Practical 2
Web Client and Server interaction using HTTP Request and Response

Objectives

Explore the existing Web service

You will learn to:

Part 1

- Understand the Request/Response interaction between client and server using HTTP
- Understand the functions of a Web service
- Web Service as a building block

Part 2

- Create a WEB API 2 Restful Service

.

Part 1

Tasks

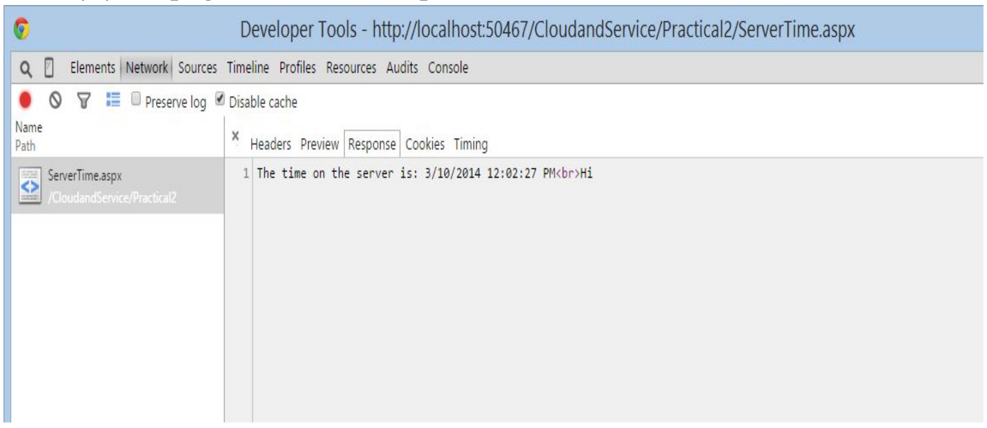
A. Create a ASPX page named ServerTime.aspx in the Practical2 folder in your Visual Studio Project

1. Add a new Web form ServerTime.aspx to your project
2. Locate the page_load Event

```
 //required to keep the page from being cached on the client's browser
        //The response of a "GET" request is cached by default, so to make sure that
our example brings back the current time each time
        Response.Cache.SetExpires(DateTime.Now.AddSeconds(2));
        Response.Cache.SetCacheability(HttpCacheability.Public);
        Response.Cache.SetValidUntilExpires(true);
        //The next few lines change the content type to plain text, get the current time,
and writes the output:
        Response.ContentType = "text/plain";
        Response.Write("The time on the server is: " + DateTime.Now.ToString());
        Response.Write("<br>");
        Response.Write("Hi " + Request.QueryString["myName"]);
```

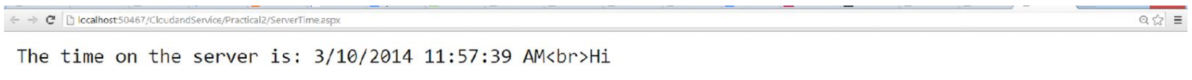
View the file in Browser.

3. Modify your page so that the output html source as follows:

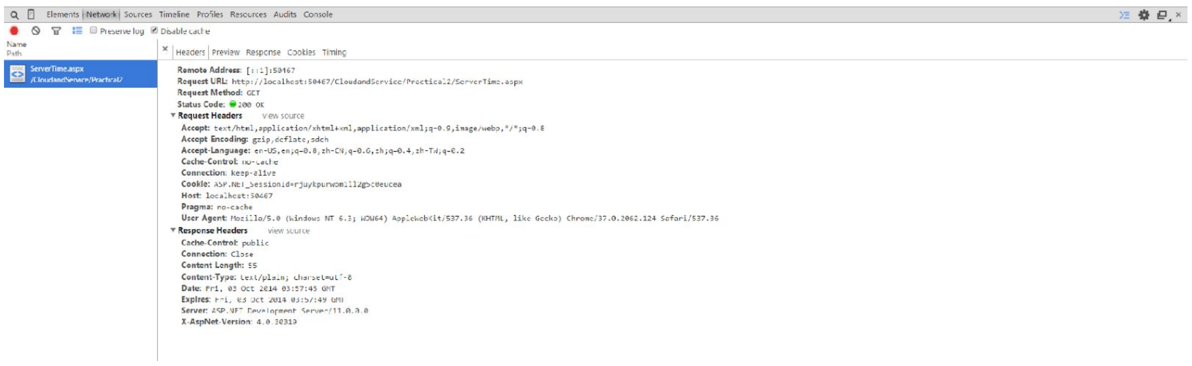

Developer Tools - http://localhost:50467/CloudandService/Practical2/ServerTime.aspx
Elements Network Sources Timeline Profiles Resources Audits Console
Preserve log
Disable cache
Name
Path
ServerTime.aspx
/CloudandService/Practical2
Headers Preview Response Cookies Timing

```
The time on the server is: 3/10/2014 12:02:27 PM<br>Hi
```

4. What is the value for the response header field " Cache-Control"



```
The time on the server is: 3/10/2014 11:57:39 AM<br>Hi
```


Elements Network Sources Timeline Profiles Resources Audits Console
Preserve log
Disable cache
Name
Path
ServerTime.aspx
/CloudandService/Practical2
Headers Preview Response Cookies Timing

```
Remote Address: [::1]:50467
Request URL: http://localhost:50467/CloudandService/Practical2/ServerTime.aspx
Request Method: GET
Status Code: 200 OK
▼ Request Headers      view source
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
  Accept Encoding: gzip,deflate,sdch
  Accept-Language: en-US,en;q=0.8,zh-CN;q=0.6,zh;q=0.4,zh-TW;q=0.2
  Cache-Control: no-cache
  Connection: keep-alive
  Cookie: ASP.NET_SessionId=rjuykpurwom1l12g5c0eucea
  Host: localhost:50467
  Pragma: no-cache
  User Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2062.124 Safari/537.36
▼ Response Headers      view source
  Cache-Control: public
  Connection: Close
  Content Length: 55
  Content-Type: text/plain; charset=utf-8
  Date: Fri, 03 Oct 2014 03:57:45 GMT
  Expires: Fri, 03 Oct 2014 03:57:49 GMT
  Server: ASP.NET Development Server/11.0.0.0
  X-AspNet-Version: 4.0.30319
```

5. Add a HTML page named Exercise1.html.

6. Design a HTML form to allow user to enter the Name. Save the file as Exercise2.html

```
   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
</head>
<body>
    HTTP Request

    <form action="ServerTime.aspx" method="GET">
        <table>
            <tr>
                <td>Send Request to ServerTime.aspx </td>
                <td><input height:12px;width:128px;"=height:12px;width:128px;" alt="search"
type="text" name="myName" size="20" value="" /></td>
                <td>
                    <input id="Submit1" type="submit" value="submit" />
                </td>
            </tr>
        </table>
    </form>
</body>
</html>
```

Change the action property value to “POST” and test the page. Did you see the user’s name displayed? Why?

7. Add the following script in the head part to display the client side time. Save the file as Exercise3.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<script language="javascript">
   function display_c() {
      var refresh = 1000; // Refresh rate in milli seconds
      mytime = setTimeout('display_ct()', refresh)
   }

   function display_ct() {
      var strcount
      var x = new Date()
      // changing the display to UTC string
      var x1 = x.toUTCString();
      //formatting the display
      document.getElementById('ct').style.fontSize = '20px';
      //set the fontcolr to '#0030c0';
      document.getElementById('ct').style.color = '#0030c0';
      document.getElementById('ct').innerHTML = x1;
      tt = display_c();
   }
</script>

</head>
<body onload="display_ct()";>
<span id='ct' ></span>
<br/>
    HTTP Request

<form action= "ServerTime.aspx" method="post">
<table>
<tr>
<td >Send Request to ServerTime.aspx </td>
<td><input height:12px;width:128px;" alt="search" type="text" name="myName"
size="20" value="" /></td>
<td>
   <input id="Submit1" type="submit" value="submit" /></td>
</tr></table>
</form>
</body>
</html>
```

View the file in Browser. Compare the client side time and server side time. Any difference?
Reference: http://www.plus2net.com/javascript_tutorial/clock.php

B. Sending Request using XmlHttpRequest Object

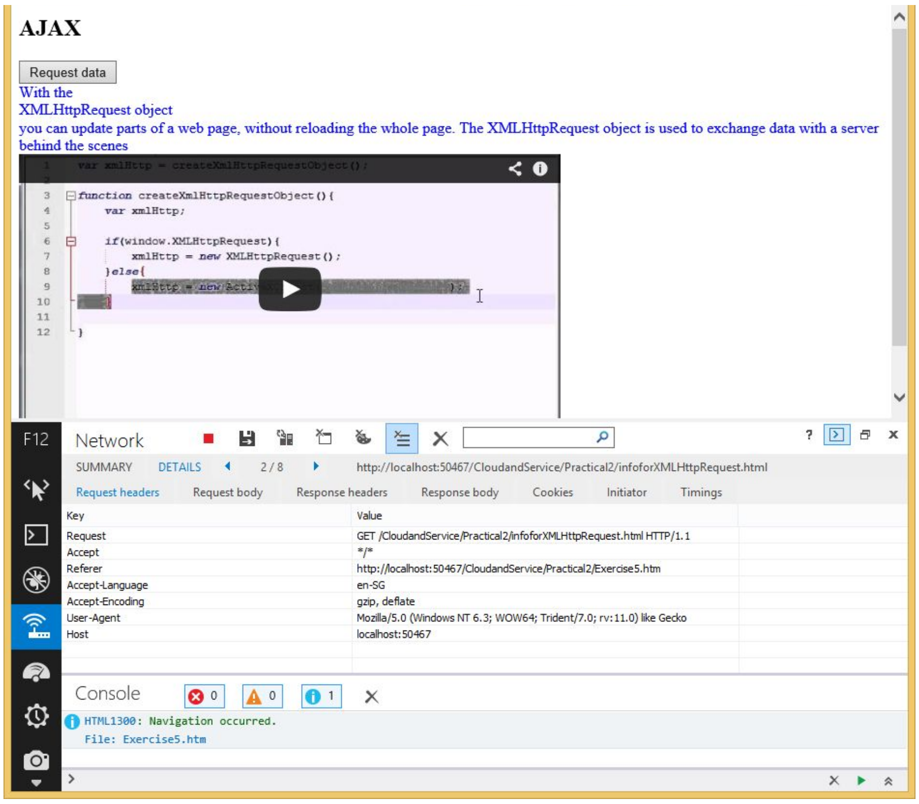
1. Add a HTML page named Exercise5.html.

```
<html>
<head>
  <script>
    function loadXMLDoc() {
      var xmlhttp;
      if (window.XMLHttpRequest) {// code for IE7+, Firefox, Chrome, Opera,
Safari
        xmlhttp = new XMLHttpRequest();
      }
      else {// code for IE6, IE5
        xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
      }
      xmlhttp.onreadystatechange = function () {
        if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {

          /* the innerHTML is between <div id="myDiv">-------</div> */

          document.getElementById("myDiv").innerHTML = '<span
style="color:blue">' + xmlhttp.responseText + "</span>";
        }
      };
      xmlhttp.open("GET", "infoforXMLHttpRequest.txt", true);

      /* we are using GET, null in the body */

      xmlhttp.send(null);
    }
    </script>
</head>
<body>

    <h2>AJAX</h2>
    <button type="button" onclick="loadXMLDoc()">Request data</button>
    <div id="myDiv"></div>

</body>
</html>
```

2. Create a html file named `infoforXMLHttpRequest.txt`

```
<div style=background-color: lightgrey;width: 300px; padding: 25px; border: 25px solid
navy; margin: 25px;>With the <br> XMLHttpRequest object</br> you can update parts of a
web page, without reloading the whole page.
The XMLHttpRequest object is used to exchange data with a server behind the scenes</div>
<iframe width="560" height="315" src="//www.youtube.com/embed/gpx2yNivuUY"
frameborder="0" allowfullscreen></iframe>
```

3. View the page in your browser.



AJAX

Request data

With the
XMLHttpRequest object
you can update parts of a web page, without reloading the whole page. The XMLHttpRequest object is used to exchange data with a server behind the scenes

```
var xmlHttp = createXmlHttpRequestObject();

function createXmlHttpRequestObject(){
    var xmlHttp;

    if(window.XMLHttpRequest){
        xmlHttp = new XMLHttpRequest();
    }else{
        xmlHttp = new Activ[illegible]);
[illegible]

}
```

F12 Network

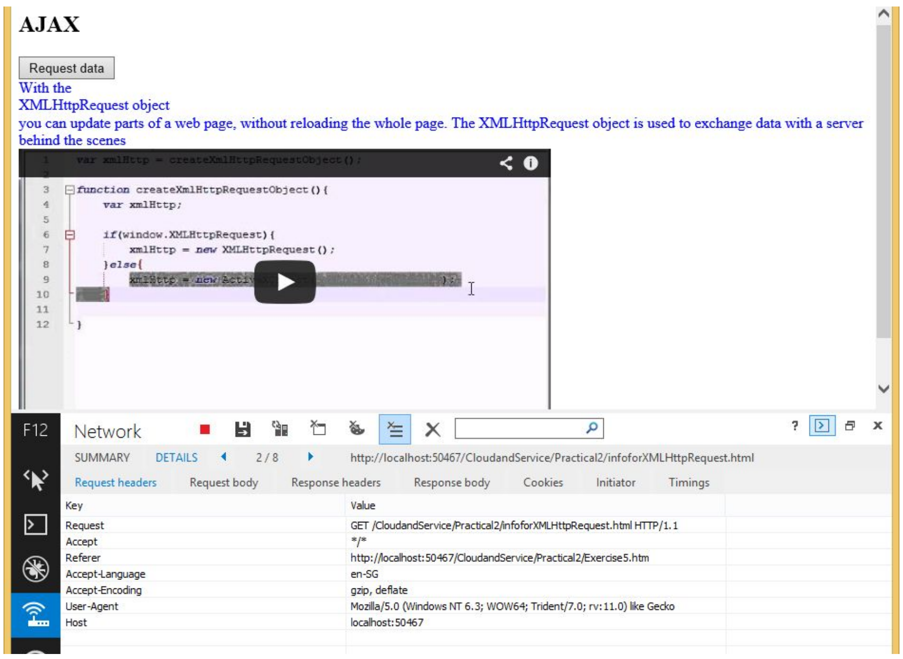
SUMMARY DETAILS 2 / 8 http://localhost:50467/CloudandService/Practical2/infoforXMLHttpRequest.html

Request headers | Request body | Response headers | Response body | Cookies | Initiator | Timings

Key	Value
Request	GET /CloudandService/Practical2/infoforXMLHttpRequest.html HTTP/1.1
Accept	*/*
Referer	http://localhost:50467/CloudandService/Practical2/Exercise5.htm
Accept-Language	en-SG
Accept-Encoding	gzip, deflate
User-Agent	Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko
Host	localhost:50467

Console 0 0 1

```
HTML1300: Navigation occurred.
   File: Exercise5.htm
```



AJAX

Request data

With the
XMLHttpRequest object
you can update parts of a web page, without reloading the whole page. The XMLHttpRequest object is used to exchange data with a server behind the scenes

```
var xmlHttp = createXmlHttpRequestObject();

function createXmlHttpRequestObject(){
    var xmlHttp;

    if(window.XMLHttpRequest){
        xmlHttp = new XMLHttpRequest();
    }else{
        xmlHttp = new Activ[illegible]);
[illegible]

}
```

F12 Network

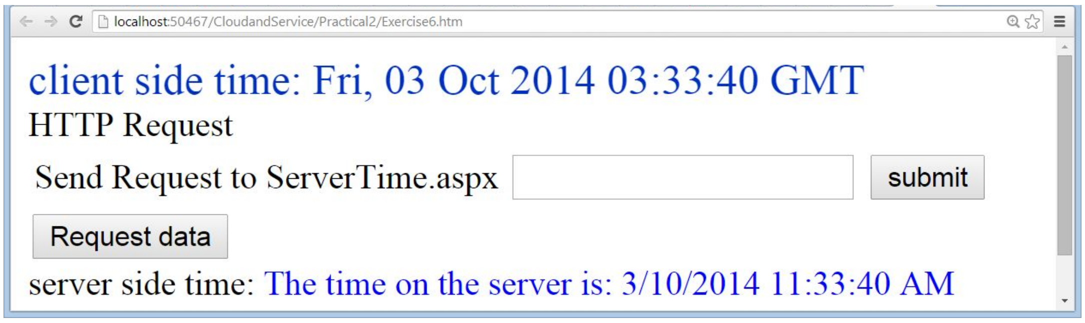
SUMMARY DETAILS 2 / 8 http://localhost:50467/CloudandService/Practical2/infoforXMLHttpRequest.html

Request headers | Request body | Response headers | Response body | Cookies | Initiator | Timings

Key	Value
Request	GET /CloudandService/Practical2/infoforXMLHttpRequest.html HTTP/1.1
Accept	*/*
Referer	http://localhost:50467/CloudandService/Practical2/Exercise5.htm
Accept-Language	en-SG
Accept-Encoding	gzip, deflate
User-Agent	Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko
Host	localhost:50467

4.

Add a HTML page named “Exercise6.html”, and display the client side time and server side time one single page.

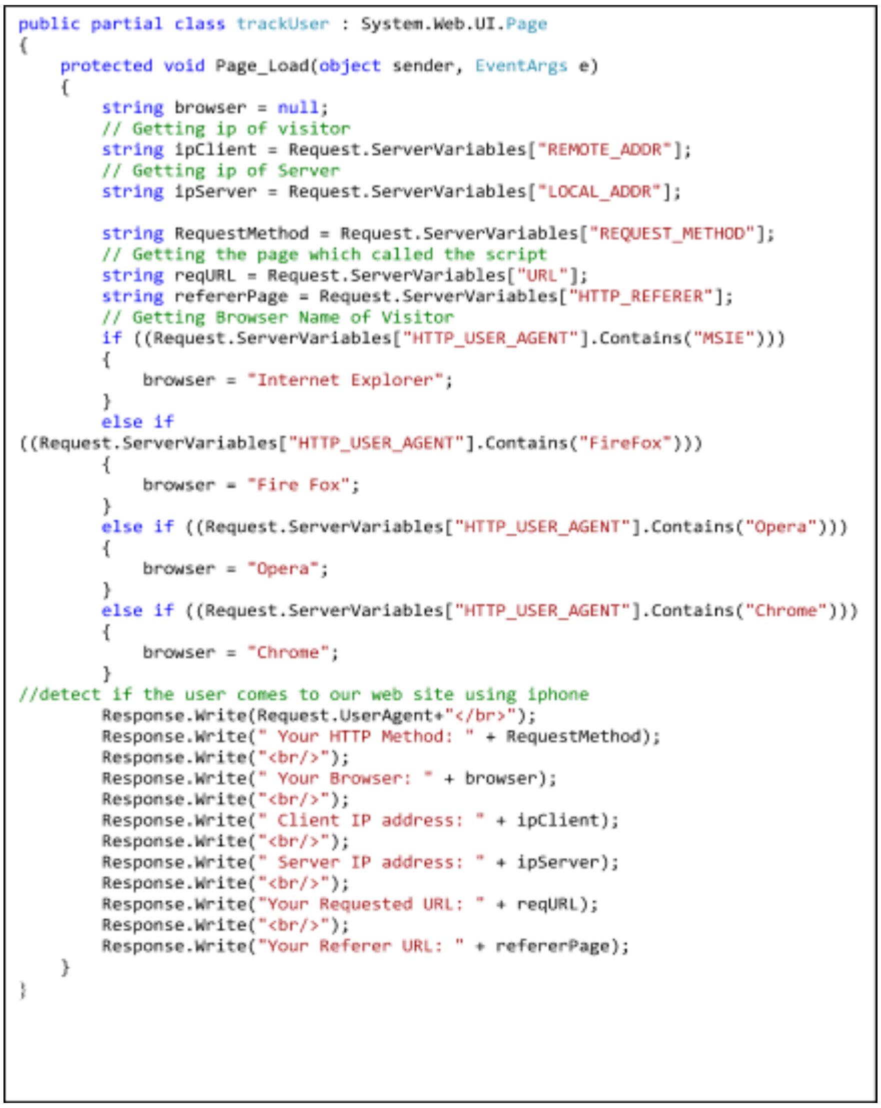


client side time: Fri, 03 Oct 2014 03:33:40 GMT
HTTP Request
Send Request to ServerTime.aspx
server side time: The time on the server is: 3/10/2014 11:33:40 AM

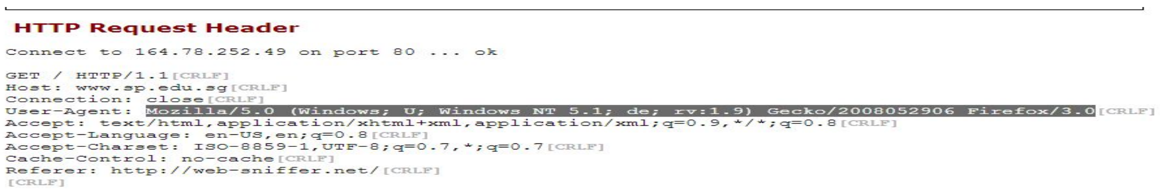
For complete sample code , please refer to the sample code in the CodePen.

C. Tracking the user information using Request.ServerVariables collection

1. Create a web form “trackUser.aspx” to display the user information received at the server side.

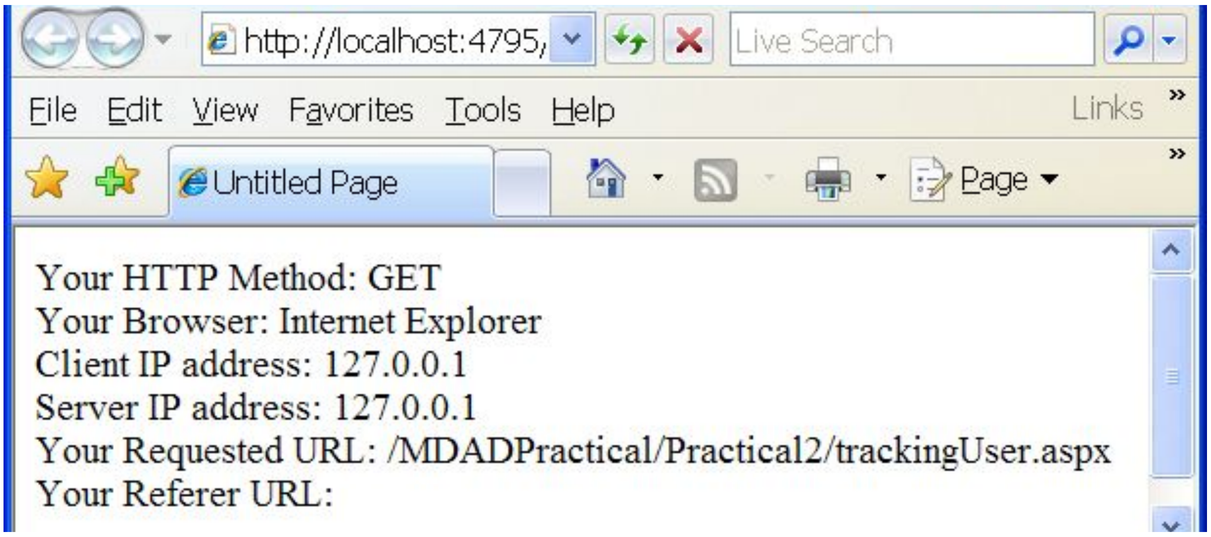


```
public partial class trackUser : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        string browser = null;
        // Getting ip of visitor
        string ipClient = Request.ServerVariables["REMOTE_ADDR"];
        // Getting ip of Server
        string ipServer = Request.ServerVariables["LOCAL_ADDR"];

        string RequestMethod = Request.ServerVariables["REQUEST_METHOD"];
        // Getting the page which called the script
        string reqURL = Request.ServerVariables["URL"];
        string refererPage = Request.ServerVariables["HTTP_REFERER"];
        // Getting Browser Name of Visitor
        if ((Request.ServerVariables["HTTP_USER_AGENT"].Contains("MSIE")))
        {
            browser = "Internet Explorer";
        }
        else if
((Request.ServerVariables["HTTP_USER_AGENT"].Contains("FireFox")))
        {
            browser = "Fire Fox";
        }
        else if ((Request.ServerVariables["HTTP_USER_AGENT"].Contains("Opera")))
        {
            browser = "Opera";
        }
        else if ((Request.ServerVariables["HTTP_USER_AGENT"].Contains("Chrome")))
        {
            browser = "Chrome";
        }
//detect if the user comes to our web site using iphone
        Response.Write(Request.UserAgent+"</br>");
        Response.Write(" Your HTTP Method: " + RequestMethod);
        Response.Write("<br/>");
        Response.Write(" Your Browser: " + browser);
        Response.Write("<br/>");
        Response.Write(" Client IP address: " + ipClient);
        Response.Write("<br/>");
        Response.Write(" Server IP address: " + ipServer);
        Response.Write("<br/>");
        Response.Write("Your Requested URL: " + reqURL);
        Response.Write("<br/>");
        Response.Write("Your Referer URL: " + refererPage);
    }
}
```



HTTP Request Header

```
Connect to 164.78.252.49 on port 80 ... ok

GET / HTTP/1.1[CRLF]
Host: www.sp.edu.sg[CRLF]
Connection: close[CRLF]
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; de; rv:1.9) Gecko/2008052906 Firefox/3.0[CRLF]
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8[CRLF]
Accept-Language: en-US,en;q=0.8[CRLF]
Accept-Charset: ISO-8859-1,UTF-8;q=0.7,*;q=0.7[CRLF]
Cache-Control: no-cache[CRLF]
Referer: http://web-sniffer.net/[CRLF]
[CRLF]
```



```
Your HTTP Method: GET
Your Browser: Internet Explorer
Client IP address: 127.0.0.1
Server IP address: 127.0.0.1
Your Requested URL: /MDADPractical/Practical2/trackingUser.aspx
Your Referer URL:
```

Explain why the client IP address and server IP address are the same?

Why your header field **referrer URL** is empty?

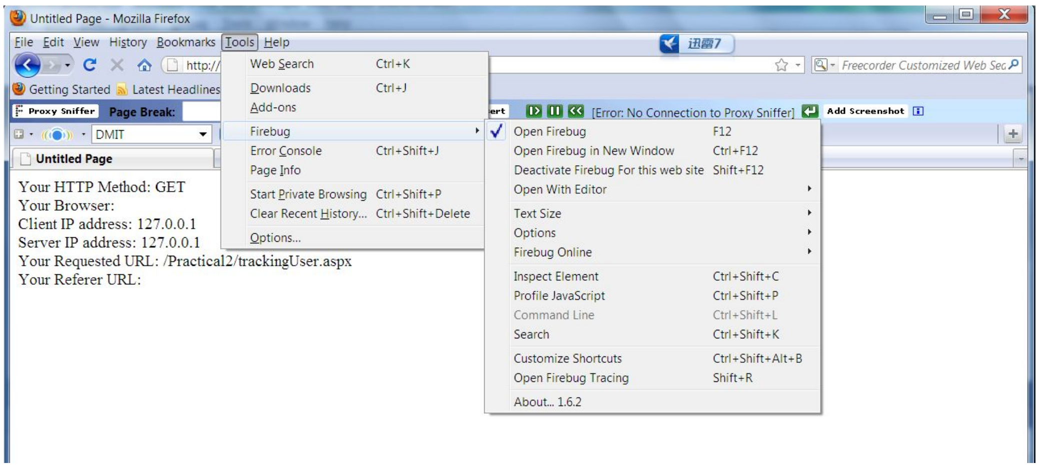
Refer to ASP ServerVariables Collection. http://www.w3schools.com/asp/coll_servervariables.asp

Reference: Detect Mobile Browsers - Open source mobile phone

detectmobilebrowsers.com/

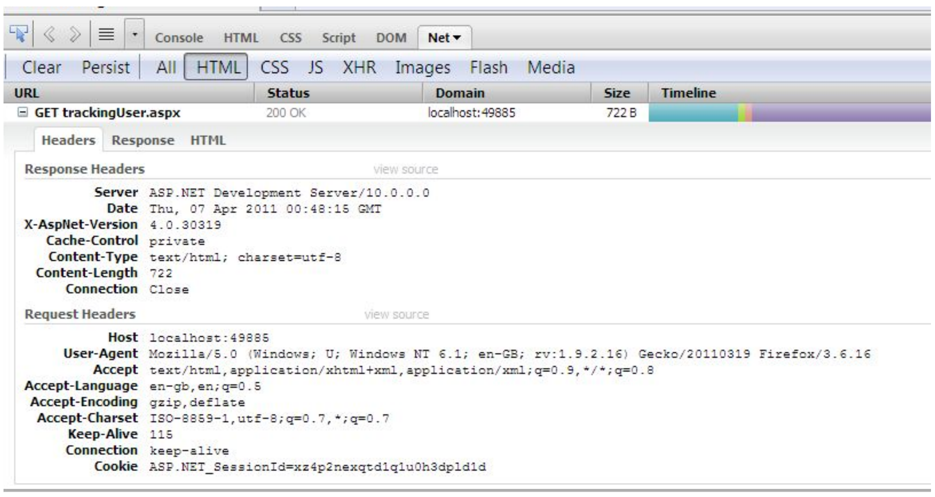
Open source scripts to detect mobile browsers and phones using Apache, JavaScript, PHP, ASP, ColdFusion, C#, .NET, Python, JSP and Rails.

2. View the same page using FireFox


Untitled Page - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Getting Started Latest Headlines
Proxy Sniffer Page Break:
DMIT
Untitled Page
[Error: No Connection to Proxy Sniffer]
Add Screenshot
Freecorder Customized Web Sea
Web Search Ctrl+K
Downloads Ctrl+J
Add-ons
Firebug
Error Console Ctrl+Shift+J
Page Info
Start Private Browsing Ctrl+Shift+P
Clear Recent History... Ctrl+Shift+Delete
Options...
Open Firebug F12
Open Firebug in New Window Ctrl+F12
Deactivate Firebug For this web site Shift+F12
Open With Editor
Text Size
Options
Firebug Online
Inspect Element Ctrl+Shift+C
Profile JavaScript Ctrl+Shift+P
Command Line Ctrl+Shift+L
Search Ctrl+Shift+K
Customize Shortcuts Ctrl+Shift+Alt+B
Open Firebug Tracing Shift+R
About... 1.6.2

```
Your HTTP Method: GET
Your Browser:
Client IP address: 127.0.0.1
Server IP address: 127.0.0.1
Your Requested URL: /Practical2/trackingUser.aspx
Your Referer URL:
```

3. Select Tools→FireBug→OpenFirebug


Console HTML CSS Script DOM Net
Clear Persist All HTML CSS JS XHR Images Flash Media
Headers Response HTML
view source

```
URL                     Status    Domain            Size    Timeline
GET trackingUser.aspx   200 OK    localhost:49885   722 B

Response Headers
          Server ASP.NET Development Server/10.0.0.0
            Date Thu, 07 Apr 2011 00:48:15 GMT
X-AspNet-Version 4.0.30319
   Cache-Control private
    Content-Type text/html; charset=utf-8
  Content-Length 722
      Connection Close

Request Headers
            Host localhost:49885
      User-Agent Mozilla/5.0 (Windows; U; Windows NT 6.1; en-GB; rv:1.9.2.16) Gecko/20110319 Firefox/3.6.16
          Accept text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
 Accept-Language en-gb,en;q=0.5
 Accept-Encoding gzip,deflate
  Accept-Charset ISO-8859-1,utf-8;q=0.7,*;q=0.7
      Keep-Alive 115
      Connection keep-alive
          Cookie ASP.NET_SessionId=xz4p2nexqtd1q1u0h3dpld1d
```

iPhone User Agent

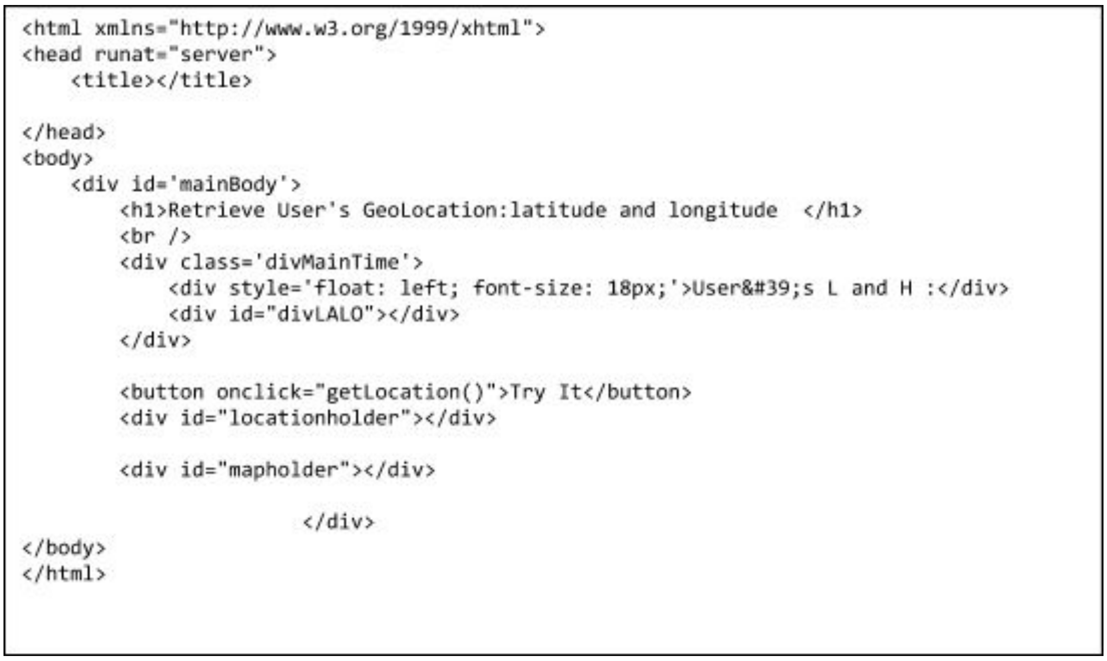
I know you're curious about what the user agent for the iPhone looks like, so here it is:

```
Mozilla/5.0 (iPhone; U; CPU iPhone OS 3_0 like Mac OS X; en-us) AppleWebKit/528.18 (KHTML,
like Gecko) Version/4.0 Mobile/7A341 Safari/528.16
```

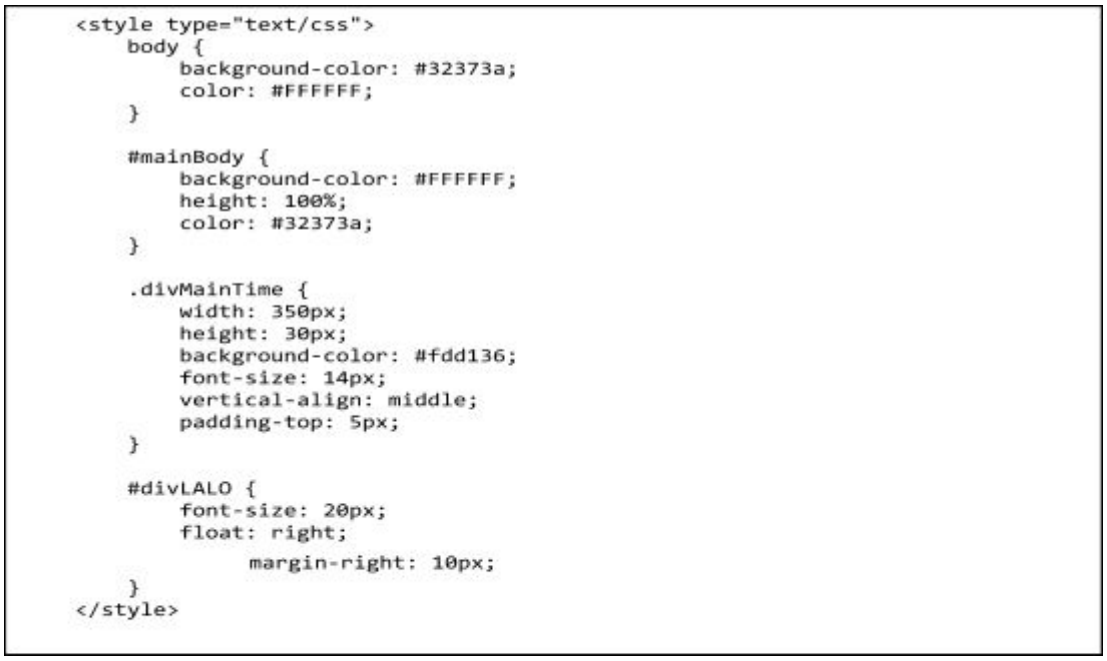
Modify the above code so that it could be able to detect if an Iphone user comes to your site.
Reference: http://www.michaelbarton.name/2010/01/14/how-to-detect-if-an-iphone-user-comes-to-your-site/

4. Get user's Geo location at the client side.
   Add an web form to the project as Geolacation.aspx

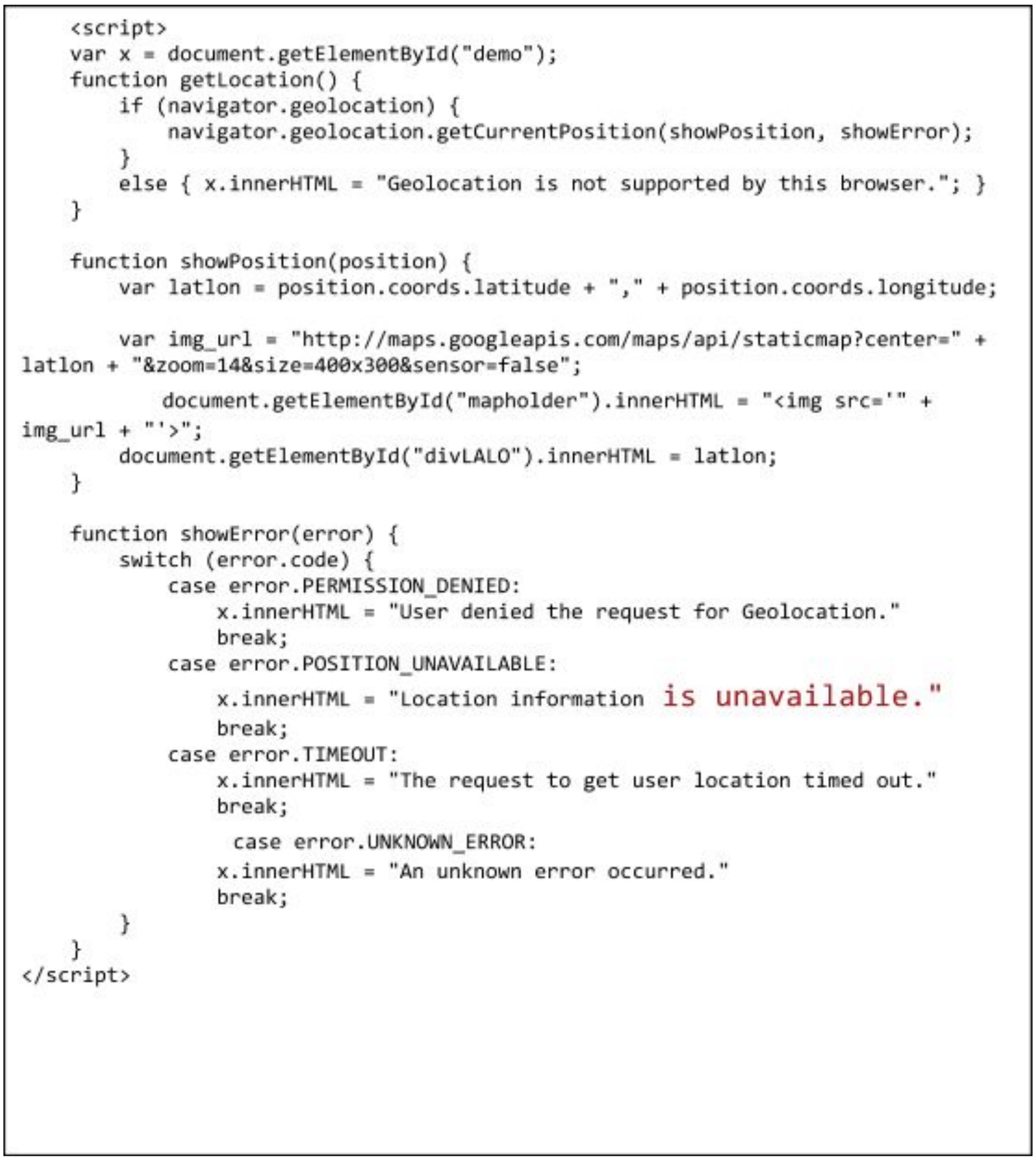
Refer to online document:
http://www.w3schools.com/html/html5_geolocation.asp)



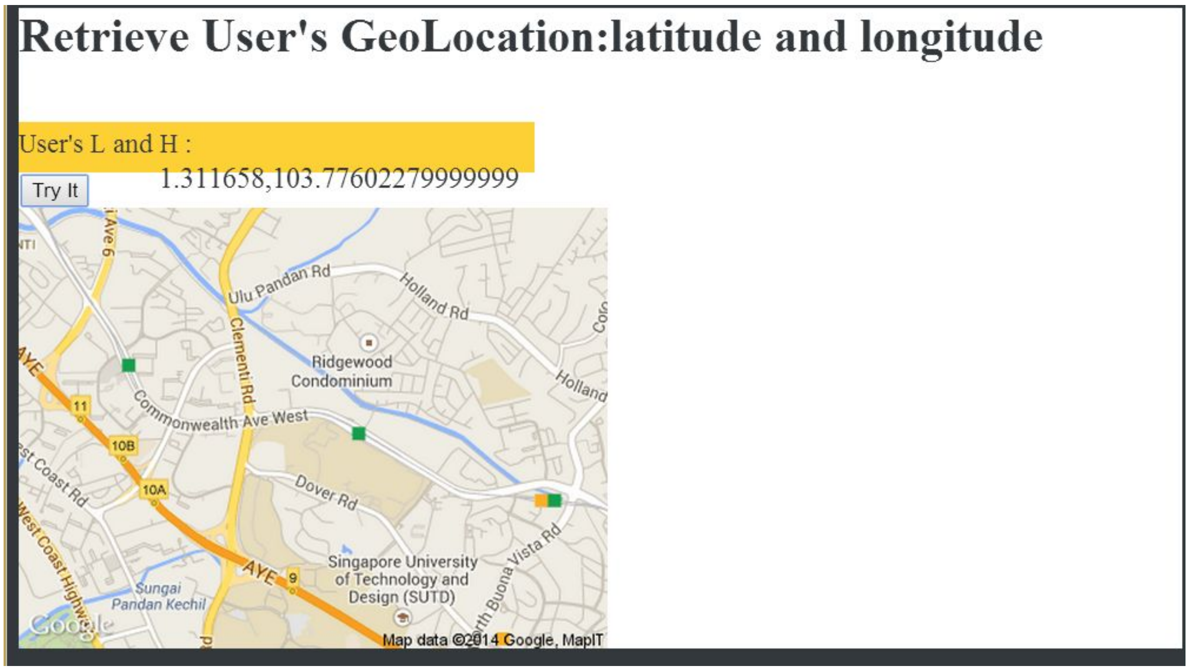
```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>

</head>
<body>
    <div id='mainBody'>
        <h1>Retrieve User's GeoLocation:latitude and longitude  </h1>
        <br />
        <div class='divMainTime'>
            <div style='float: left; font-size: 18px;'>User&#39;s L and H :</div>
            <div id="divLALO"></div>
        </div>

        <button onclick="getLocation()">Try It</button>
        <div id="locationholder"></div>

        <div id="mapholder"></div>

                           </div>
</body>
</html>
```



```
    <style type="text/css">
        body {
            background-color: #32373a;
            color: #FFFFFF;
        }

        #mainBody {
            background-color: #FFFFFF;
            height: 100%;
            color: #32373a;
        }

        .divMainTime {
            width: 350px;
            height: 30px;
            background-color: #fdd136;
            font-size: 14px;
            vertical-align: middle;
            padding-top: 5px;
        }

        #divLALO {
            font-size: 20px;
            float: right;
                 margin-right: 10px;
        }
    </style>
```



```
    <script>
    var x = document.getElementById("demo");
    function getLocation() {
        if (navigator.geolocation) {
            navigator.geolocation.getCurrentPosition(showPosition, showError);
        }
        else { x.innerHTML = "Geolocation is not supported by this browser."; }
    }

    function showPosition(position) {
        var latlon = position.coords.latitude + "," + position.coords.longitude;

        var img_url = "http://maps.googleapis.com/maps/api/staticmap?center=" +
latlon + "&zoom=14&size=400x300&sensor=false";
            document.getElementById("mapholder").innerHTML = "<img src='" +
img_url + "'>";
        document.getElementById("divLALO").innerHTML = latlon;
    }

    function showError(error) {
        switch (error.code) {
            case error.PERMISSION_DENIED:
                x.innerHTML = "User denied the request for Geolocation."
                break;
            case error.POSITION_UNAVAILABLE:
                x.innerHTML = "Location information is unavailable."
                break;
            case error.TIMEOUT:
                x.innerHTML = "The request to get user location timed out."
                break;
                 case error.UNKNOWN_ERROR:
                x.innerHTML = "An unknown error occurred."
                break;
        }
    }
</script>
```



Retrieve User's GeoLocation:latitude and longitude

User's L and H :

1.311658,103.77602279999999

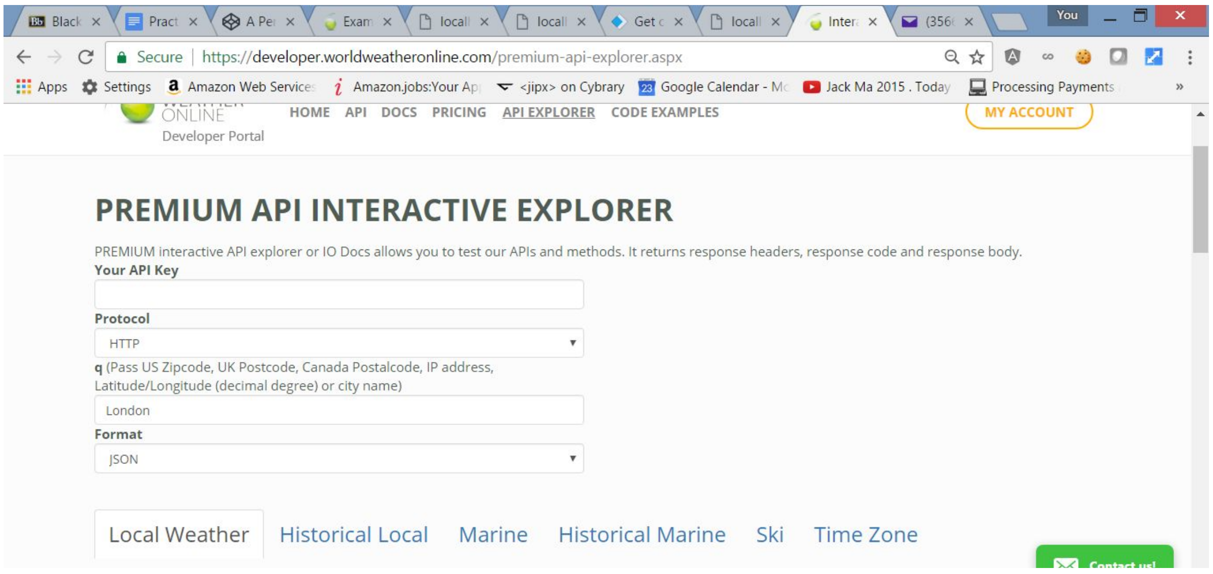
5. Add the following code to display the country name.

```
    <div>
            <script type="text/javascript"
src="http://j.maxmind.com/app/geoip.js"></script>
            <br />
            Country Code:
    <script type="text/javascript">document.write(geoip_country_code());</script>
            <br />
            Country Name:
    <script type="text/javascript">document.write(geoip_country_name());</script>
            <br />
            City:
    <script type="text/javascript">document.write(geoip_city());</script>
            <br />
            Region:
    <script type="text/javascript">document.write(geoip_region());</script>
            <br />
            Region Name:
    <script type="text/javascript">document.write(geoip_region_name());</script>
            <br />
            Latitude:
    <script type="text/javascript">document.write(geoip_latitude());</script>
            <br />
            Longitude:
    <script type="text/javascript">document.write(geoip_longitude());</script>
            <br />
            Postal Code:
    <script type="text/javascript">document.write(geoip_postal_code());</script>
        </div>
```

D. Sending a request to an existing Web Service using Online Tools

https://developer.worldweatheronline.com/signup.aspx

https://developer.worldweatheronline.com/premium-api-explorer.aspx


https://developer.worldweatheronline.com/premium-api-explorer.aspx
Developer Portal
HOME API DOCS PRICING API EXPLORER CODE EXAMPLES
MY ACCOUNT

PREMIUM API INTERACTIVE EXPLORER

PREMIUM interactive API explorer or IO Docs allows you to test our APIs and methods. It returns response headers, response code and response body.

Your API Key

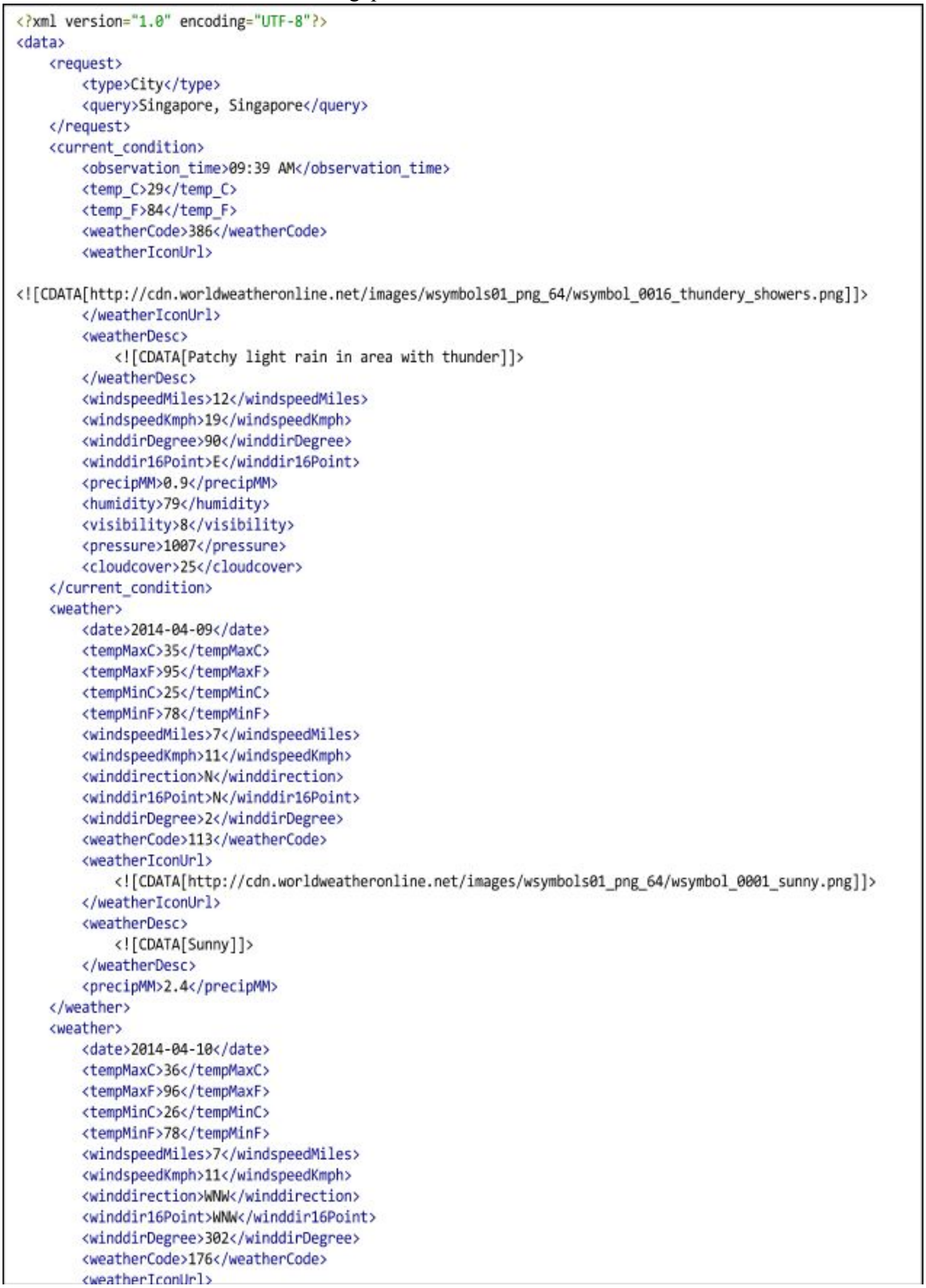
Protocol: HTTP

q (Pass US Zipcode, UK Postcode, Canada Postalcode, IP address, Latitude/Longitude (decimal degree) or city name): London

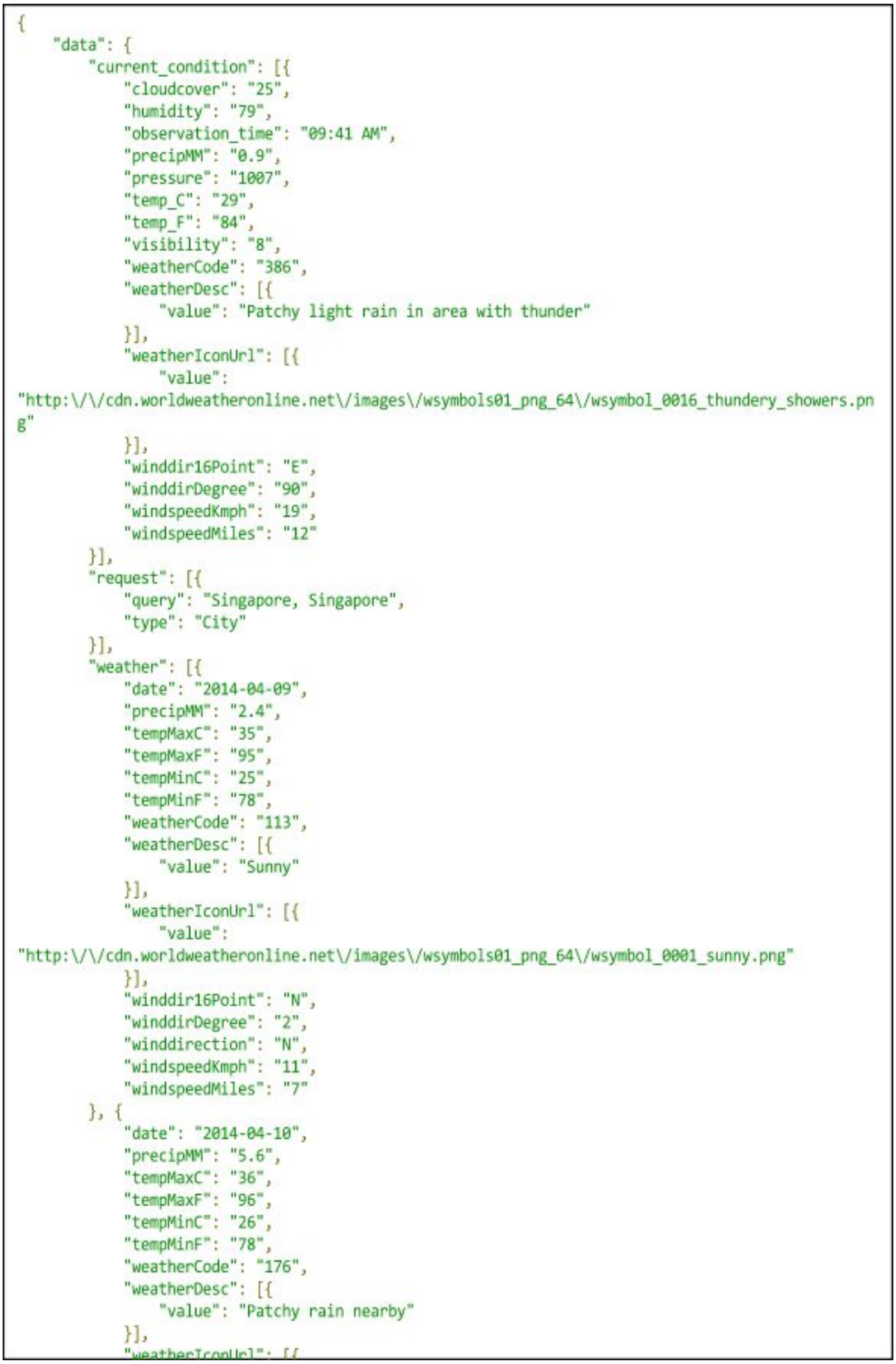
Format: JSON

Local Weather
Historical Local
Marine
Historical Marine
Ski
Time Zone
Contact us!

Get the weather condition in Singapore in XML format and Json format.

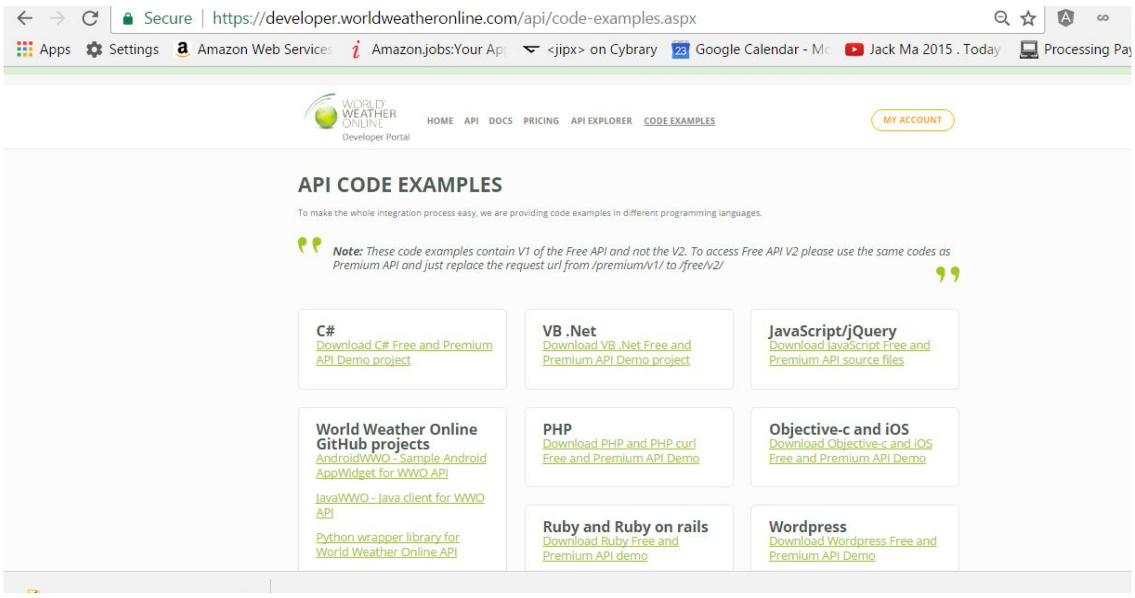


```
<?xml version="1.0" encoding="UTF-8"?>
<data>
    <request>
        <type>City</type>
        <query>Singapore, Singapore</query>
    </request>
    <current_condition>
        <observation_time>09:39 AM</observation_time>
        <temp_C>29</temp_C>
        <temp_F>84</temp_F>
        <weatherCode>386</weatherCode>
        <weatherIconUrl>

<![CDATA[http://cdn.worldweatheronline.net/images/wsymbols01_png_64/wsymbol_0016_thundery_showers.png]]>
        </weatherIconUrl>
        <weatherDesc>
            <![CDATA[Patchy light rain in area with thunder]]>
        </weatherDesc>
        <windspeedMiles>12</windspeedMiles>
        <windspeedKmph>19</windspeedKmph>
        <winddirDegree>90</winddirDegree>
        <winddir16Point>E</winddir16Point>
        <precipMM>0.9</precipMM>
        <humidity>79</humidity>
        <visibility>8</visibility>
        <pressure>1007</pressure>
        <cloudcover>25</cloudcover>
    </current_condition>
    <weather>
        <date>2014-04-09</date>
        <tempMaxC>35</tempMaxC>
        <tempMaxF>95</tempMaxF>
        <tempMinC>25</tempMinC>
        <tempMinF>78</tempMinF>
        <windspeedMiles>7</windspeedMiles>
        <windspeedKmph>11</windspeedKmph>
        <winddirection>N</winddirection>
        <winddir16Point>N</winddir16Point>
        <winddirDegree>2</winddirDegree>
        <weatherCode>113</weatherCode>
        <weatherIconUrl>
            <![CDATA[http://cdn.worldweatheronline.net/images/wsymbols01_png_64/wsymbol_0001_sunny.png]]>
        </weatherIconUrl>
        <weatherDesc>
            <![CDATA[Sunny]]>
        </weatherDesc>
        <precipMM>2.4</precipMM>
    </weather>
    <weather>
        <date>2014-04-10</date>
        <tempMaxC>36</tempMaxC>
        <tempMaxF>96</tempMaxF>
        <tempMinC>26</tempMinC>
        <tempMinF>78</tempMinF>
        <windspeedMiles>7</windspeedMiles>
        <windspeedKmph>11</windspeedKmph>
        <winddirection>WNW</winddirection>
        <winddir16Point>WNW</winddir16Point>
        <winddirDegree>302</winddirDegree>
        <weatherCode>176</weatherCode>
        <weatherIconUrl>
```



```
{
    "data": {
        "current_condition": [{
            "cloudcover": "25",
            "humidity": "79",
            "observation_time": "09:41 AM",
            "precipMM": "0.9",
            "pressure": "1007",
            "temp_C": "29",
            "temp_F": "84",
            "visibility": "8",
            "weatherCode": "386",
            "weatherDesc": [{
                "value": "Patchy light rain in area with thunder"
            }],
            "weatherIconUrl": [{
                "value":
"http:\/\/cdn.worldweatheronline.net\/images\/wsymbols01_png_64\/wsymbol_0016_thundery_showers.pn
g"
            }],
            "winddir16Point": "E",
            "winddirDegree": "90",
            "windspeedKmph": "19",
            "windspeedMiles": "12"
        }],
        "request": [{
            "query": "Singapore, Singapore",
            "type": "City"
        }],
        "weather": [{
            "date": "2014-04-09",
            "precipMM": "2.4",
            "tempMaxC": "35",
            "tempMaxF": "95",
            "tempMinC": "25",
            "tempMinF": "78",
            "weatherCode": "113",
            "weatherDesc": [{
                "value": "Sunny"
            }],
            "weatherIconUrl": [{
                "value":
"http:\/\/cdn.worldweatheronline.net\/images\/wsymbols01_png_64\/wsymbol_0001_sunny.png"
            }],
            "winddir16Point": "N",
            "winddirDegree": "2",
            "winddirection": "N",
            "windspeedKmph": "11",
            "windspeedMiles": "7"
        }, {
            "date": "2014-04-10",
            "precipMM": "5.6",
            "tempMaxC": "36",
            "tempMaxF": "96",
            "tempMinC": "26",
            "tempMinF": "78",
            "weatherCode": "176",
            "weatherDesc": [{
                "value": "Patchy rain nearby"
            }],
            "weatherIconUrl": [{
```

Read more:



API CODE EXAMPLES

To make the whole integration process easy, we are providing code examples in different programming languages.

Note: *These code examples contain V1 of the Free API and not the V2. To access Free API V2 please use the same codes as Premium API and just replace the request url from /premium/v1/ to /free/v2/*

C#
Download C# Free and Premium API Demo project

VB .Net
Download VB .Net Free and Premium API Demo project

JavaScript/jQuery
Download JavaScript Free and Premium API source files

World Weather Online GitHub projects
AndroidWWO - Sample Android AppWidget for WWO API

JavaWWO - Java client for WWO API

Python wrapper library for World Weather Online API

PHP
Download PHP and PHP curl Free and Premium API Demo

Objective-c and iOS
Download Objective-c and iOS Free and Premium API Demo

Ruby and Ruby on rails
Download Ruby Free and Premium API demo

Wordpress
Download Wordpress Free and Premium API Demo

View the json text in json viewer at http://jsonviewer.stack.hu/, http://codebeautify.org/view/jsonviewer

View the xml text in xml viewer at
http://xmlgrid.net/
http://codebeautify.org/xmlviewer/7c1e02

What is the function of XMLHttpRequest object.

The XMLHttpRequest object is used to exchange data with a server.

Send a Request To a Server

To send a request to a server, we use the open() and send() methods of the XMLHttpRequest object:

```
xmlhttp.open("GET","ajax_info.txt",true);
xmlhttp.send();
```

Method	Description
open(*method,url,async*)	Specifies the type of request, the URL, and if the request should be handled asynchrono *method*: the type of request: GET or POST *url*: the location of the file on the server *async*: true (asynchronous) or false (synchronous)

send(*string*)	Sends the request off to the server. *string*: Only used for POST requests

GET or POST?

GET is simpler and faster than POST, and can be used in most cases.

However, always use POST requests when:

- A cached file is not an option (update a file or database on the server)
- Sending a large amount of data to the server (POST has no size limitations)
- Sending user input (which can contain unknown characters), POST is more robust and secure than GET

Part 2

Tasks

Creating a web API 2 Restful Service

Adapted from Getting Started with ASP.NET Web API 2 (C#)http://www.asp.net/web-api/overview/getting-started-with-aspnet-web-api/tutorial-your-first-web-api

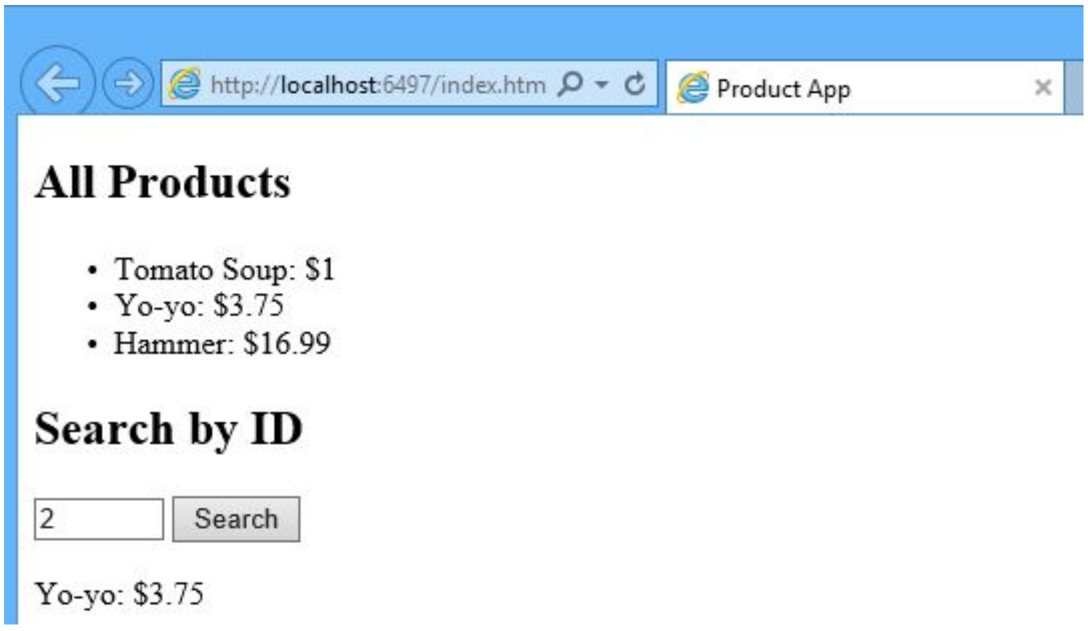
HTTP is not just for serving up web pages. It is also a powerful platform for building APIs that expose services and data. HTTP is simple, flexible, and ubiquitous. Almost any platform that you can think of has an HTTP library, so HTTP services can reach a broad range of clients, including browsers, mobile devices, and traditional desktop applications.

ASP.NET Web API is a framework for building web APIs on top of the .NET Framework. In this tutorial, you will use ASP.NET Web API to create a web API that returns a list of products.

Create a Web API Project

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.

This is how your final products look like:



All Products

- Tomato Soup: $1
- Yo-yo: $3.75
- Hammer: $16.99

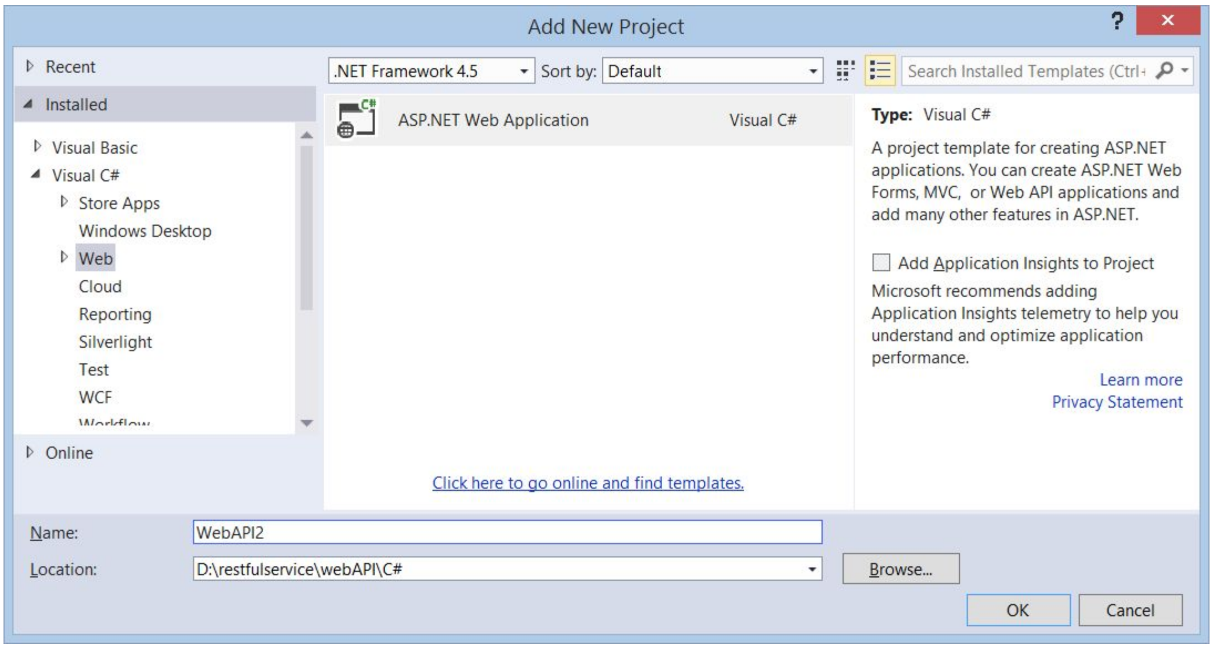
Search by ID

2

Yo-yo: $3.75

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 "<u>**WebAPI2**</u>" and click **OK**.



Type: Visual C#

A project template for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

☐ Add Application Insights to Project

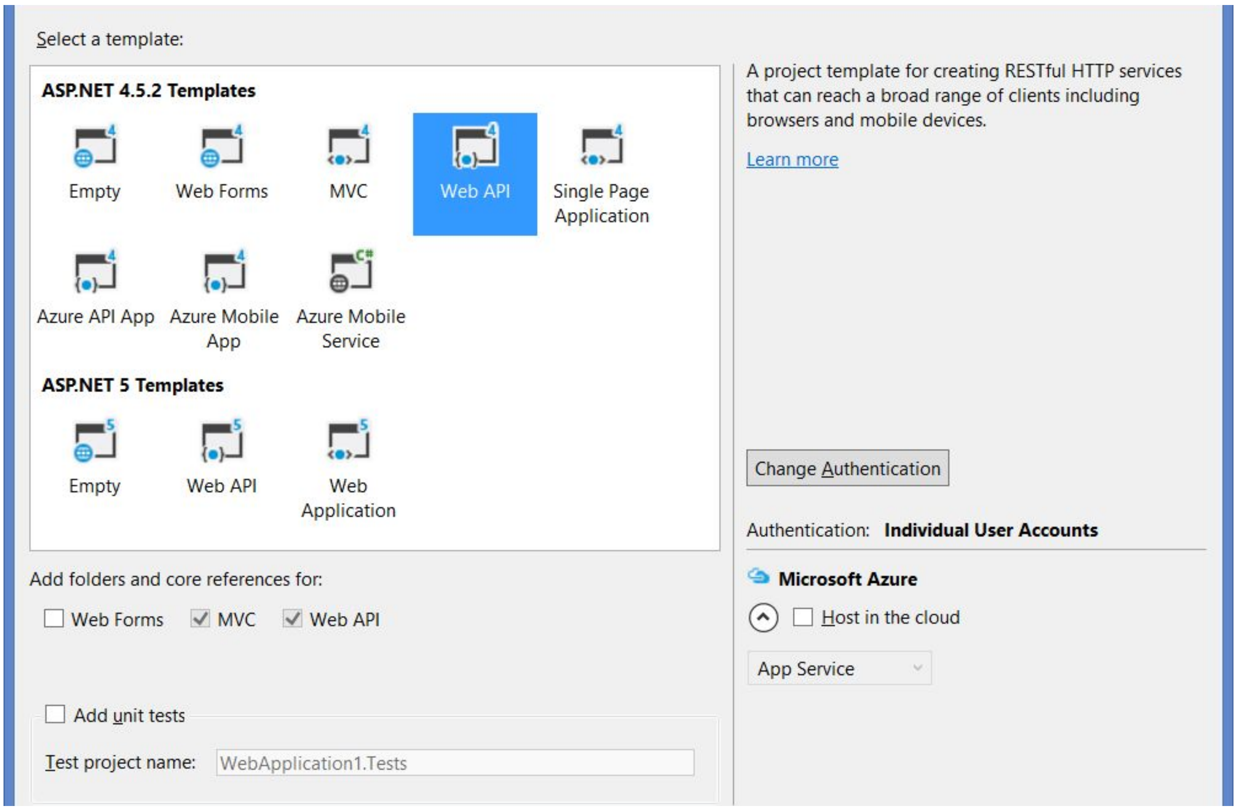
Microsoft recommends adding Application Insights telemetry to help you understand and optimize application performance.

Learn more
Privacy Statement

Name: WebAPI2

Location: D:\restfulservice\webAPI\C#

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



A project template for creating RESTful HTTP services that can reach a broad range of clients including browsers and mobile devices.

Authentication: **Individual User Accounts**

Add folders and core references for:

☐ Web Forms ☑ MVC ☑ Web API

☐ Host in the cloud

☐ Add unit tests

Test project name: WebApplication1.Tests

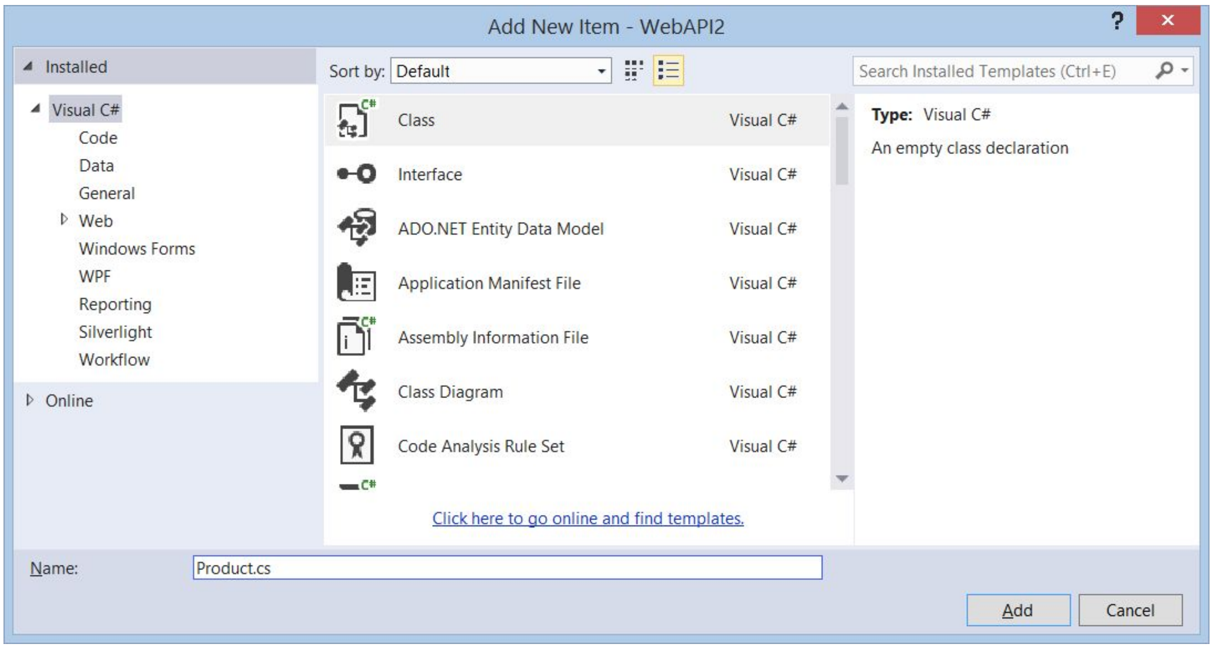
You can also create a Web API project using the "Web API" template. The Web API template uses ASP.NET MVC to provide API help pages. I'm using the Empty template for this tutorial because I want to show Web API without MVC. In general, you don't need to know ASP.NET MVC to use Web API.

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.

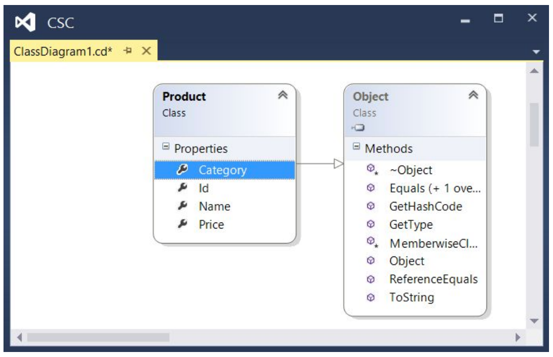
If Solution Explorer is not already visible, click the **View** menu and select **Solution Explorer**. In Solution Explorer, right-click the Models folder. From the context menu, select **Add** then select **Class**.


Add New Item - WebAPI2
Installed
Visual C#
Code
Data
General
Web
Windows Forms
WPF
Reporting
Silverlight
Workflow
Online
Sort by: Default
Search Installed Templates (Ctrl+E)
Class Visual C#
Interface Visual C#
ADO.NET Entity Data Model Visual C#
Application Manifest File Visual C#
Assembly Information File Visual C#
Class Diagram Visual C#
Code Analysis Rule Set Visual C#
Click here to go online and find templates.
Type: Visual C#
An empty class declaration
Add
Cancel

Name: Product.cs

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; }
    }
}
```



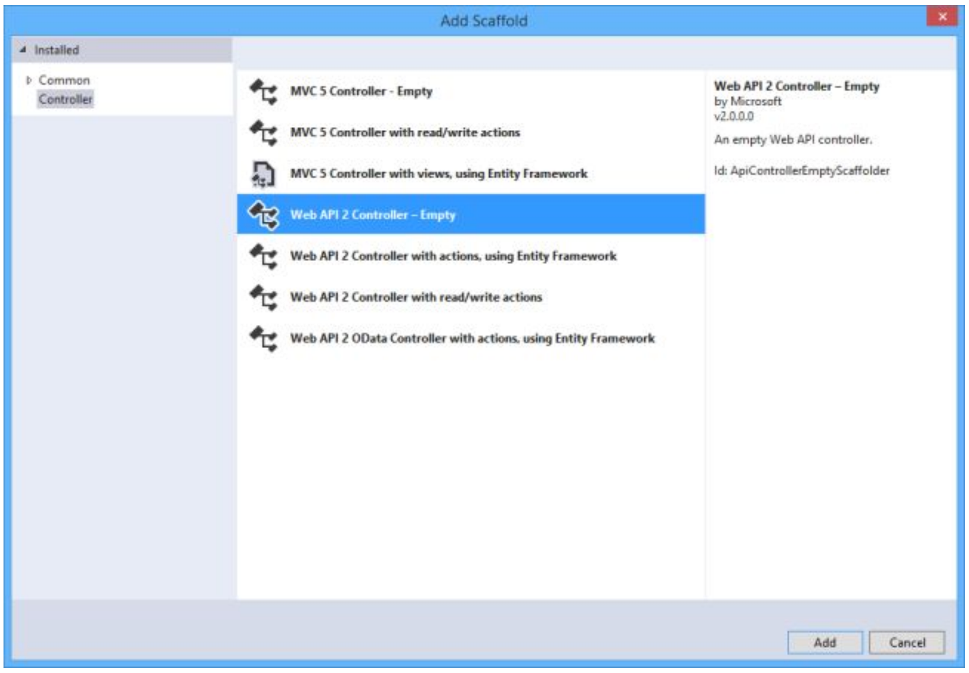
Adding a Controller

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.

Note If you have used ASP.NET MVC, you are already familiar with controllers. Web API controllers are similar to MVC controllers, but inherit the **ApiController** class instead of the **Controller** class.

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**.



Add Scaffold

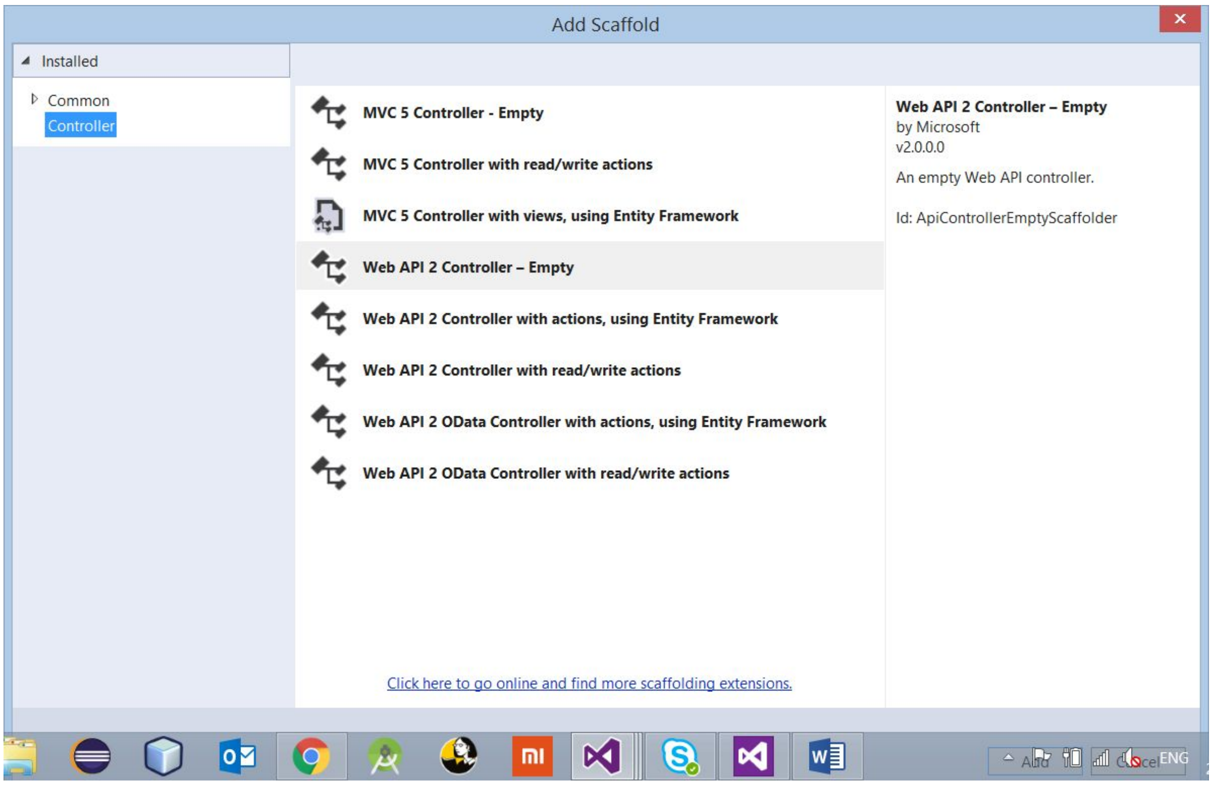
Installed
Common
Controller

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
Web API 2 OData Controller with actions, using Entity Framework

Web API 2 Controller – Empty
by Microsoft
v2.0.0.0
An empty Web API controller.
Id: ApiControllerEmptyScaffolder

Add Cancel

In the **Add Controller** dialog, name the controller "ProductsControllerV1". Click **Add**.



Add Scaffold

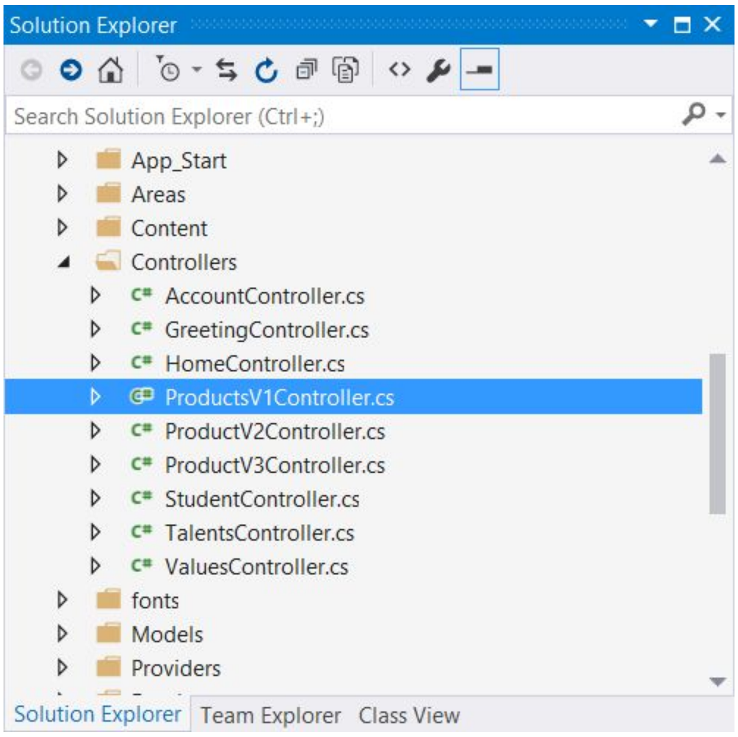
Installed
Common
Controller

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
Web API 2 OData Controller with actions, using Entity Framework
Web API 2 OData Controller with read/write actions

Web API 2 Controller – Empty
by Microsoft
v2.0.0.0
An empty Web API controller.
Id: ApiControllerEmptyScaffolder

Click here to go online and find more scaffolding extensions.

The scaffolding creates a file named **ProductsV1Controller**.cs in the Controllers folder.



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 ProductsApp.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Web.Http;
```

```
using ProductStore.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Web.Http;
using System.Web.Http.Cors;


namespace ProductStore.Controllers
{
       [EnableCors(origins:
"http://localhost:49345,http://csc123client.azurewebsites.net", headers: "*",
methods: "*")]
```

```
       public class ProductsController : ApiController
       {

       // static readonly IProductRepository repository = new
ProductRepository();

       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 }
       };

       //code for version 1


       [HttpGet]
       [Route("api/v1/products/version")]
       //http://localhost:9000/api/v1/products/version
       public string[] GetVersion()
       {
              return new string[]
              {"hello",
                     "version 2",
                     "2"
              };
       }


       [HttpGet]
       [Route("api/v1/products/message/")]

//http://localhost:9000/api/v1/products/message?name1=ji&name2=jii1&name3=ji3
       public HttpResponseMessage GetMultipleNames(String name1, string name2,
string name3)
       {
              var response = new HttpResponseMessage();
              response.Content = new StringContent("<html><body>Hello World "
+
                     " name1 =" + name1 +
                     " name2= " + name2 +
                     " name3=" + name3 +
                     " is provided in path parameter</body></html>");
             response.Content.Headers.ContentType = new
MediaTypeHeaderValue("text/html");
              return response;
       }
```

```
        [HttpGet]
        [Route("api/v1/products")]
        //http://localhost:9000/api/v1/products
        public IEnumerable<Product> GetAllProducts()
          {
               return products;
          }


        [HttpGet]
        [Route("api/v1/products/{id:int:min(2)}")]
        //http://localhost:9000/api/v1/products/3
        public IHttpActionResult GetProduct(int id)
        {
             var product = products.FirstOrDefault((p) => p.Id == id);
             if (product == null)
             {
             return NotFound();
           }
             return Ok(product);
        }
        }


        }
```

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:

- The `GetAllProducts` method returns the entire list of products as an **IEnumerable<Product>** type.
- The `GetProduct` method looks up a single product by its ID.

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/v1/products
GetProduct	/api/v1/products/*id*

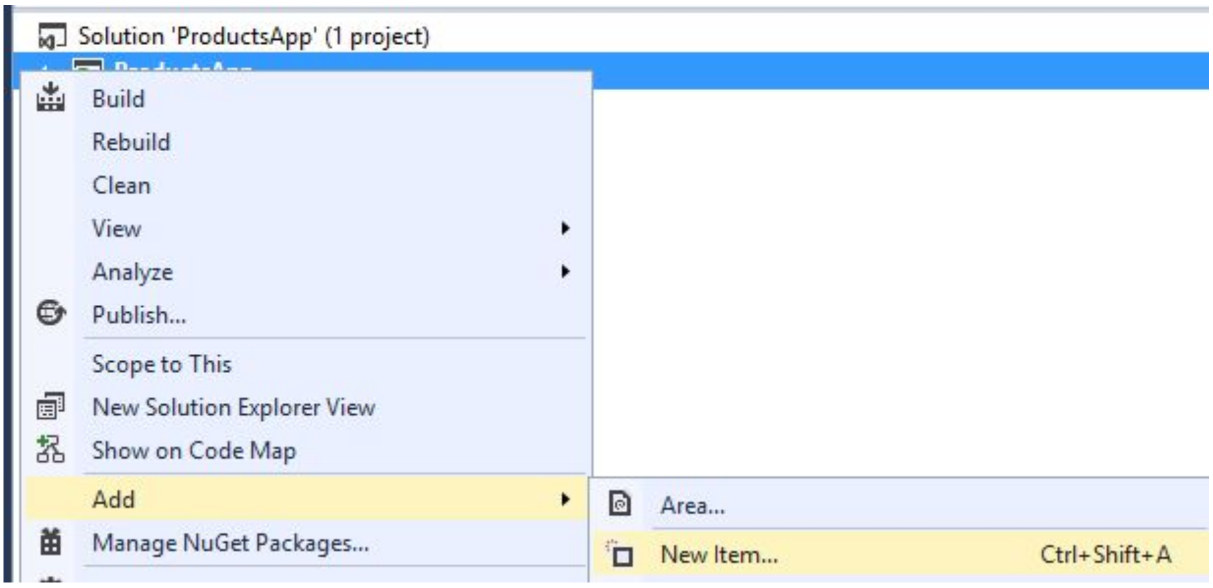
For the `GetProduct` method, the *id* in the URI is a placeholder. For example, to get the product with ID of 5, the URI is `api/v1/products/5`.

For more information about how Web API routes HTTP requests to controller methods, see Routing in ASP.NET Web API.

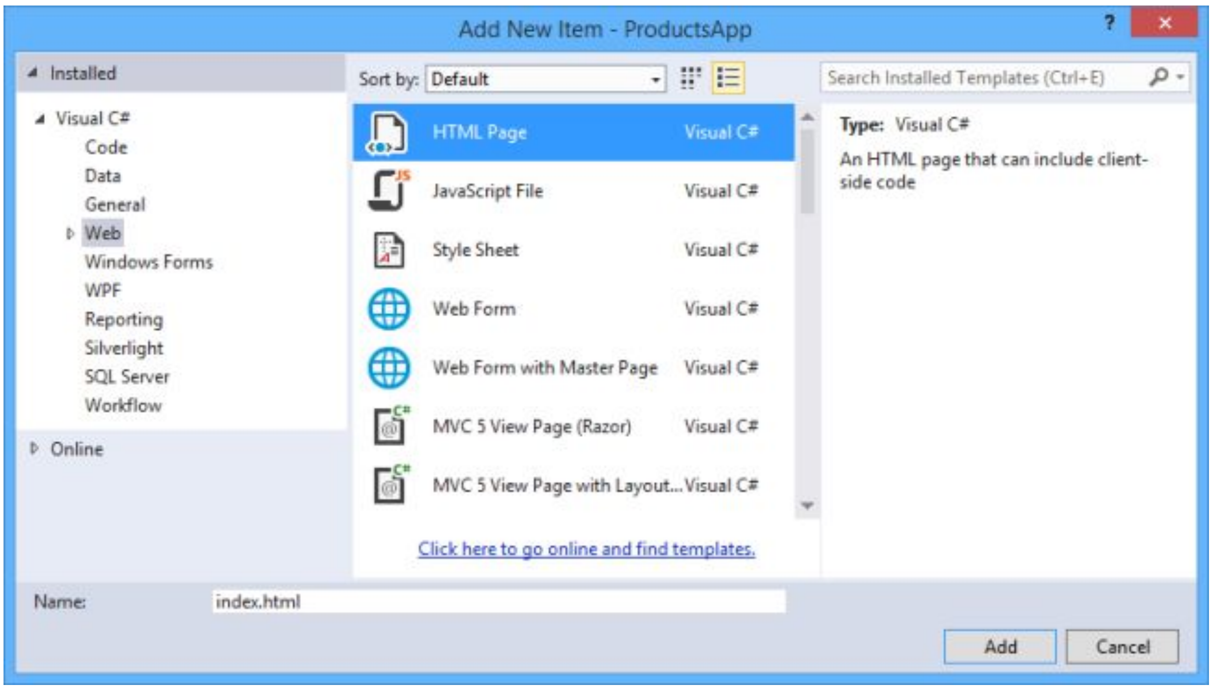
Calling the Web API with Javascript and jQuery

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 (webAPI2) 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".


Add New Item - ProductsApp
Installed
Visual C#
Code
Data
General
Web
Windows Forms
WPF
Reporting
Silverlight
SQL Server
Workflow
Online
Sort by: Default
Search Installed Templates (Ctrl+E)
HTML Page Visual C#
JavaScript File Visual C#
Style Sheet Visual C#
Web Form Visual C#
Web Form with Master Page Visual C#
MVC 5 View Page (Razor) Visual C#
MVC 5 View Page with Layout... Visual C#
Click here to go online and find templates.
Add
Cancel

Type: Visual C#

An HTML page that can include client-side code

Name: 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>
    var uri = 'api/v1/products';
//http://localhost:9000/api/v1/products/
//http://csc123.azurewebsites.net/api/v1/products


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

    function formatItem(item) {
      return item.Name + ': $' + item.Price;
    }

    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. You can also download it from http://jquery.com/, and the ASP.NET "Web API" project template includes jQuery as well.

Getting a List of Products

To get a list of products, send an HTTP GET request to "/api/products".

The jQuery getJSON function sends an AJAX request. For response contains array of JSON objects. The `done` function specifies a callback that is called if the request succeeds. In the callback, we update the DOM with the product information.

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

Getting a Product By ID

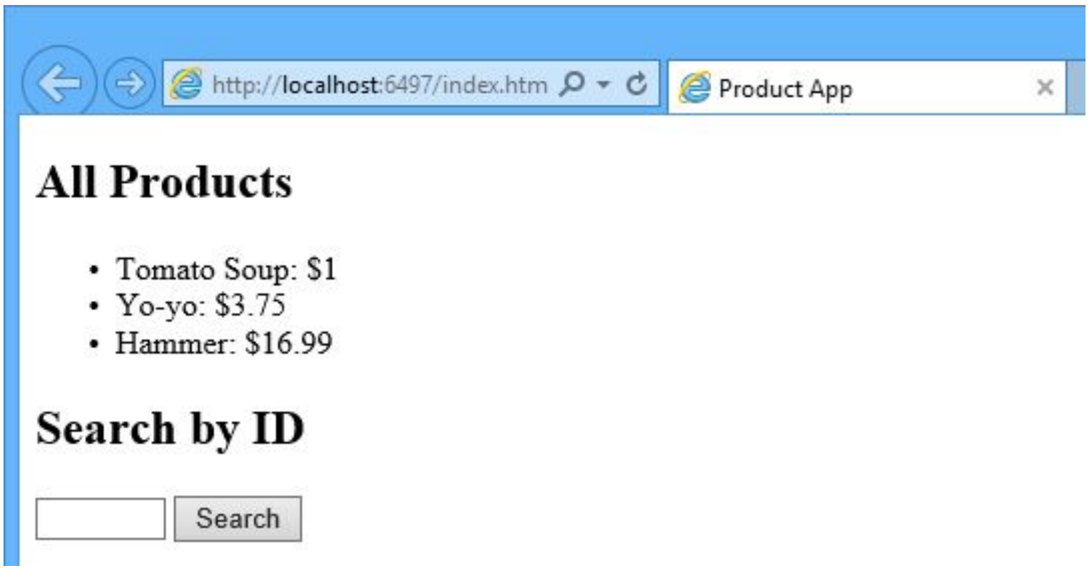
To get a product by ID, send an HTTP GET request to "/api/products/*id*", where *id* is the product ID.

```
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);
        });
}
```

We still call `getJSON` to send the AJAX request, but this time we put the ID in the request URI. The response from this request is a JSON representation of a single product.

Running the Application

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

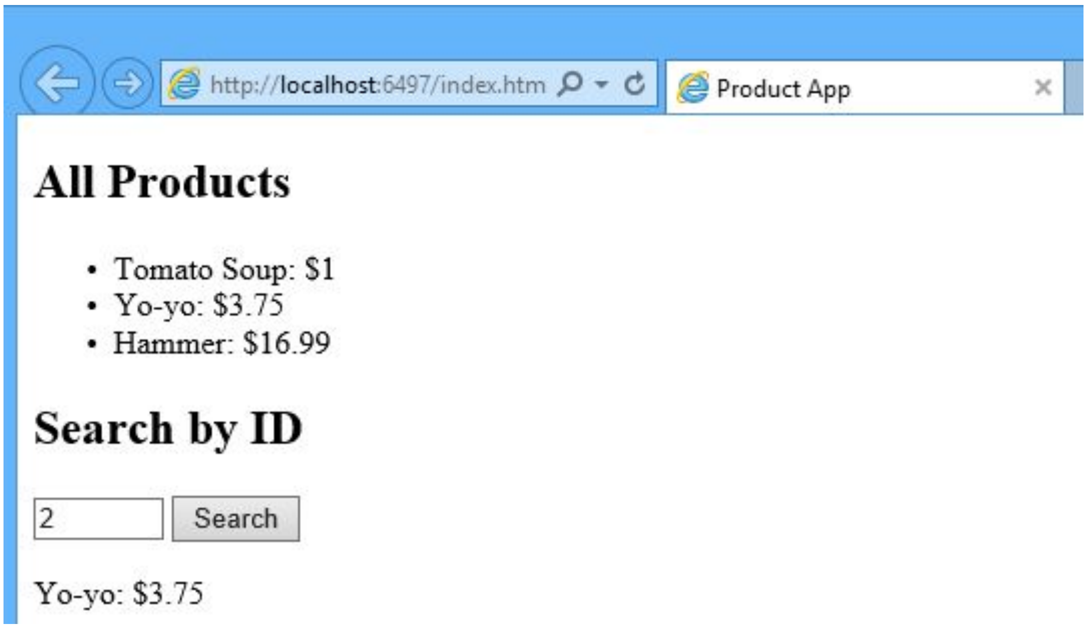


All Products

- Tomato Soup: $1
- Yo-yo: $3.75
- Hammer: $16.99

Search by ID

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



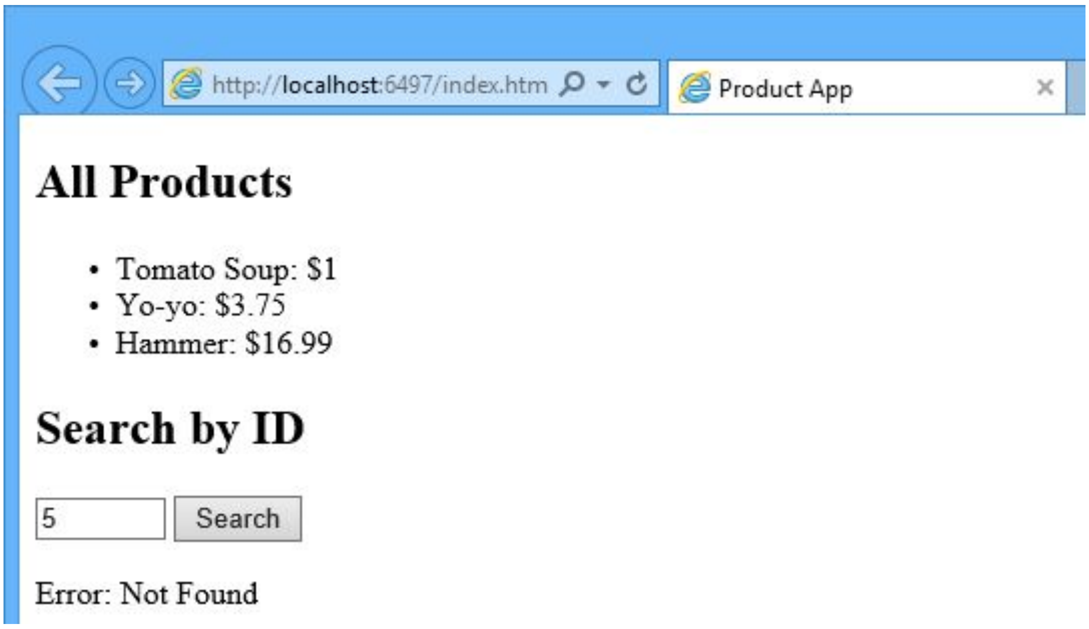
All Products

- Tomato Soup: $1
- Yo-yo: $3.75
- Hammer: $16.99

Search by ID

Yo-yo: $3.75

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



All Products

- Tomato Soup: $1
- Yo-yo: $3.75
- Hammer: $16.99

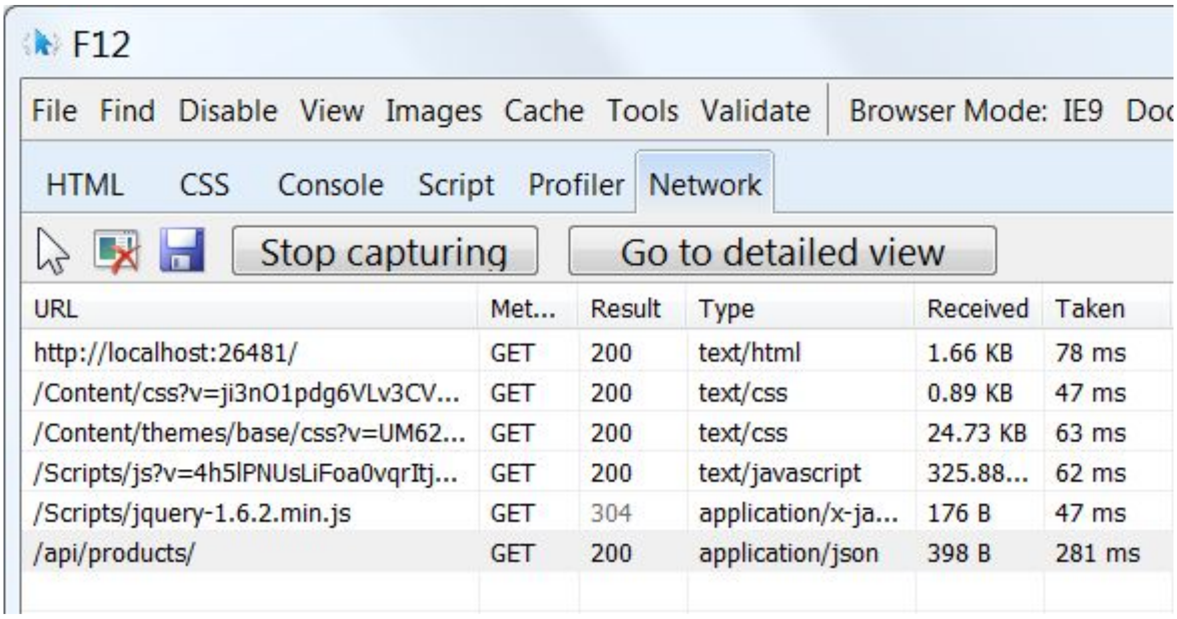
Search by ID

5 Search

Error: Not Found

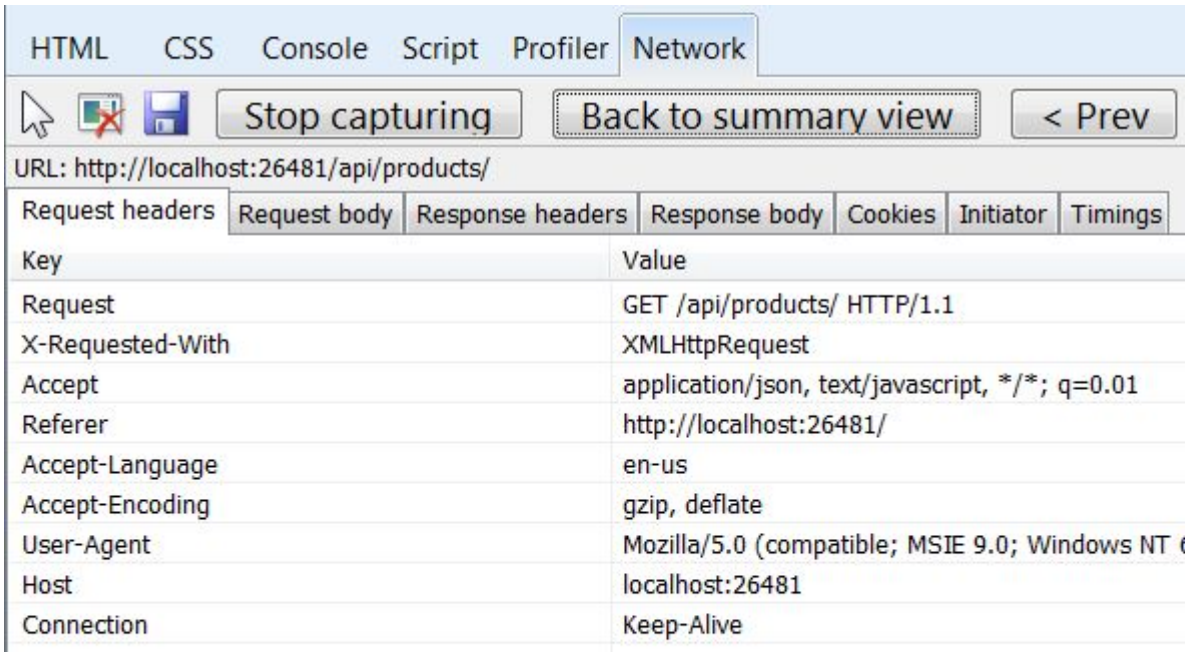
Using F12 to View the HTTP Request and Response

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:


F12
File Find Disable View Images Cache Tools Validate Browser Mode: IE9 Doc
HTML CSS Console Script Profiler Network
Stop capturing
Go to detailed view

URL	Met...	Result	Type	Received	Taken
http://localhost:26481/	GET	200	text/html	1.66 KB	78 ms
/Content/css?v=ji3nO1pdg6VLv3CV...	GET	200	text/css	0.89 KB	47 ms
/Content/themes/base/css?v=UM62...	GET	200	text/css	24.73 KB	63 ms
/Scripts/js?v=4h5lPNUsLiFoa0vqrItj...	GET	200	text/javascript	325.88...	62 ms
/Scripts/jquery-1.6.2.min.js	GET	304	application/x-ja...	176 B	47 ms
/api/products/	GET	200	application/json	398 B	281 ms

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.



URL: http://localhost:26481/api/products/

Key	Value
Request	GET /api/products/ HTTP/1.1
X-Requested-With	XMLHttpRequest
Accept	application/json, text/javascript, */*; q=0.01
Referer	http://localhost:26481/
Accept-Language	en-us
Accept-Encoding	gzip, deflate
User-Agent	Mozilla/5.0 (compatible; MSIE 9.0; Windows NT
Host	localhost:26481
Connection	Keep-Alive

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, 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.

Debugging Tips:

Enable CORS

Now let's enable CORS in the WebService app. First, add the CORS NuGet package. In Visual Studio, from the Tools menu, select Library Package Manager, then select Package Manager Console. In the Package Manager Console window, type the following command:

```
Install-Package Microsoft.AspNet.WebApi.Cors
```

This command installs the latest package and updates all dependencies, including the core Web API libraries. User the -Version flag to target a specific version. The CORS package requires Web API 2.0 or later.
Open the file App_Start/WebApiConfig.cs. Add the following code to the WebApiConfig.Register method.

```
using System.Web.Http;
namespace WebService
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            // New code
            config.EnableCors();

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

Next, add the [EnableCors] attribute to the `Controller` class:
[EnableCors(origins: "http://localhost:50565", headers: "*", methods: "*")]

https://docs.microsoft.com/en-us/aspnet/web-api/overview/security/enabling-cross-origin-requests-in-web-api

See this App Running on Azure(optional)

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.

Deploy to Azure

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

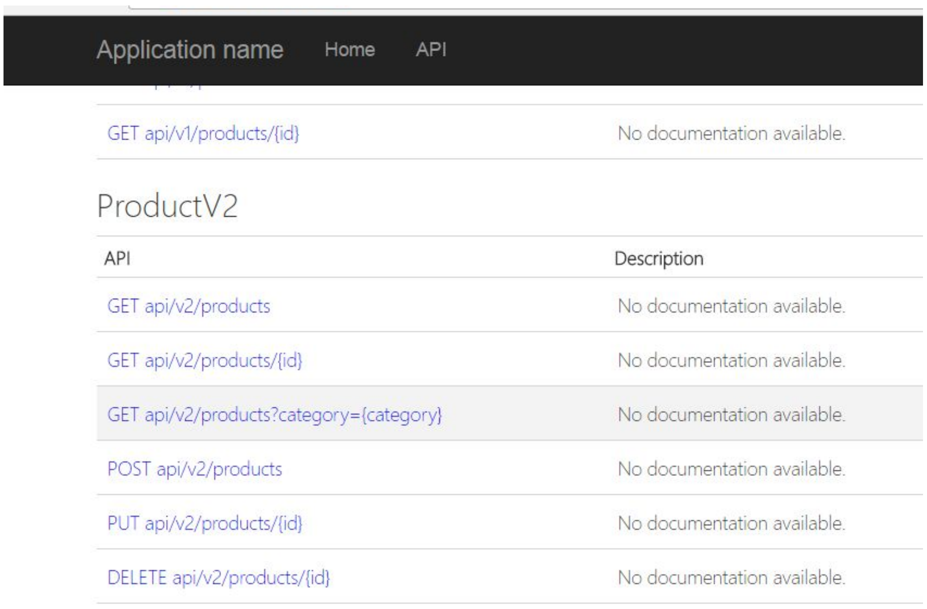
- Open an Azure account for free - You get credits you can use to try out paid Azure services, and even after they're used up you can keep the account and use free Azure services.
- Activate MSDN subscriber benefits - Your MSDN subscription gives you credits every month that you can use for paid Azure 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.
- For more about creating fluid and responsive web applications on top of an HTTP service, see 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.
   - *This article was originally created on January 20, 2014*

Working and Working:
Complete version 2 of the WEB API to :
Using Repositry to store and manage products
Add http put and delete
http://csc123.azurewebsites.net/help

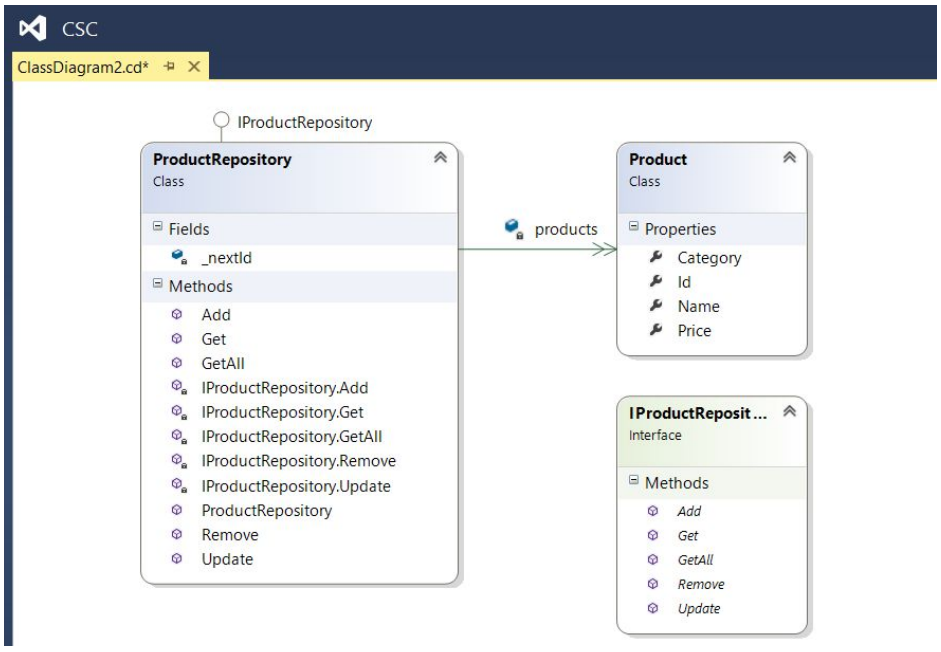


Application name Home API

GET api/v1/products/{id}	No documentation available.

ProductV2

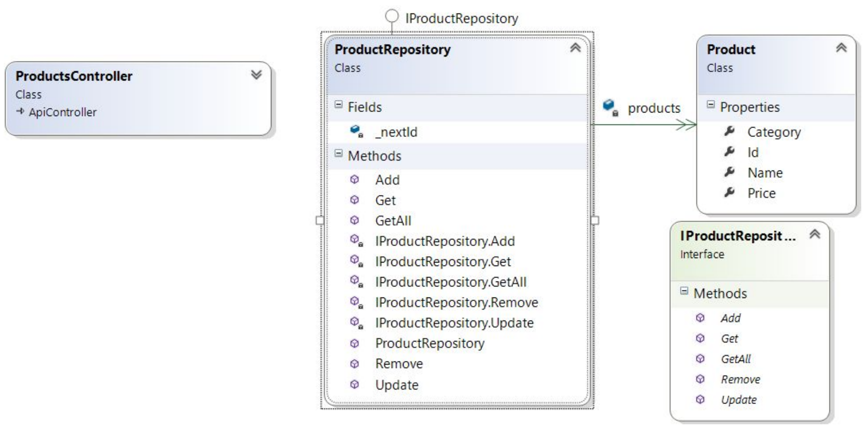
API	Description
GET api/v2/products	No documentation available.
GET api/v2/products/{id}	No documentation available.
GET api/v2/products?category={category}	No documentation available.
POST api/v2/products	No documentation available.
PUT api/v2/products/{id}	No documentation available.
DELETE api/v2/products/{id}	No documentation available.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ProductStore.Models
{
        interface IProductRepository
        {
        IEnumerable<Product> GetAll();
        Product Get(int id);
        Product Add(Product item);
        void Remove(int id);
        bool Update(Product item);
        }
    }
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace ProductStore.Models
{
        public class ProductRepository : IProductRepository

        {
        private List<Product> products = new List<Product>();
        private int _nextId = 1;

        public ProductRepository()
        {
                Add(new Product { Name = "Tomato soup", Category = "Groceries",
Price = 1.39M });
                Add(new Product { Name = "Yo-yo", Category = "Toys", Price = 3.75M
});
                Add(new Product { Name = "Hammer", Category = "Hardware", Price =
16.99M });
        }

        public IEnumerable<Product> GetAll()
        {
                return products;
        }

         public Product Get(int id)
        {
            return products.Find(p => p.Id == id);
      }

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

            item.Id = _nextId++;
            products.Add(item);
            return item;
      }

      public void Remove(int id)
      {
            products.RemoveAll(p => p.Id == id);
      }

      public bool Update(Product item)
      {
            if (item == null)
            {
            throw new ArgumentNullException("item");
            }
            int index = products.FindIndex(p => p.Id == item.Id);
            if (index == -1)
            {
             return false;
            }
            products.RemoveAt(index);
            products.Add(item);
            return true;
      }
      }
    }
```

Now Typing the code in the controller to use the repository:



Reference:
http://www.asp.net/web-api/overview/older-versions/creating-a-web-api-that-supports-crud-operations

Enabling CRUD Operations in ASP.NET Web API 2

By Mike Wasson|January 28, 2012

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 products	GET	/api/v2/products
Get a product by ID	GET	/api/v2/products/*id*
Get a product by category	GET	/api/v2/products?category=*category*
Create a new product	POST	/api/v2/products
Update a product	PUT	/api/v2/products/*id*
Delete a product	DELETE	/api/v2/products/*id*

Notice that some of the URIs include the product ID in path. For example, to get the product whose ID is 28, the client sends a GET request for`http://hostname/api/v2/products/28`.

Resources

The products API defines URIs for two resource types:

Resource	URI
The list of all the products.	/api/v2/products
An individual product.	/api/v2/products/*id*

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.

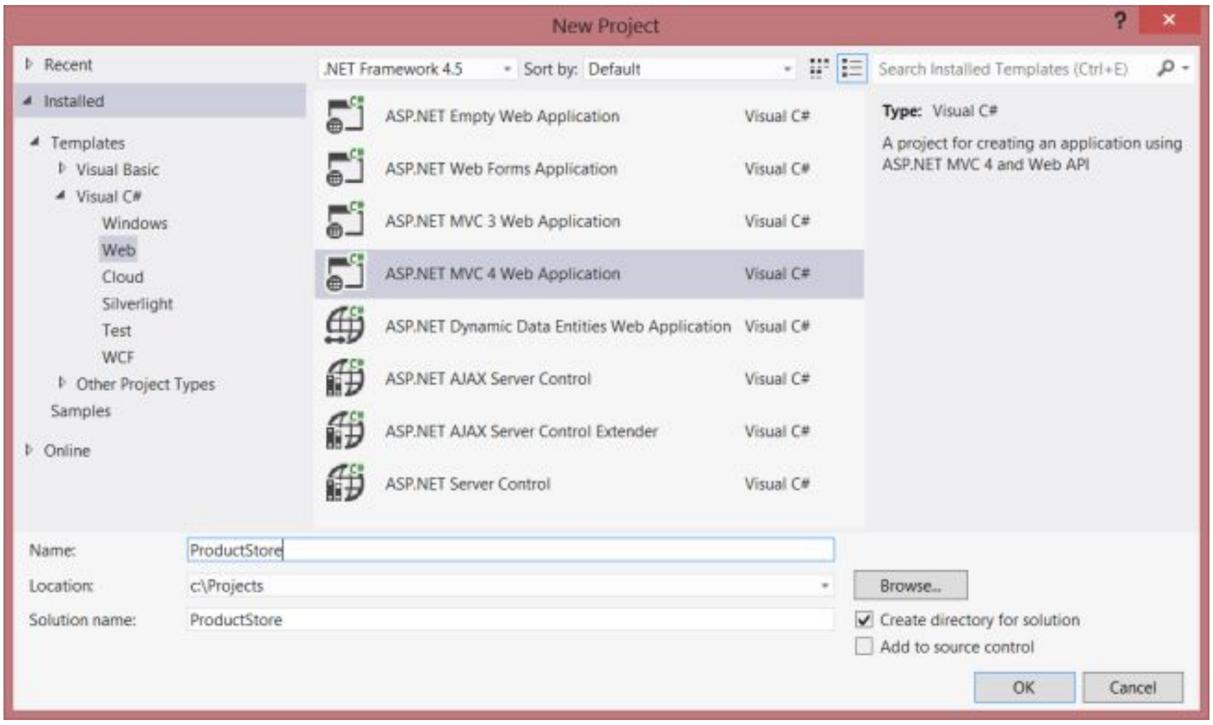
- POST creates a new resource. The server assigns the URI for the new object and returns this URI as part of the response message.
- DELETE deletes a resource at a specified URI.

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**.


New Project
Recent
Installed
Templates
Visual Basic
Visual C#
Windows
Web
Cloud
Silverlight
Test
WCF
Other Project Types
Samples
Online
.NET Framework 4.5
Sort by: Default
Search Installed Templates (Ctrl+E)
ASP.NET Empty Web Application Visual C#
ASP.NET Web Forms Application Visual C#
ASP.NET MVC 3 Web Application Visual C#
ASP.NET MVC 4 Web Application Visual C#
ASP.NET Dynamic Data Entities Web Application Visual C#
ASP.NET AJAX Server Control Visual C#
ASP.NET AJAX Server Control Extender Visual C#
ASP.NET Server Control Visual C#
Browse...
Add to source control
OK
Cancel

Type: Visual C#

A project for creating an application using ASP.NET MVC 4 and Web API

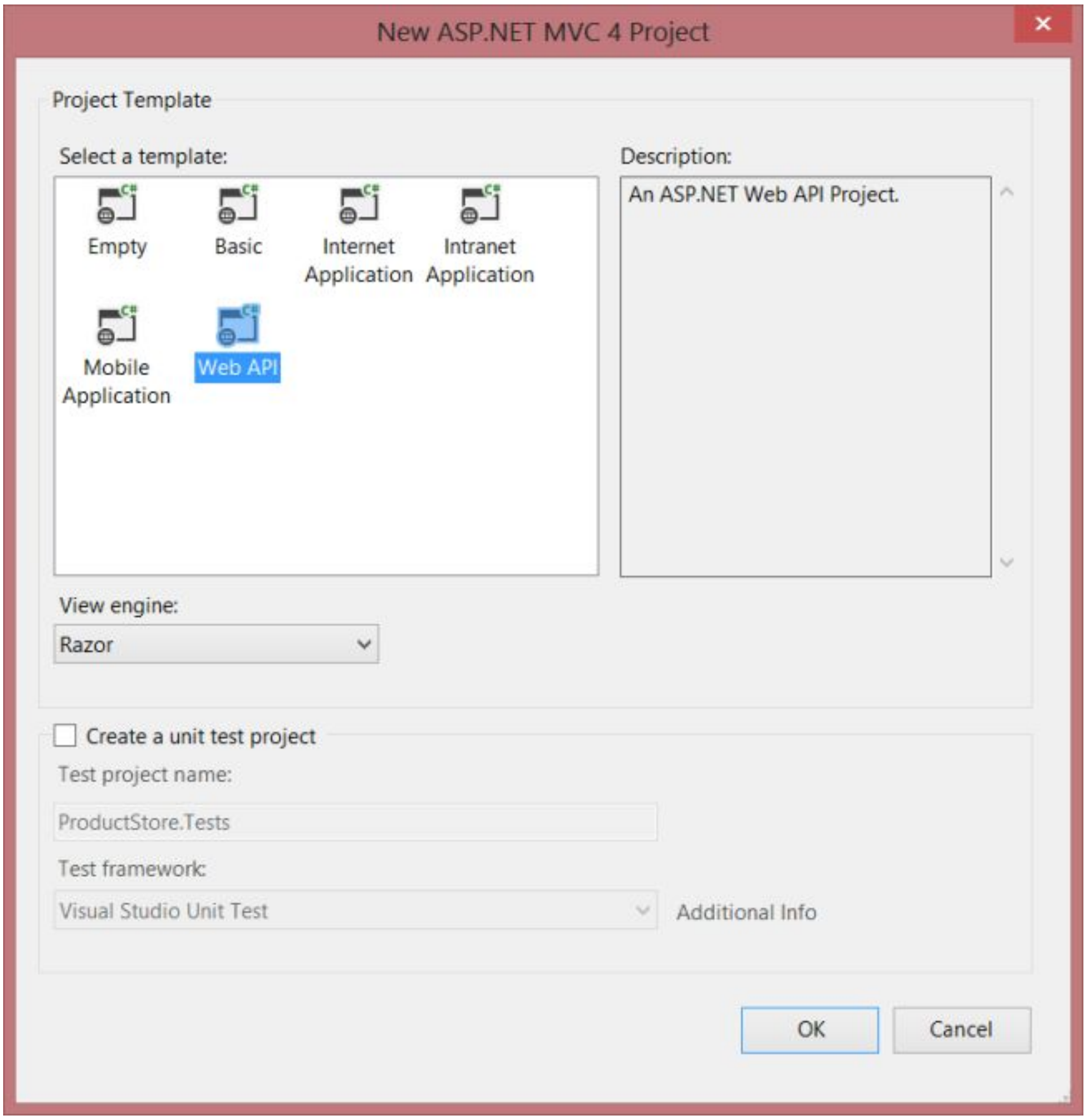
Name: ProductStore

Location: c:\Projects

Solution name: ProductStore

[✓] Create directory for solution

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


New ASP.NET MVC 4 Project

Project Template

Select a template: Empty, Basic, Internet Application, Intranet Application, Mobile Application, Web API (selected)

Description: An ASP.NET Web API Project.

View engine: Razor

Create a unit test project (unchecked)

Test project name: ProductStore.Tests

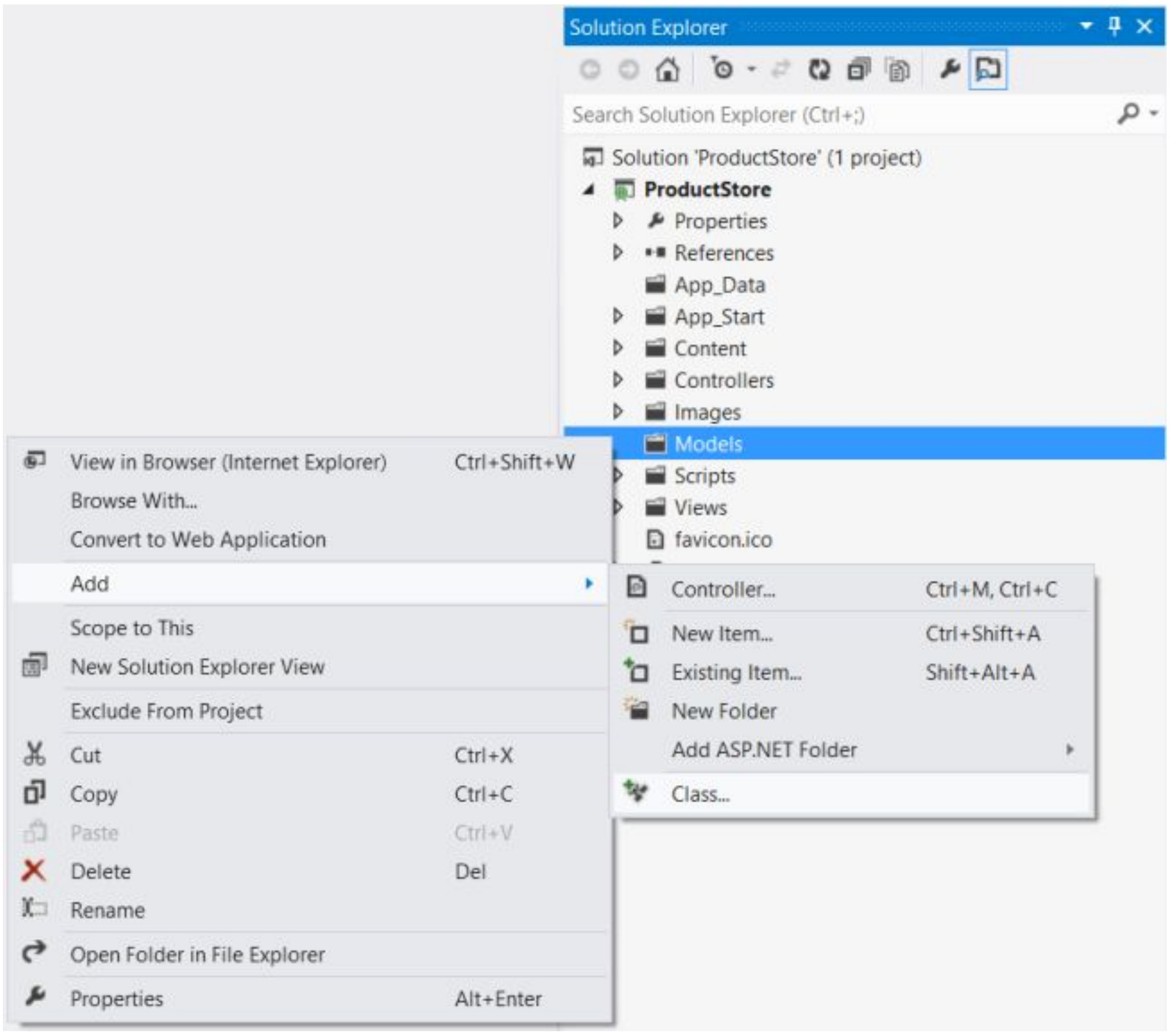
Test framework: Visual Studio Unit Test

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 `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 meny, select **Add**, then select **Class**. Name the class "Product".



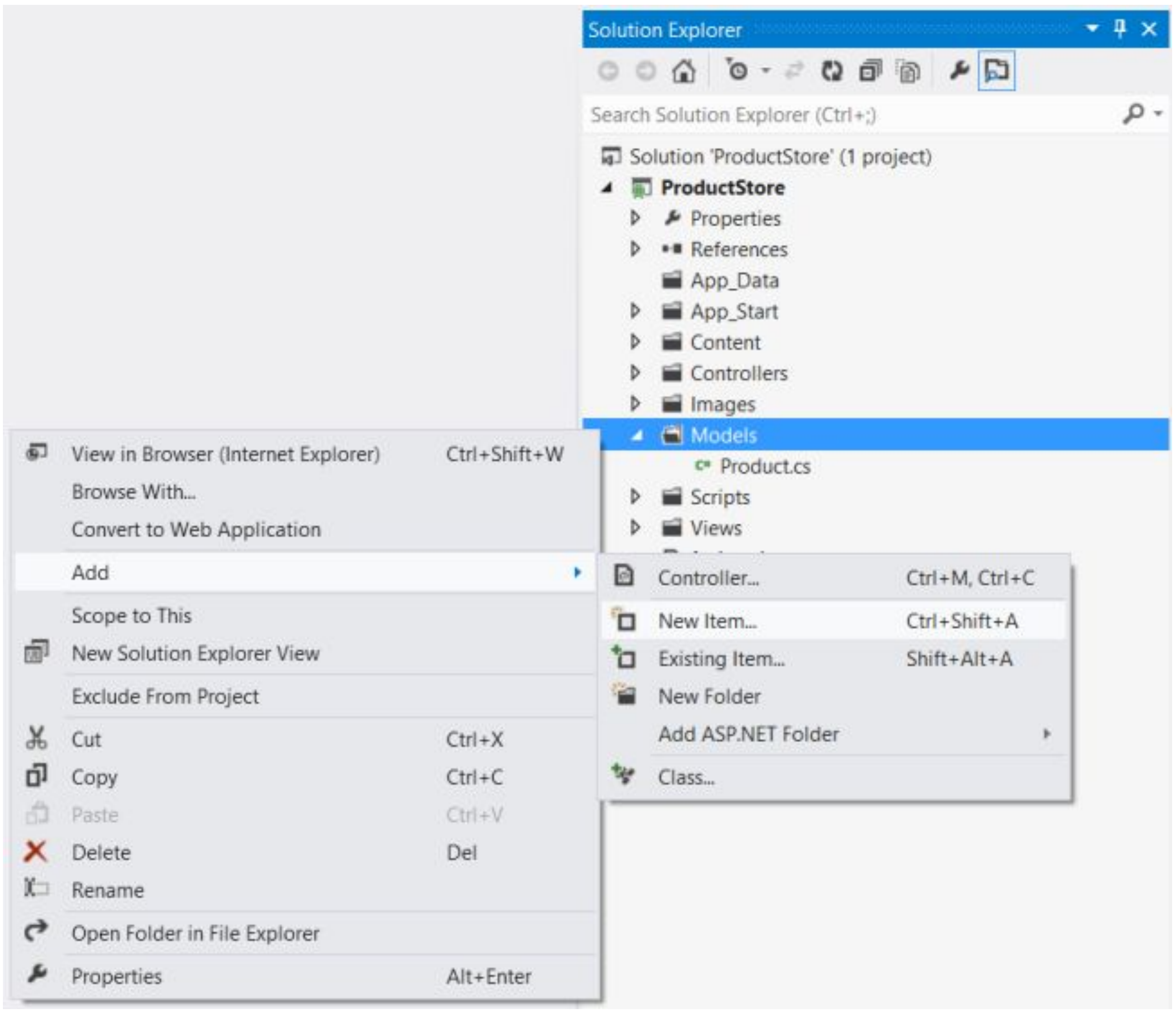
Add the following properties to the `Product` class.

```
namespace ProductStore.Models
{
    public class Product
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Category { get; set; }
        public decimal Price { get; set; }
    }
}
```

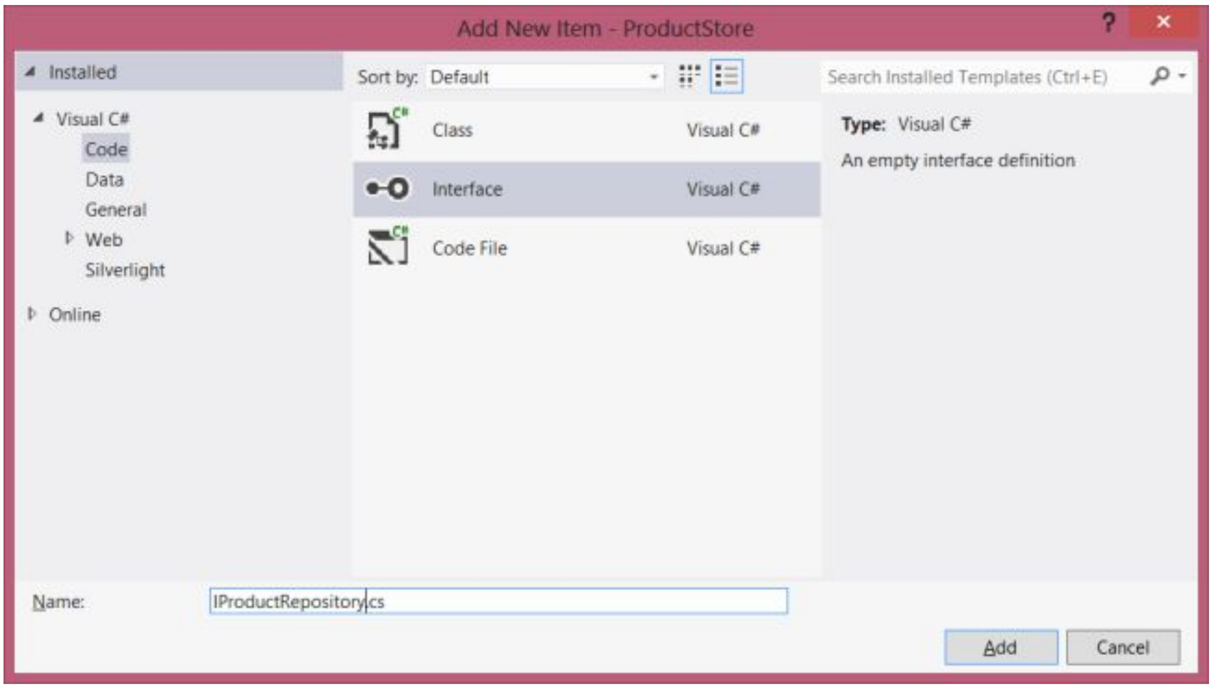
Adding a Repository

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**.



In the **Templates** pane, select **Installed Templates** and expand the C# node. Under C#, select **Code**. In the list of code templates, select **Interface**. Name the interface "IProductRepository".



Type: Visual C#

An empty interface definition

Name: IProductRepository.cs

Add the following implementation:

```
namespace ProductStore.Models
{
    public interface IProductRepository
    {
        IEnumerable<Product> GetAll();
        Product Get(int id);
        Product Add(Product item);
        void Remove(int id);
        bool Update(Product item);
    }
}
```

Now add another class to the Models folder, named "**ProductRepository.**" This class will implement the `IProductRespository` interface. Add the following implementation:

```
namespace ProductStore.Models
{
    public class ProductRepository : IProductRepository
    {
        private List<Product> products = new List<Product>();
        private int _nextId = 1;

        public ProductRepository()
        {
            Add(new Product { Name = "Tomato soup", Category = "Groceries", Price
= 1.39M });
            Add(new Product { Name = "Yo-yo", Category = "Toys", Price = 3.75M });
            Add(new Product { Name = "Hammer", Category = "Hardware", Price =
16.99M });
        }
        public IEnumerable<Product> GetAll()
        {
            return products;
        }

        public Product Get(int id)
        {
            return products.Find(p => p.Id == id);
        }

        public Product Add(Product item)
        {
            if (item == null)
            {
                throw new ArgumentNullException("item");
            }
            item.Id = _nextId++;
            products.Add(item);
            return item;
        }

        public void Remove(int id)
        {
            products.RemoveAll(p => p.Id == id);
        }

        public bool Update(Product item)
        {
            if (item == null)
            {
                throw new ArgumentNullException("item");
            }
            int index = products.FindIndex(p => p.Id == item.Id);
            if (index == -1)
            {
                return false;
            }
            products.RemoveAt(index);
            products.Add(item);
            return true;
        }
    }
}
```

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.

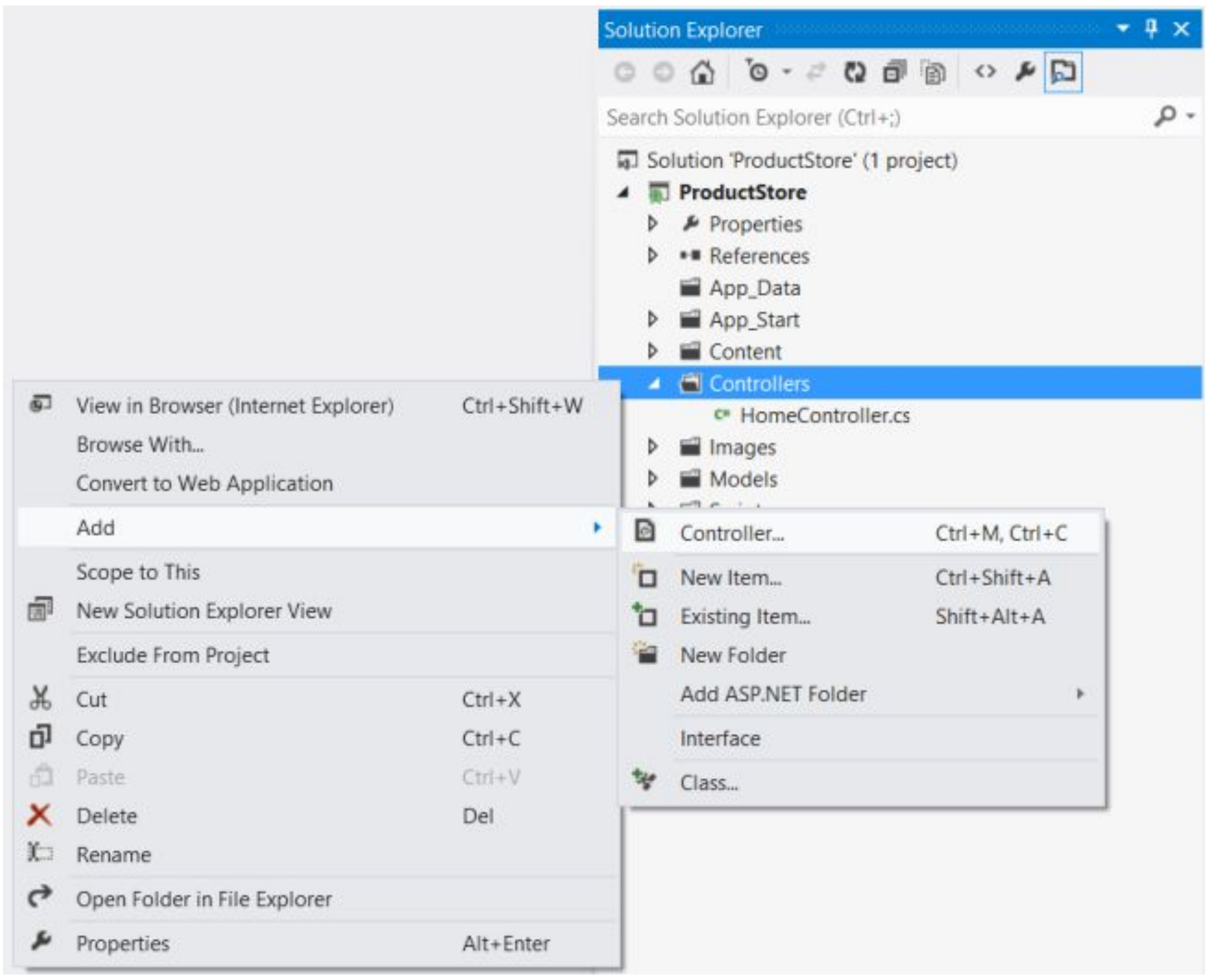
Adding a Web API Controller

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.

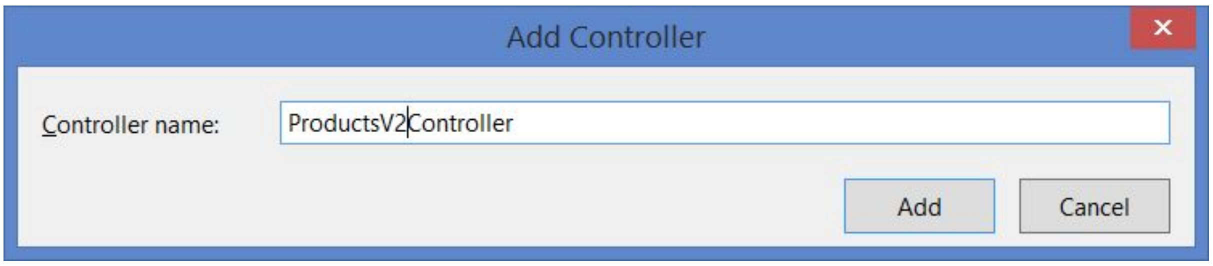
- HomeController is a traditional ASP.NET MVC controller. It is responsible for serving HTML pages for the site, and is not directly related to our web API.
- ValuesController is an example WebAPI controller.

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**.



Controller name: ProductsV2Controller

It is not necessary to put your contollers 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 **ProductsV2Controller.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 **IProductRepository** instance.

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

Calling `new ProductRepository()` in the controller is not the best design, because it ties the controller to a particular implementation of `IProductRepository`. 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`ProductsV2Controller` class.

Action	HTTP method	Relative URI
Get a list of all products	GET	/api/V2/products
Get a product by ID	GET	/api/V2/products/*id*
Get a product by category	GET	/api/V2/products?category=*category*

To get the list of all products, add this method to the `ProductsController` class:

```
public class ProductsController : ApiController
{
    public IEnumerable<Product> GetAllProducts()
    {
        return repository.GetAll();
    }
    // ....
}
```

The method name starts with "Get", so by convention it maps to GET requests. Also, because the method has no parameters, it maps to a URI that does not contain an *"id"* segment in the path.

To get a product by ID, add this method to the `ProductsController` class:

```
public Product GetProduct(int id)
{
    Product item = repository.Get(id);
    if (item == null)
    {
        throw new HttpResponseException(HttpStatusCode.NotFound);
    }
    return item;
}
```

This method name also starts with "Get", but the method has a parameter named *id*. This parameter is mapped to the "id" segment of the URI path. The ASP.NET Web API framework automatically converts the ID to the correct data type (**int**) for the parameter.

The GetProduct method throws an exception of type **HttpResponseException** if *id* is not valid. This exception will be translated by the framework into a 404 (Not Found) error.

Finally, add a method to find products by category:

```
public IEnumerable<Product> GetProductsByCategory(string category)
{
    return repository.GetAll().Where(
        p => string.Equals(p.Category, category,
StringComparison.OrdinalIgnoreCase));
}
```

If the request URI has a query string, Web API tries to match the query parameters to parameters on the controller method. Therefore, a URI of the form "api/products?category=*category*" will map to this method.

Creating a Resource

Next, we'll add a method to the `ProductsController` class to create a new product. Here is a simple implementation of the method:

```
// Not the final implementation!
public Product PostProduct(Product item)
{
    item = repository.Add(item);
    return item;
}
```

Note two things about this method:

- The method name starts with "Post...". To create a new product, the client sends an HTTP POST request.
- The method takes a parameter of type Product. In Web API, parameters with complex types are deserialized from the request body. Therefore, we expect the client to send a serialized representation of a product object, in either XML or JSON format.

This implementation will work, but it is not quite complete. Ideally, we would like the HTTP response to include the following:

- **Response code:** By default, the Web API framework sets the response status code to 200 (OK). But according to the HTTP/1.1 protocol, when a POST request results in the creation of a resource, the server should reply with status 201 (Created).
- **Location:** When the server creates a resource, it should include the URI of the new resource in the Location header of the response.

ASP.NET Web API makes it easy to manipulate the HTTP response message. Here is the improved implementation:

```
public HttpResponseMessage PostProduct(Product item)
{
    item = repository.Add(item);
    var response = Request.CreateResponse<Product>(HttpStatusCode.Created, item);

    string uri = Url.Link("DefaultApi", new { id = item.Id });
    response.Headers.Location = new Uri(uri);
    return response;
}
```

Notice that the method return type is now **HttpResponseMessage**. By returning an **HttpResponseMessage** instead of a Product, we can control the details of the HTTP response message, including the status code and the Location header.

The **CreateResponse** method creates an **HttpResponseMessage** and automatically writes a serialized representation of the Product object into the body fo the response message.

This example does not validate the `Product`. For information about model validation, see Model Validation in ASP.NET Web API.

Updating a Resource

Updating a product with PUT is straightforward:

```
public void PutProduct(int id, Product product)
{
    product.Id = id;
    if (!repository.Update(product))
    {
        throw new HttpResponseException(HttpStatusCode.NotFound);
    }
}
```

The method name starts with "Put...", so Web API matches it to PUT requests. The method takes two parameters, the product ID and the updated product. The *id* parameter is taken from the URI path, and the *product* parameter is deserialized from the request body. By default, the ASP.NET Web API framework takes simple parameter types from the route and complex types from the request body.

Deleting a Resource

To delete a resourse, define a "Delete..." method.

```
public void DeleteProduct(int id)
    {
        Product item = repository.Get(id);
        if (item == null)
        {
            throw new HttpResponseException(HttpStatusCode.NotFound);
        }

        repository.Remove(id);
    }
```

If a DELETE request succeeds, it can return status 200 (OK) with an entity-body that describes the status; status 202 (Accepted) if the deletion is still pending; or status 204 (No Content) with no entity body. In this case, the `DeleteProduct` method has a `void` return type, so ASP.NET Web API automatically translates this into status code 204 (No Content).

This article was originally created on January 28, 2012