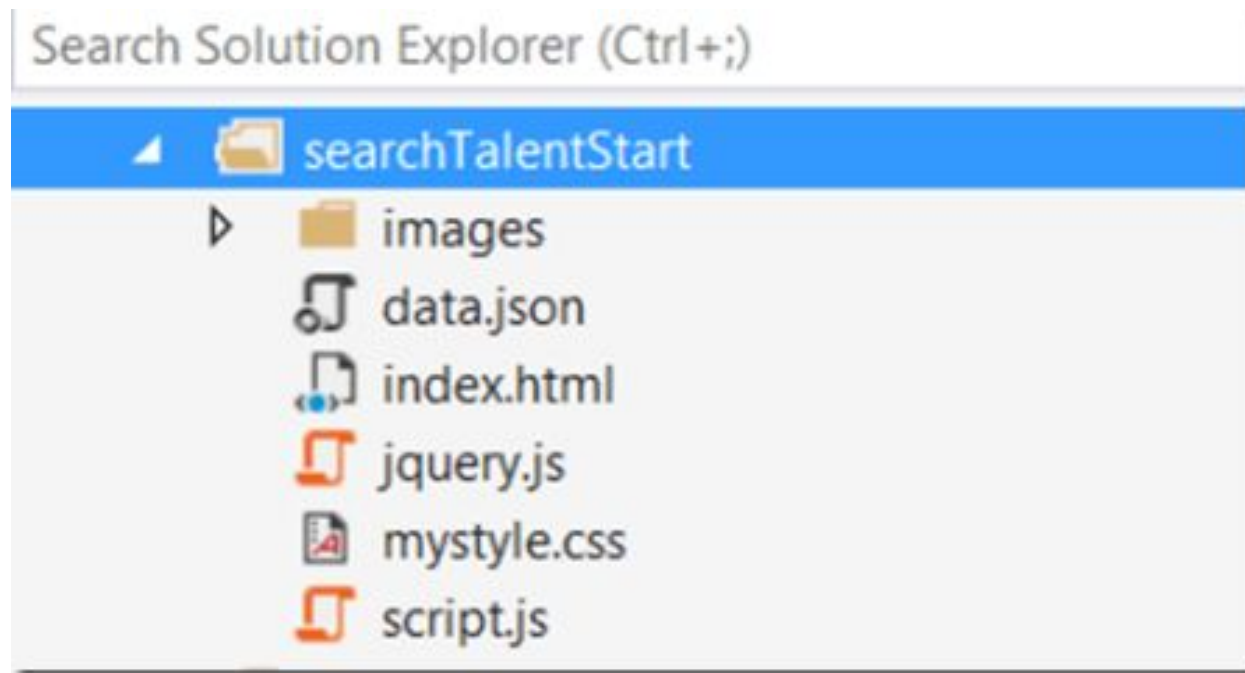
(Optional+additional features)

Change the below client code to consume the restful service at the above URL.

Download the [zip file](#) for a talent search. Create a folder in your web application project:

1. Create a folder “SearchTalent” in the above created Web Application

2. Copy all the files in the Zip file to the folder. Your project structure looks like this.



3. View the index.html in your browser.
4. When you type something in the textbox, searching gets started.

Question: Which event has been triggered when user typing the keywords in the textbox?



Test search by name function is working. Modify the code so that it can search either by **name** or **bio** .

Hint:

```
if ((val.Name.search(myExp) != -1) ||
```

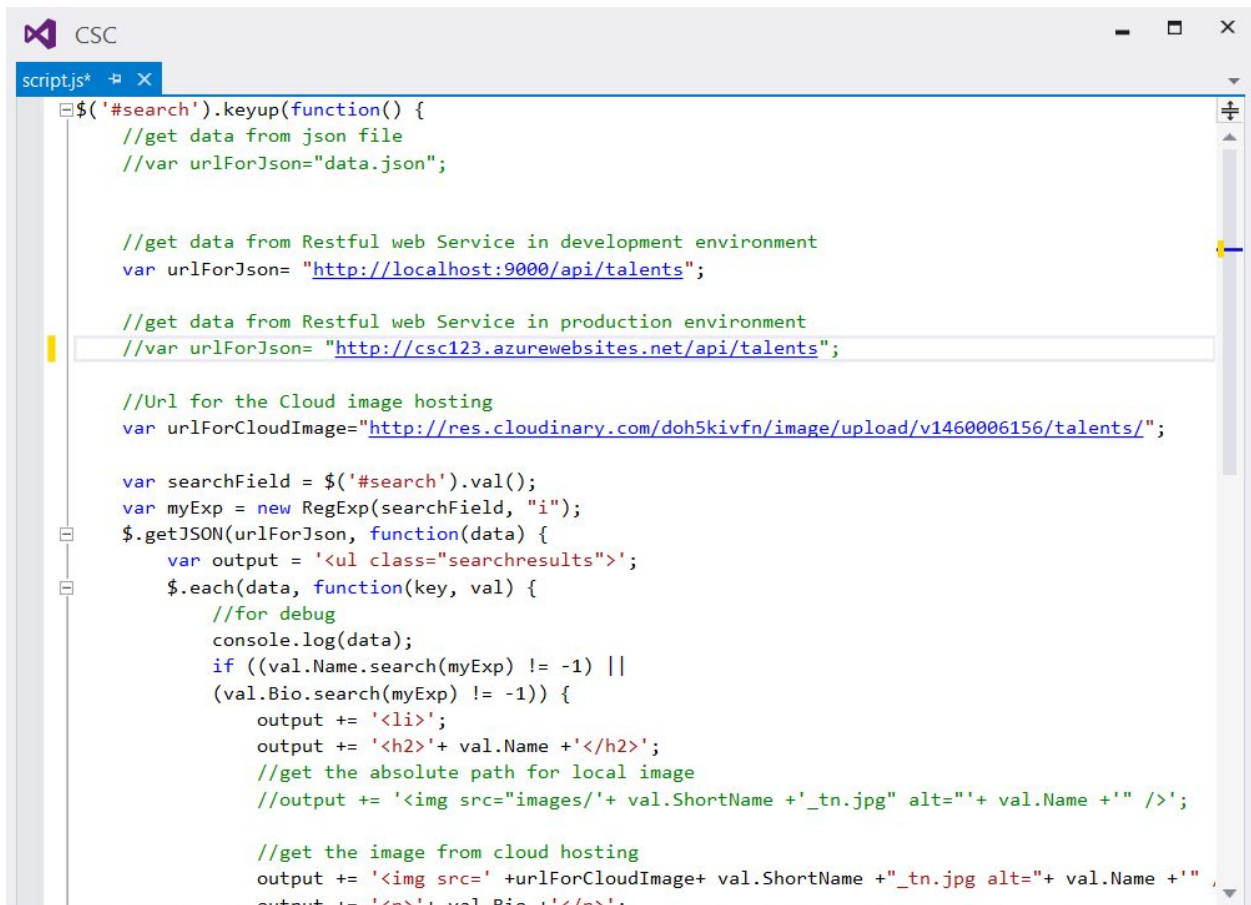
```
(val.Bio.search(myExp) != -1))
```

To get a better picture of how it works, please watch the video;

(1) login to Lynda through SP: <http://eliser.lib.sp.edu.sg/lynda>

(2) Video Link:

<http://www.lynda.com/Ajax-tutorials/Preparing-live-search-AJAX-app/114900/120878-4.html>



```
script.js*  X
$( '#search' ).keyup(function() {
    //get data from json file
    //var urlForJson="data.json";

    //get data from Restful web Service in development environment
    var urlForJson= "http://localhost:9000/api/talents";

    //get data from Restful web Service in production environment
    //var urlForJson= "http://csc123.azurewebsites.net/api/talents";

    //Url for the Cloud image hosting
    var urlForCloudImage="http://res.cloudinary.com/doh5kivfn/image/upload/v1460006156/talents/";

    var searchField = $('#search').val();
    var myExp = new RegExp(searchField, "i");
    $.getJSON(urlForJson, function(data) {
        var output = '<ul class="searchresults">';
        $.each(data, function(key, val) {
            //for debug
            console.log(data);
            if ((val.Name.search(myExp) != -1) ||
                (val.Bio.search(myExp) != -1)) {
                output += '<li>';
                output += '<h2>'+ val.Name +'</h2>';
                //get the absolute path for local image
                //output += '';

                //get the image from cloud hosting
                output += '<img src=' + urlForCloudImage+ val.ShortName +"_tn.jpg alt="+ val.Name +">';
                output += '</li>';
            }
        });
        output += '</ul>';
    });
    $('#results').html(output);
});
```

To **OFFLOAD** your WEb Application, Talents pictures have been uploaded to Cloudinary in the above sample code.

{“task”: “Challenging”}

Publish your Restful Web Service to Azure and consuming

the Restful Service hosted at Azure at your client application.

(To be implemented in Assignment 2

Creating a Resource

Updating a Resource

Deleting a Resource)