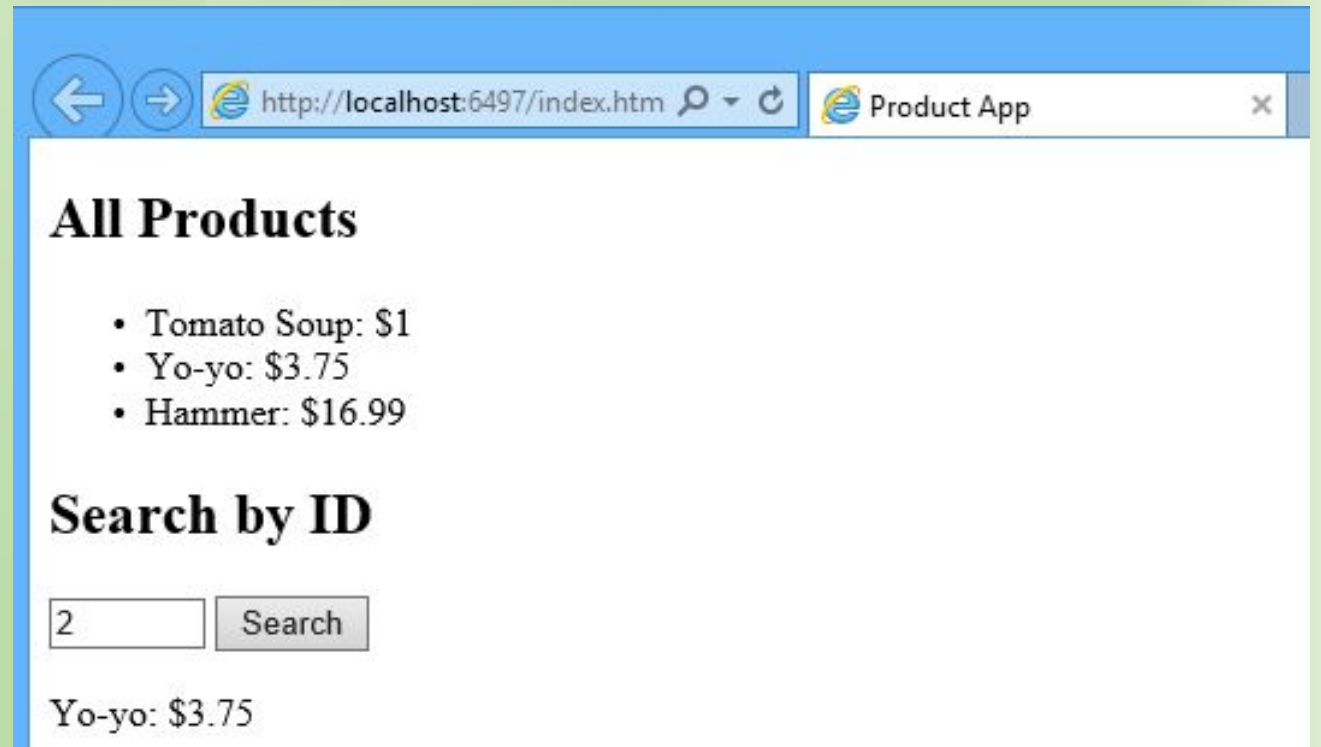
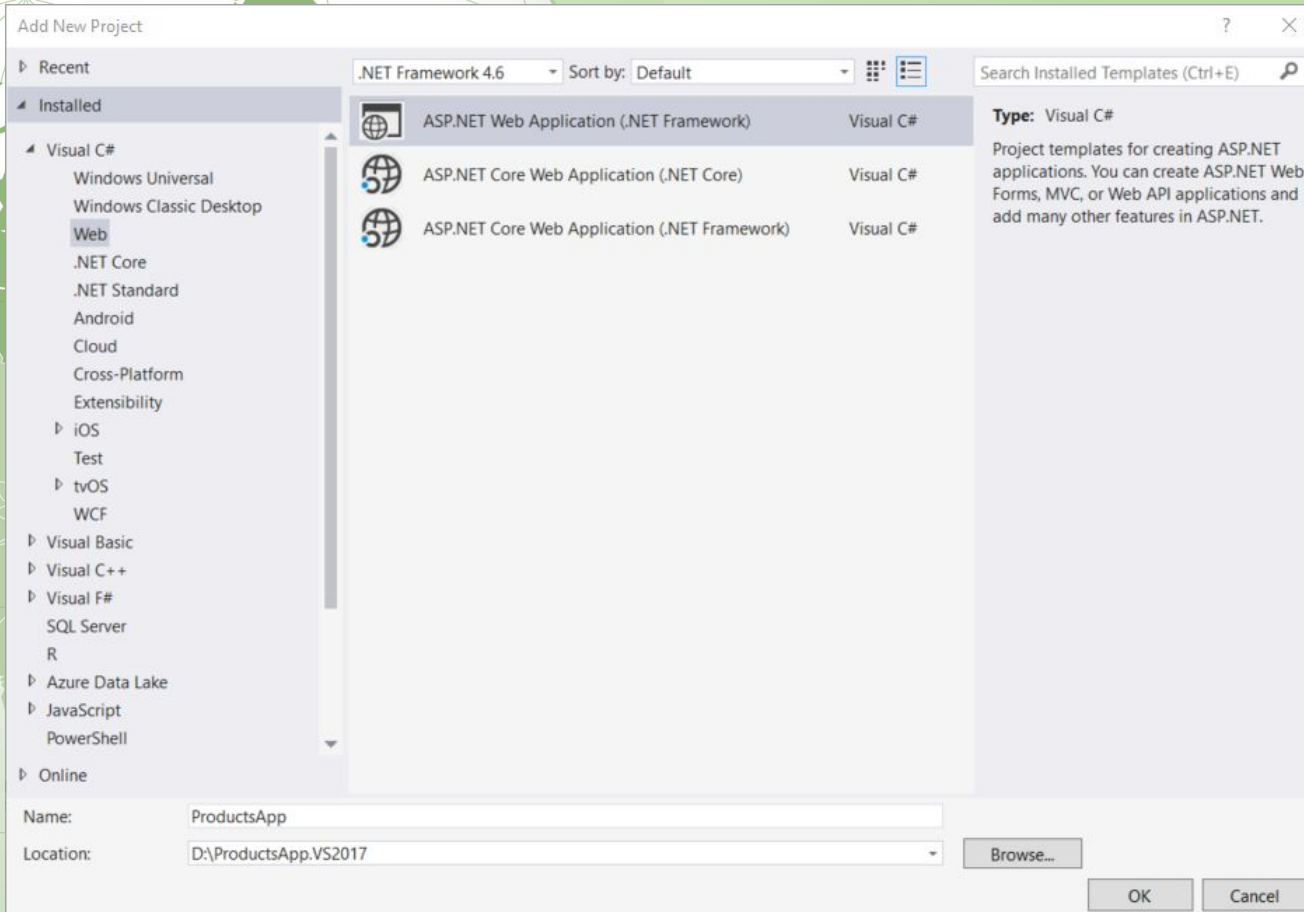


In this tutorial, you will use ASP.NET Web API to create a web API that returns a list of products. The front-end web page uses jQuery to display the results.



- 
- *Start 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 Web Application**. Name the project "ProductsApp" and click **OK**.*



- 
- In the **New ASP.NET Project** dialog, select the **Empty** template. Under "Add folders and core references for", check **Web API**. Click **OK**.

New ASP.NET Web Application - ProductsApp

ASP.NET 4.6 Templates

Empty Web Forms MVC Web API Single Page Application

Azure API App Azure Mobile App

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

[Learn more](#)

Change Authentication

Authentication: **No Authentication**

Add folders and core references for:

Web Forms MVC Web API

Add unit tests

Test project name: ProductsApp.Tests


OK Cancel










Adding a Model

- *A model is an object that represents the data in your application. ASP.NET Web API can automatically serialize your model to JSON, XML, or some other format, and then write the serialized data into the body of the HTTP response message. As long as a client can read the serialization format, it can deserialize the object. Most clients can parse either XML or JSON. Moreover, the client can indicate which format it wants by setting the Accept header in the HTTP request message.*
- *Let's start by creating a simple model that represents a product.*
- *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**.*





ation and extend it to the cloud.


 Add a service


-  Controller...
-  New Item... Ctrl+Shift+A
-  Existing Item... Shift+Alt+A
-  New Scaffolded Item...
-  New Folder
-  Add ASP.NET Folder
-  From Cookiecutter...
-  Docker Support
- REST API Client...
- New Azure WebJob Project
- Existing Project as Azure WebJob
-  Class...


Deploy





 View in Browser (Microsoft Edge) Ctrl+Shift+W


 Browse With...


 Add


 Scope to This


 New Solution Explorer View


 Show on Code Map


 Exclude From Project


 Cut Ctrl+X


 Copy Ctrl+C

 Paste Ctrl+V

 Delete Del

 Rename

 Open Folder in File Explorer

 Properties Alt+Enter

Solution Explorer

Search Solution Explorer (Ctrl+;)

Solution 'ProductsApp.VS2017' (1 project)

- ProductsApp**
 - Connected Services
 - Properties
 - References
 - App_Data
 - App_Start
 - Controllers
 - Models
 - Global.asax
 - packages.config
 - WebResource.config

Team Explorer

Properties

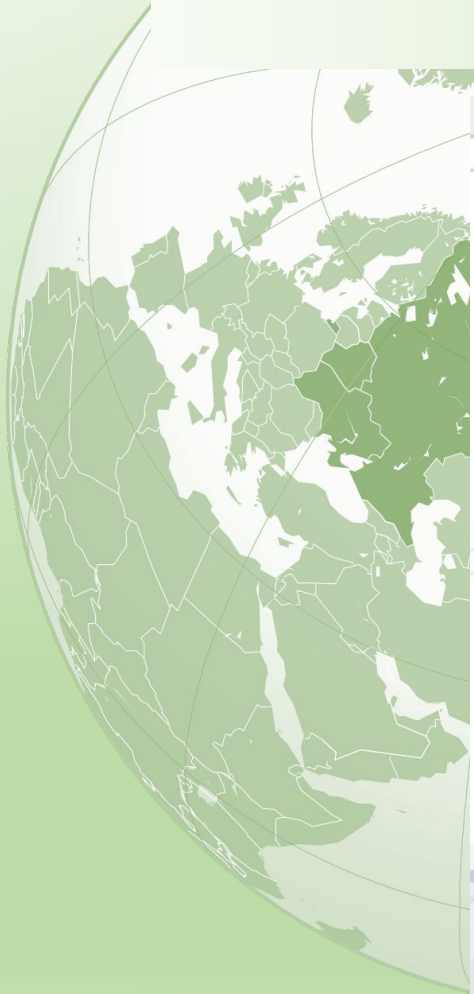
Models

Name the class "Product". Add the following properties to the Product class.

- *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; }*
- *}*
- *}*

Adding a Controller

- *In Web API, a controller is an object that handles HTTP requests. We'll add a controller that can return either a list of products or a single product specified by ID.*
- *In **Solution Explorer**, right-click the **Controllers** folder. Select **Add** and then select **Controller**.*



The screenshot shows the Visual Studio interface with the Solution Explorer on the right. The 'Controllers' folder is selected, and a context menu is open over it. The menu items are:

- Controller...
- New Item... (Ctrl+Shift+A)
- Existing Item... (Shift+Alt+A)
- New Scaffolded Item...
- New Folder
- Add ASP.NET Folder (▶)
- From Cookiecutter...
- Docker Support
- REST API Client...
- New Azure WebJob Project
- Existing Project as Azure WebJob
- Web API Controller Class (v2.1)
- Class...

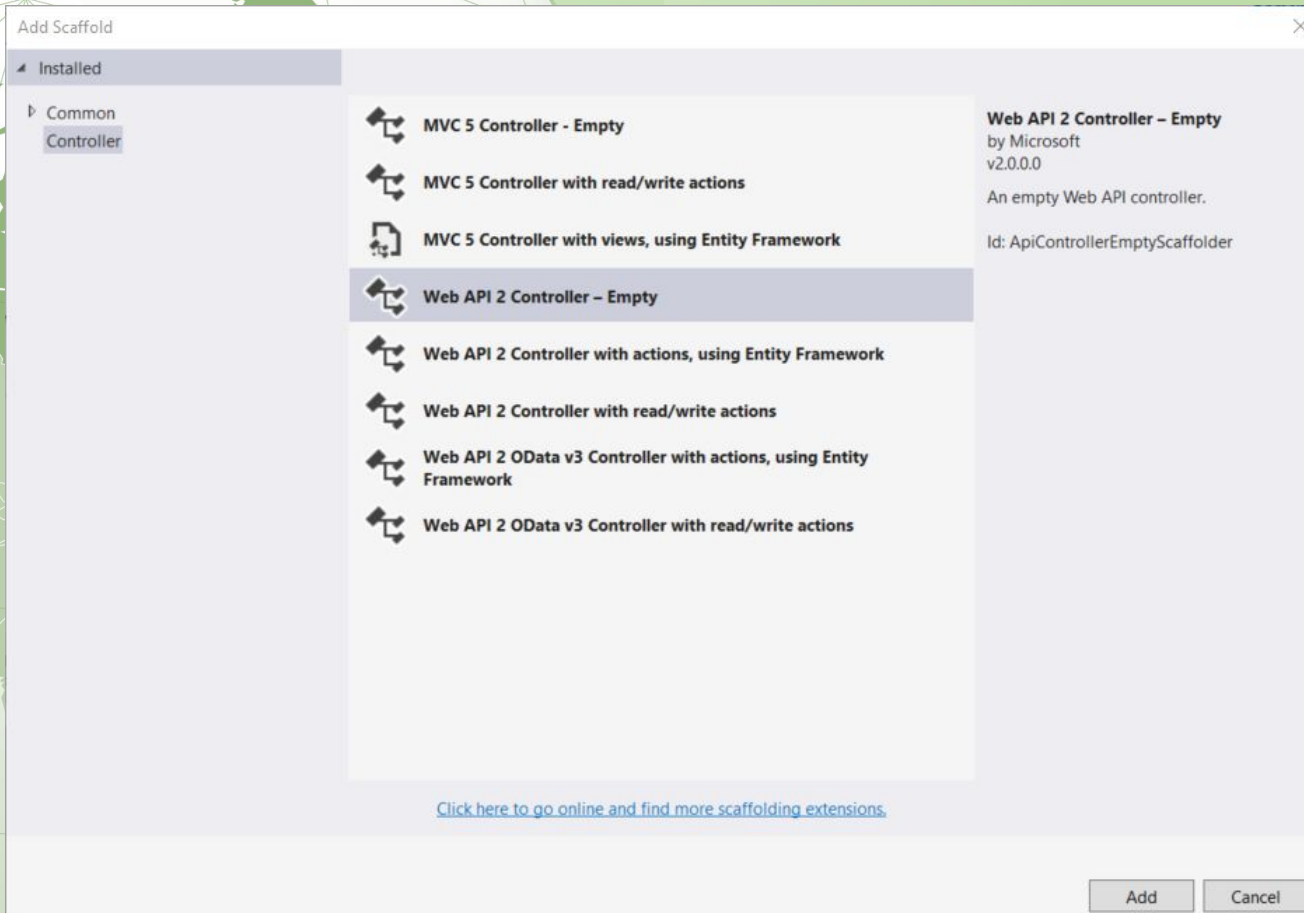
The 'Add' option is expanded, showing a sub-menu with the following items:

- View in Browser (Microsoft Edge) (Ctrl+Shift+W)
- Browse With...
- Add
- Scope to This
- New Solution Explorer View
- Show on Code Map
- Exclude From Project
- Cut (Ctrl+X)
- Copy (Ctrl+C)
- Paste (Ctrl+V)
- Delete (Del)
- Rename
- Open Folder in File Explorer
- Properties (Alt+Enter)

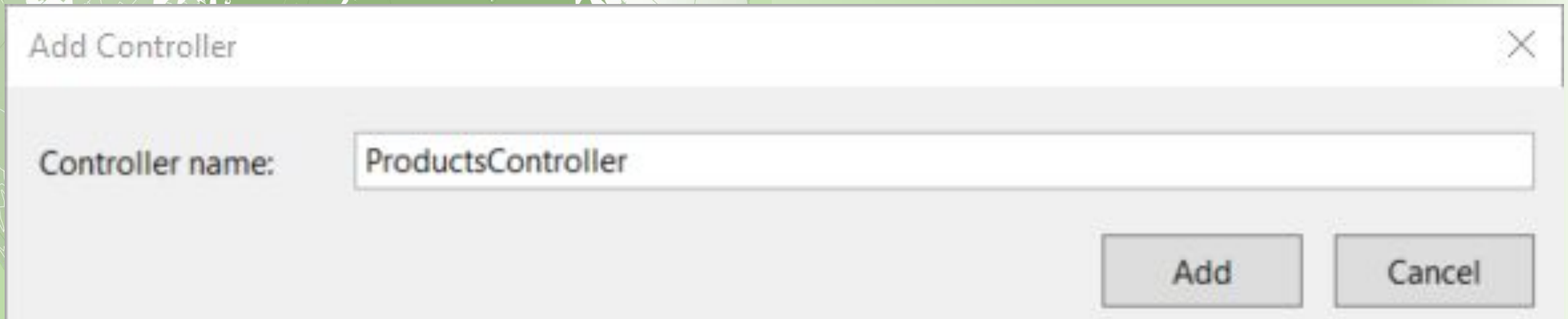
The Solution Explorer on the right shows the project structure for 'ProductsApp.VS2017' (1 project):

- ProductsApp
 - Connected Services
 - Properties
 - References
 - App_Data
 - App_Start
 - Controllers (selected)
 - Models
 - Product.cs
 - Global.asax
 - packages.config
 - WebResource.config

In the Add Scaffold dialog, select Web API Controller - Empty. Click Add. (лучше – with r/w action)

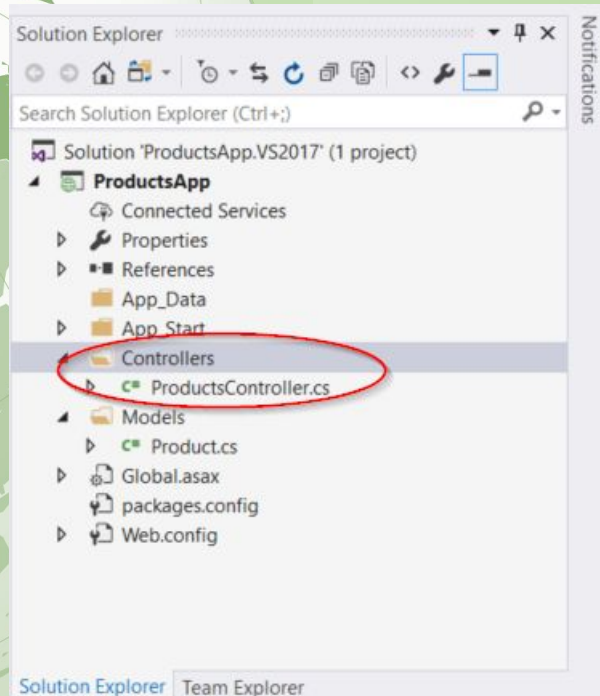


In the Add Controller dialog, name the controller "ProductsController". Click Add.



The image shows a screenshot of a software dialog box titled "Add Controller". The dialog has a title bar with the text "Add Controller" and a close button (an 'X' icon) on the right. Below the title bar, there is a label "Controller name:" followed by a text input field containing the text "ProductsController". At the bottom right of the dialog, there are two buttons: "Add" and "Cancel". The background of the slide features a faint, stylized globe.

The scaffolding creates a file named **ProductsController.cs** in the **Controllers** folder.



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

```
• 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)
•     {
```



• Вы делаете:

• 1 – тестовый массив данных

• 2 – метод, который ищет элемент массива по его Id

• 3 – метод, который находит все элементы массива

• Далее эту информацию может запросить кто угодно (при условии, что она развернута на сервере и сервер корректно работает)



- *That's it! You have a working web API. Each method on the controller corresponds to one or more URIs:*

- *Controller Method URI*

- *GetAllProducts /api/products*

- *GetProduct /api/products/id*

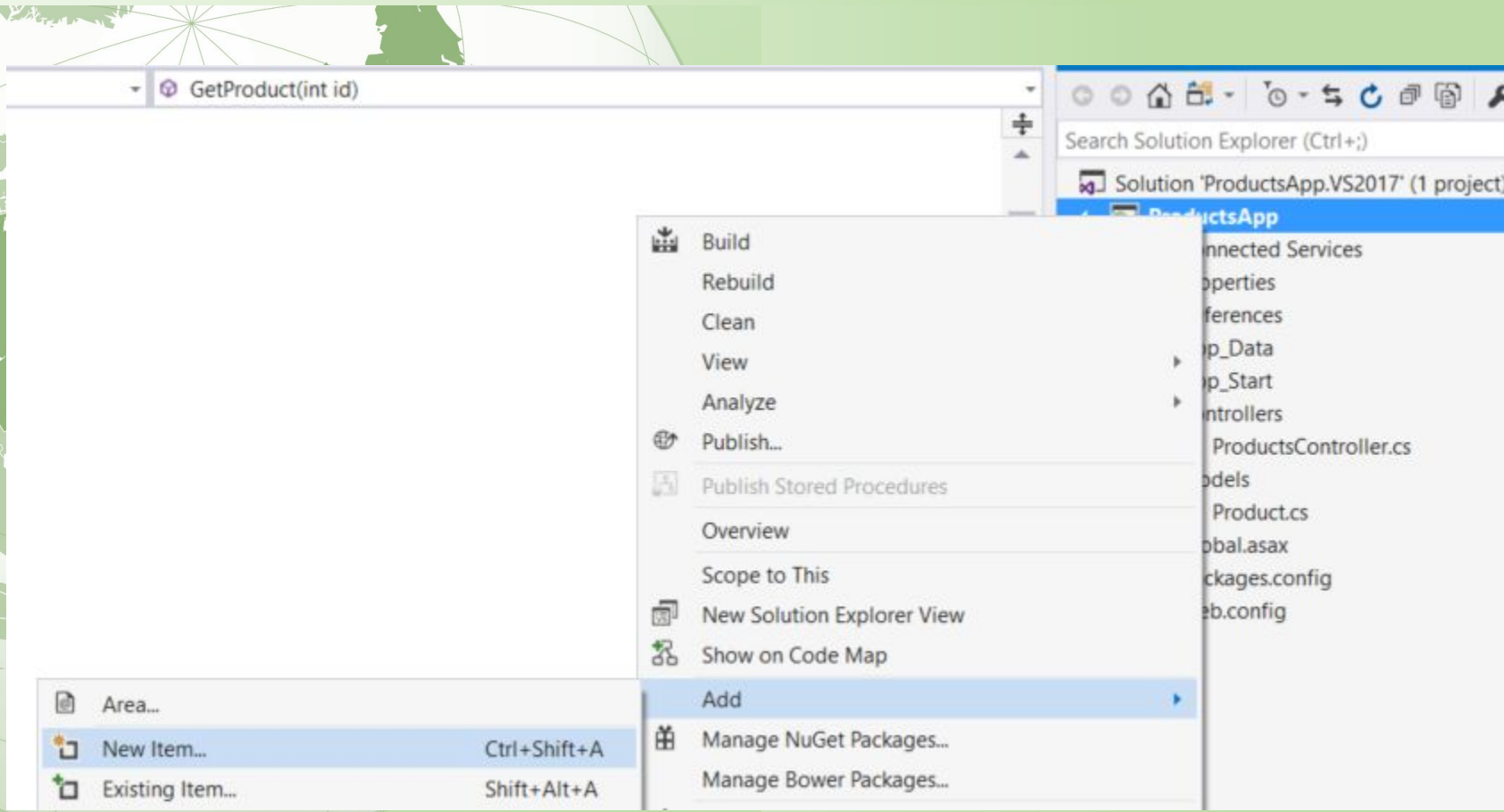
- *For the GetProduct method, the id in the URI is a placeholder. For example, to get the product with ID of 5, the URI is api/products/5.*

Calling the Web API with Javascript and jQuery

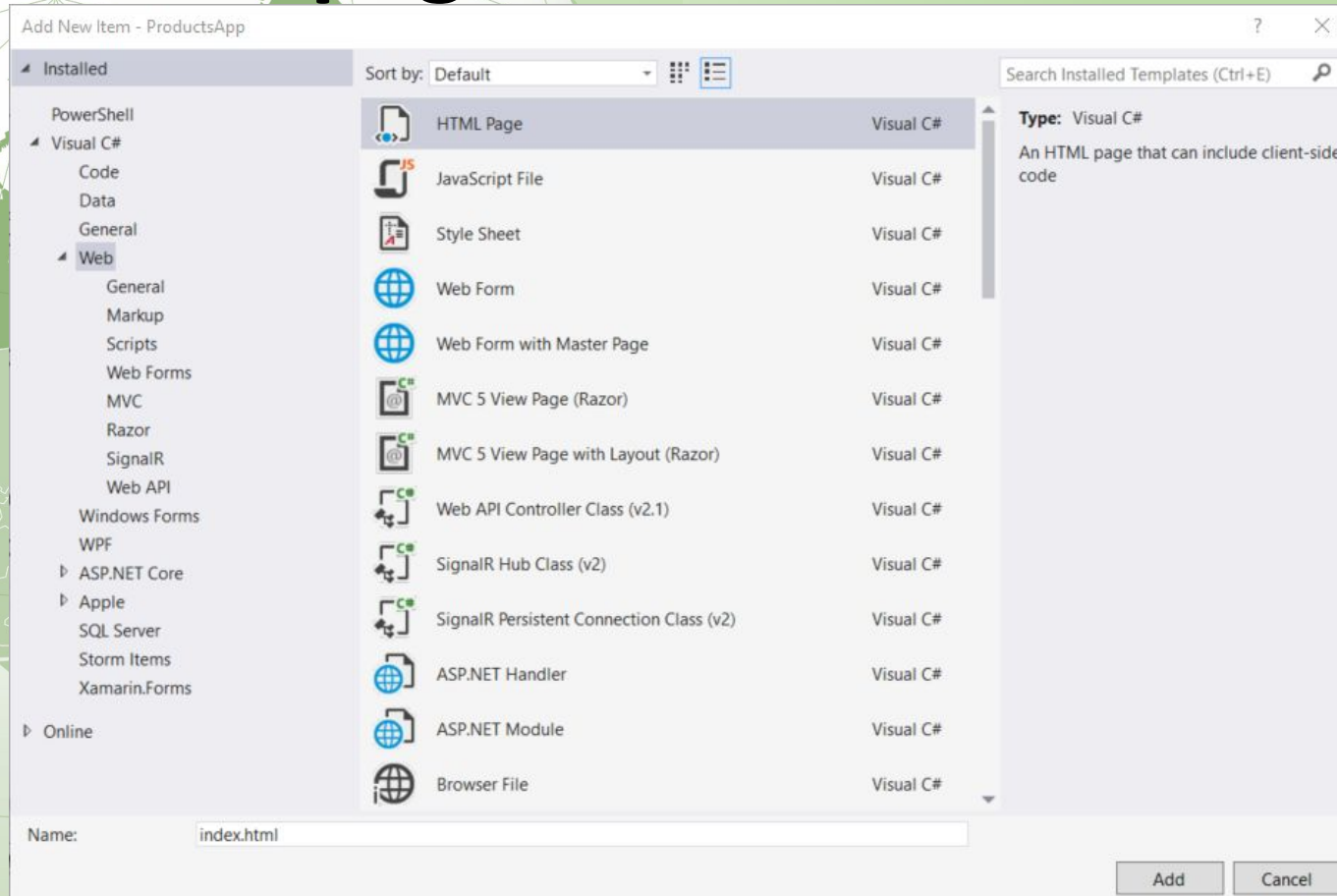
- *In this section, we'll add an HTML page that uses AJAX to call the web API. We'll use jQuery to make the AJAX calls and also to update the page with the results.*



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".

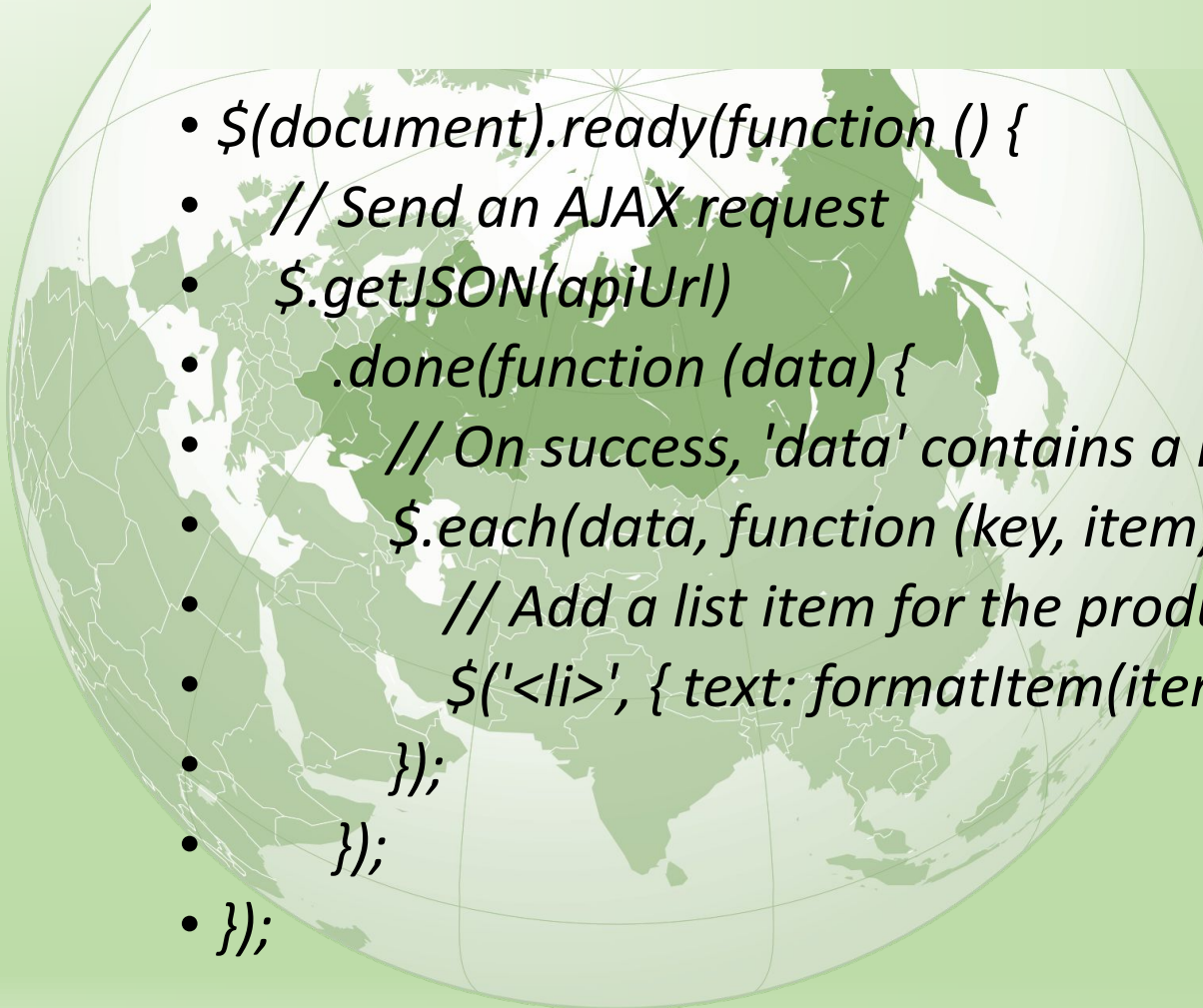


Полный код по ссылке

- <https://github.com/aspnet/Docs/blob/master/aspnet/web-api/overview/getting-started-with-aspnet-web-api/tutorial-your-first-web-api/samples/sample3.html>

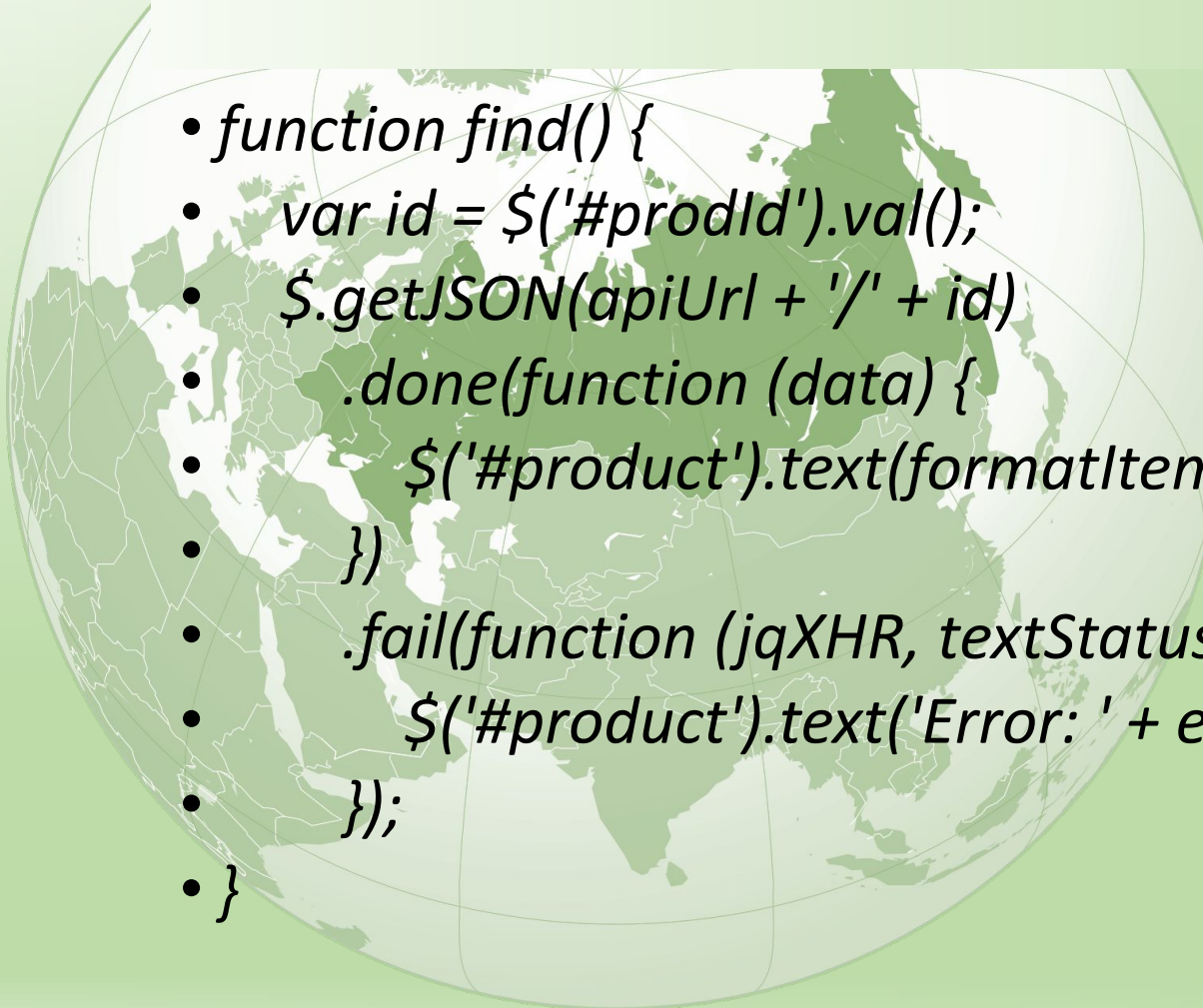


Получение всего списка данных



- `$(document).ready(function () {`
- `// Send an AJAX request`
- `$.getJSON(apiUrl)`
- `.done(function (data) {`
- `// On success, 'data' contains a list of products.`
- `$.each(data, function (key, item) {`
- `// Add a list item for the product.`
- `$('#', { text: formatItem(item) }).appendTo($('#products'));`
- `});`
- `});`
- `});`

Поиск по Id



```
• function find() {  
•   var id = $('#prodId').val();  
•   $.getJSON(apiUrl + '/' + id)  
•     .done(function (data) {  
•       $('#product').text(formatItem(data));  
•     })  
•     .fail(function (jqXHR, textStatus, err) {  
•       $('#product').text('Error: ' + err);  
•     });  
• }
```


Основное

- *Страница обращается к сервису по адресу ...*
- *Функция поиска запрашивает по этому адресу данные `getJSON`*
- *Эти данные приходят по протоколу TCP/IP*
- *Собранные данные трактуются как данные в формате `JSON`*
- *«Распознанные» данные выводятся на страницу*

Running the Application

- *Press F5 to start debugging the application.*



Using F12 to View the HTTP Request and Response

- *Запустите в браузере средства разработчика клавишей F12*

