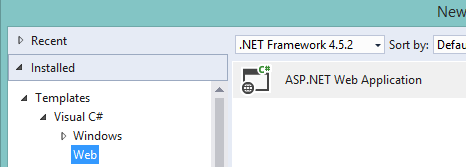
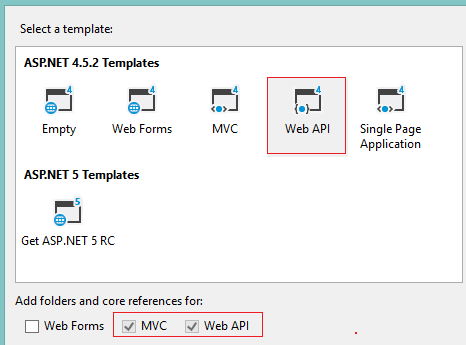
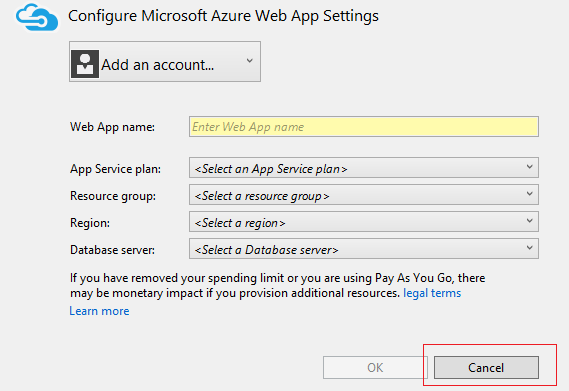
File -> Project -> C# -> ASP.NET Web Application



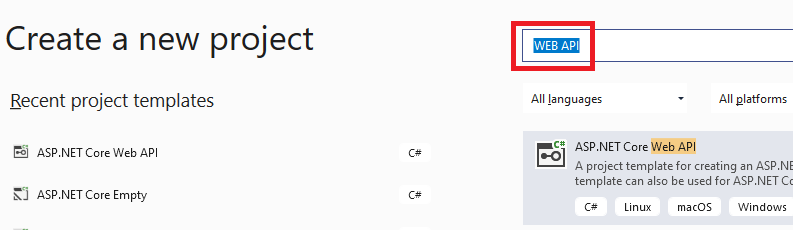


Not using Azure, cancel



No escoja Azure, dele Cancel

**VISUAL STUDIO 2019**



Cree modelo Product

* Right-click Model folder
* Add
* Class
* Name Product

namespace ProductsApp.Models

{

public class Product

{

public int ID { get; set; }

public string Name { get; set; }

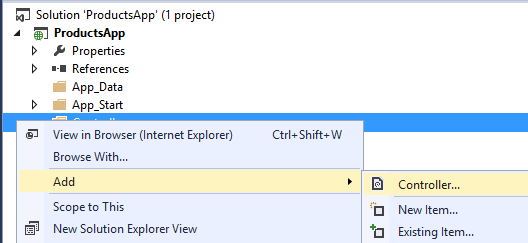
public string Category { get; set; }

public decimal Price { get; set; }

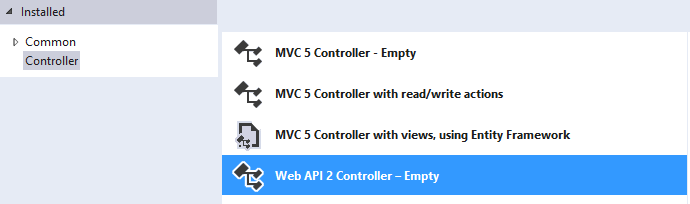
}

}

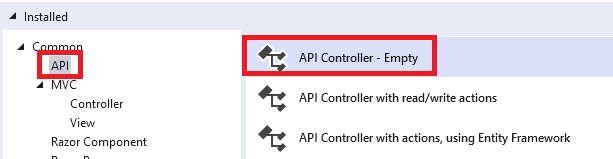
In **Solution Explorer**, right-click the Controllers folder. Select **Add** and then select **Controller**.



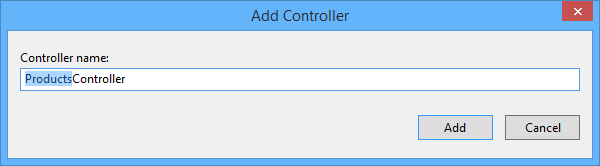
In the **Add Scaffold** dialog, select **Web API Controller - Empty**. Click **Add**.



VISUAL STUDIO 2019



In the **Add Controller** dialog, name the controller "ProductsController". Click **Add**.



You don't need to put your contollers into a folder named Controllers. The folder name is just a convenient way to organize your source files.

If this file is not open already, double-click the file to open it. Replace the code in this file with the following:

using System.Collections.Generic;

using System.Linq;

using ProductsApp.Models;

using System.Web.Http;

namespace ProductsApp.Controllers

{

public class ProductsController : ApiController

{

Product[] products = new Product[]

{

new Product { Id = 1, Name = "Tomato Soup", Category = "Groceries", Price = 1 },

new Product { Id = 2, Name = "Yo-yo", Category = "Toys", Price = 3.75M },

new Product { Id = 3, Name = "Hammer", Category = "Hardware", Price = 16.99M }

};

public IEnumerable<Product> GetAllProducts()

{

return products;

}

public IHttpActionResult GetProduct(int id)

{

var product = products.FirstOrDefault((p) => p.Id == id);

if (product == null)

{

return NotFound();

}

return Ok(product);

}

}

}

VISUAL STUDIO 2019

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

using System.Linq;

using WebApplication6.Models;

namespace WebApplication6.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class ProductController : ControllerBase

{

Product[] products = new Product[]

{

new Product { Id = 1, Name = "Tomato Soup", Category = "Groceries", Price = 1 },

new Product { Id = 2, Name = "Yo-yo", Category = "Toys", Price = 3.75M },

new Product { Id = 3, Name = "Hammer", Category = "Hardware", Price = 16.99M }

};

public IEnumerable<Product> GetAllProducts()

{

return products;

}

public IHttpActionResult GetProduct(int id)

{

var product = products.FirstOrDefault((p) => p.Id == id);

if (product == null)

{

return NotFound();

}

return Ok(product);

}

}

}

Archive launchsettings

"profiles": {

"IIS Express": {

"commandName": "IISExpress",

"launchBrowser": true,

"launchUrl": "weatherforecast",

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

}

},

"WebApplication6": {

"commandName": "Project",

"launchBrowser": true,

"launchUrl": "weatherforecast",

"applicationUrl": "https://localhost:5001;http://localhost:5000",

"environmentVariables": {

"ASPNETCORE\_ENVIRONMENT": "Development"

}

}

To keep the example simple, products are stored in a fixed array inside the controller class. Of course, in a real application, you would query a database or use some other external data source.

The controller defines two methods that return products::

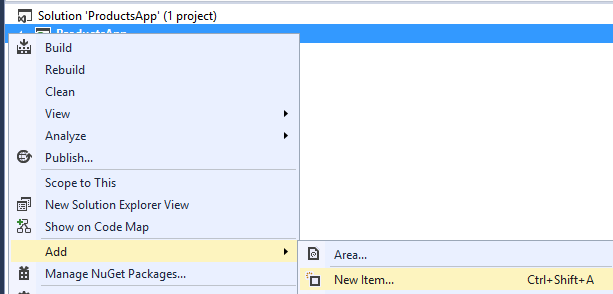
|  |  |  |  |
| --- | --- | --- | --- |
| **Controller method** | **Purpose** | **URI** | **Sample** |
| GetAllProducts | List all products | /api/products | /api/products |
| GetProduct | List ONE product | /api/products/id | /api/products/5 |

For more information about how Web API routes HTTP requests to controller methods, see [Routing in ASP.NET Web API](https://www.asp.net/web-api/overview/web-api-routing-and-actions/routing-in-aspnet-web-api).

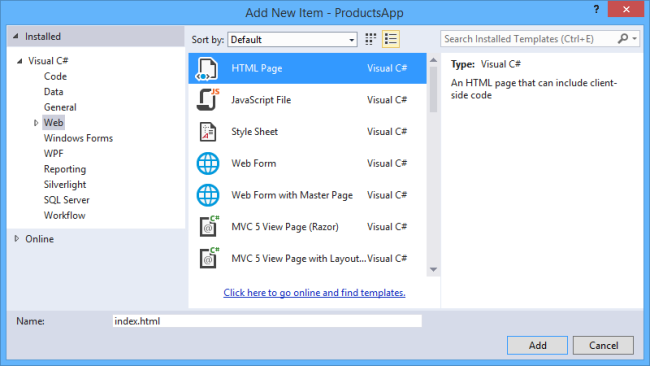
Calling the Web API with Javascript and jQuery

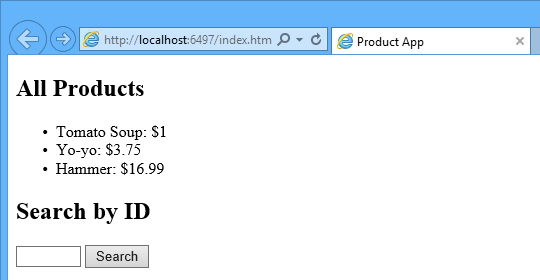
In this section, we'll add an HTML page that uses AJAX to call the web API. We'll use jQuery to make the AJAX calls and also to update the page with the results.

In Solution Explorer, right-click the project and select **Add**, then select **New Item**.



In the **Add New Item** dialog, select the **Web** node under **Visual C#**, and then select the **HTML Page** item. Name the page "index.html".





Replace everything in this file with the following:

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>Product App</title>

</head>

<body>

<div>

**<h2>All** Products</h2>

<ul id="products" />

</div>

<div>

**<h2>Search by ID</h2>**

<input type="text" id="prodId" size="5" />

<input type="button" value="Search" onclick="find();" />

<p id="product" />

</div>

<script src="http://ajax.aspnetcdn.com/ajax/jQuery/jquery-2.0.3.min.js"></script>

<script>

// CONTROLADOR A LLAMAR

var uri = 'api/products';

$(document).ready(function () {

// LIST ALL PRODUCTS, SEND AJAX REQUEST api/products

// ====================================

$.getJSON(uri)

.done(function (data) {

// On success, 'data' contains a list of products.

$.each(data, function (key, item) {

$('<li>', { text: formatItem(item) }).appendTo($('#products'));

});

});

});

function formatItem(item) {

return item.Name + ': $' + item.Price;

}

// LIST ONE PRODUCT, SEND AJAX REQUEST api/products/#

// ====================================

function find() {

var id = $('#prodId').val();

$.getJSON(uri + '/' + id)

.done(function (data) {

$('#product').text(formatItem(data));

})

.fail(function (jqXHR, textStatus, err) {

$('#product').text('Error: ' + err);

});

}

</script>

</body>

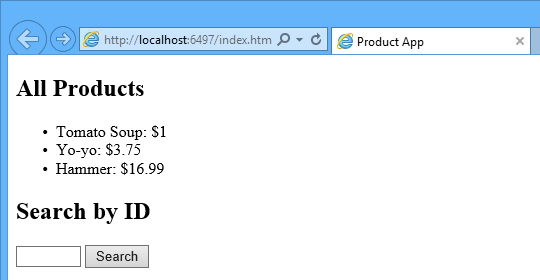
</html>

There are several ways to get jQuery. In this example, I used the [Microsoft Ajax CDN](https://www.asp.net/ajaxlibrary/cdn.ashx). You can also download it from <http://jquery.com/>, and the ASP.NET "Web API" project template includes jQuery as well.

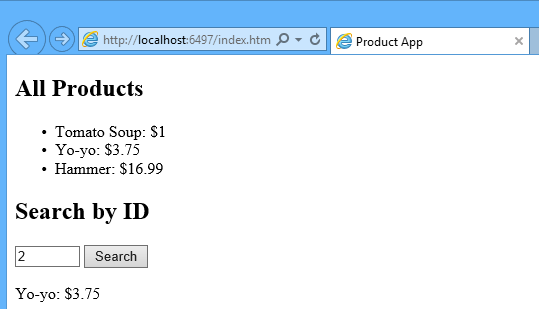
Running the Application

http://localhost:32079/Test.html

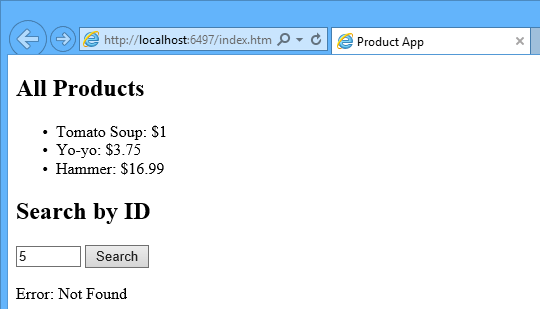
Press F5 to start debugging the application. The web page should look like the following:



To get a product by ID, enter the ID and click Search:

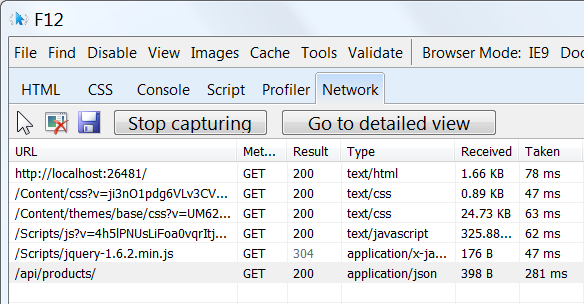


If you enter an invalid ID, the server returns an HTTP error:

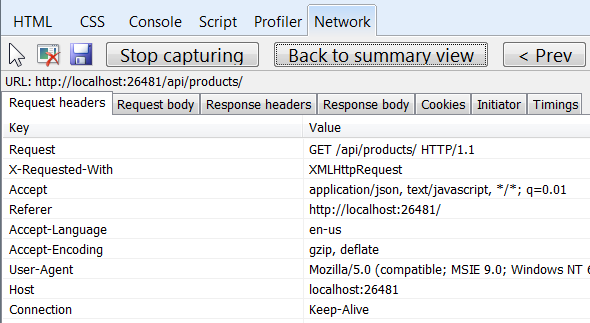


Using F12 to View the HTTP Request and Response

When you are working with an HTTP service, it can be very useful to see the HTTP request and request messages. You can do this by using the F12 developer tools in Internet Explorer 9. From Internet Explorer 9, press **F12** to open the tools. Click the **Network** tab and press **Start Capturing**. Now go back to the web page and press **F5** to reload the web page. Internet Explorer will capture the HTTP traffic between the browser and the web server. The summary view shows all the network traffic for a page:



Locate the entry for the relative URI “api/products/”. Select this entry and click **Go to detailed view**. In the detail view, there are tabs to view the request and response headers and bodies. For example, if you click the **Request headers** tab, you can see that the client requested "application/json" in the Accept header.



If you click the Response body tab, you can see how the product list was serialized to JSON. Other browsers have similar functionality. Another useful tool is [Fiddler](http://www.fiddler2.com/fiddler2/), a web debugging proxy. You can use Fiddler to view your HTTP traffic, and also to compose HTTP requests, which gives you full control over the HTTP headers in the request.

See this App Running on Azure

Would you like to see the finished site running as a live web app? You can deploy a complete version of the app to your Azure account by simply clicking the following button.

[http://azuredeploy.net/deploybutton.png](https://deploy.azure.com/?WT.mc_id=deploy_azure_aspnet&repository=https://github.com/tfitzmac/WebAPI-ProductsApp#/form/setup)

You need an Azure account to deploy this solution to Azure. If you do not already have an account, you have the following options:

* [Open an Azure account for free](http://azure.microsoft.com/en-us/pricing/free-trial/?WT.mc_id=A443DD604) - You get credits you can use to try out paid Azure services, and even after they're used up you can keep the account and use free Azure services.
* [Activate MSDN subscriber benefits](http://azure.microsoft.com/en-us/pricing/member-offers/msdn-benefits-details/?WT.mc_id=A443DD604) - Your MSDN subscription gives you credits every month that you can use for paid Azure services.

Next Steps

* For a more complete example of an HTTP service that supports POST, PUT, and DELETE actions and writes to a database, see [Using Web API 2 with Entity Framework 6](https://www.asp.net/web-api/overview/data/using-web-api-with-entity-framework/part-1).
* For more about creating fluid and responsive web applications on top of an HTTP service, see [ASP.NET Single Page Application](https://www.asp.net/single-page-application).
* For information about how to deploy a Visual Studio web project to Azure App Service, see [Create an ASP.NET web app in Azure App Service](http://azure.microsoft.com/en-us/documentation/articles/web-sites-dotnet-get-started/).