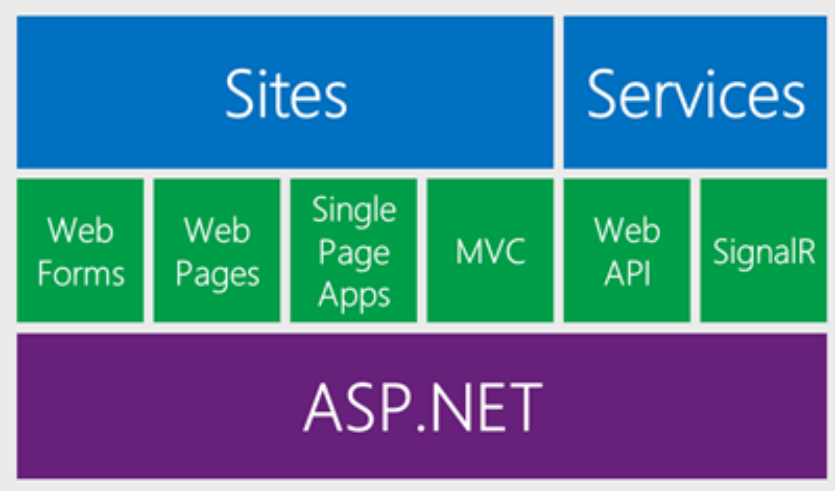
Advanced Programming 2
Recitation 10 – Web Applications Server Side Part I

Roi Yehoshua
2017

ASP.NET

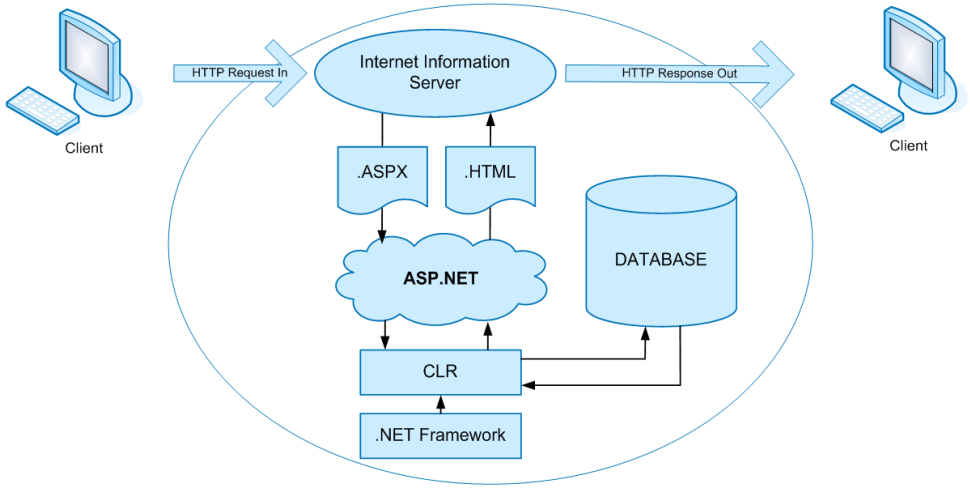
ASP.NET

- ASP.NET is an open-source server-side web application framework developed by Microsoft
- ASP.NET Core was released in 2016 and merges into one application:
  - ASP.NET Web Pages
  - ASP.NET MVC
  - ASP.NET Web API – API application model



ASP.NET Web Forms

- Web forms are made up of two components:
  - the visual portion (the .aspx file), and
  - the code behind the form (the .aspx.cs file)
- Offers a rich suite of server controls
- Supports event-driven programming model



ASP.NET Web Forms

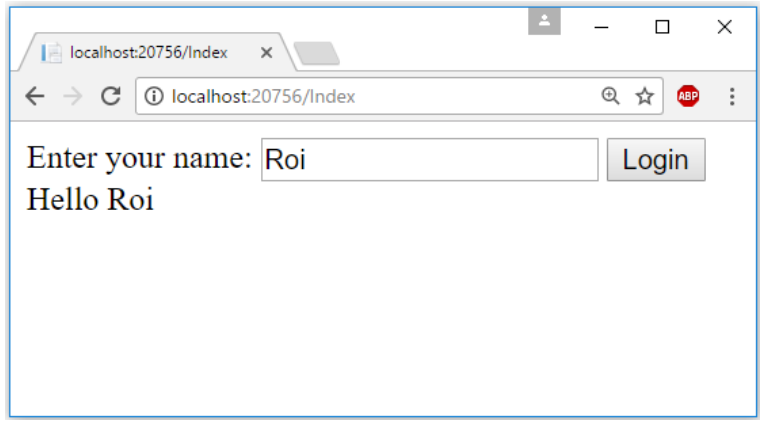
Default.aspx

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="Index.aspx.cs" Inherits="WebApplication1.Index" %>

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        Enter your name:
        <asp:TextBox ID="txtUserName" runat="server"></asp:TextBox>
        <asp:Button ID="btnLogin" runat="server"
OnClick="btnLogin_Click" Text="Login" />
        <br />
        <asp:Label ID="lblUserName" runat="server"
Text="Label"></asp:Label>
    </div>
    </form>
</body>
</html>
```

Default.aspx.cs

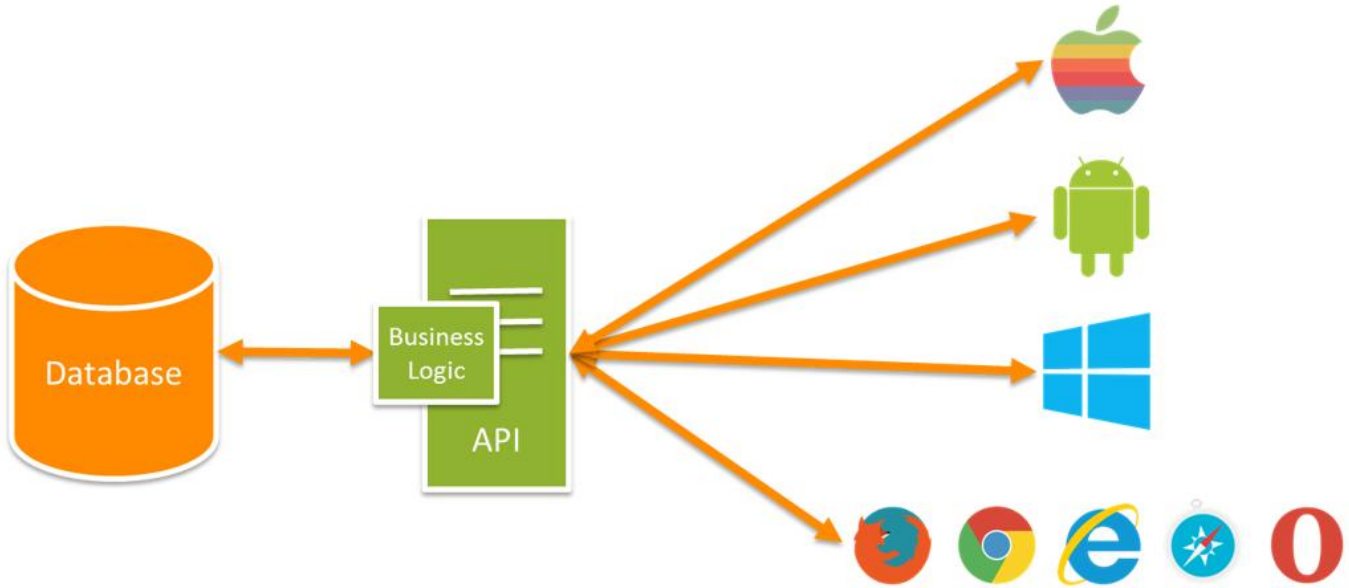
```
public partial class Index : System.Web.UI.Page
{
    protected void btnLogin_Click(object sender,
EventArgs e)
    {
        lblUserName.Text = "Hello: " +
txtUserName.Text;
    }
}
```



Web API

ASP.NET Web API

- A platform for building RESTful services using the .NET framework
- Reaches broad range of clients, including browsers and mobile devices
- Each app uses the same API to get, update and manipulate data
- The apps themselves then become relatively lightweight UI layers

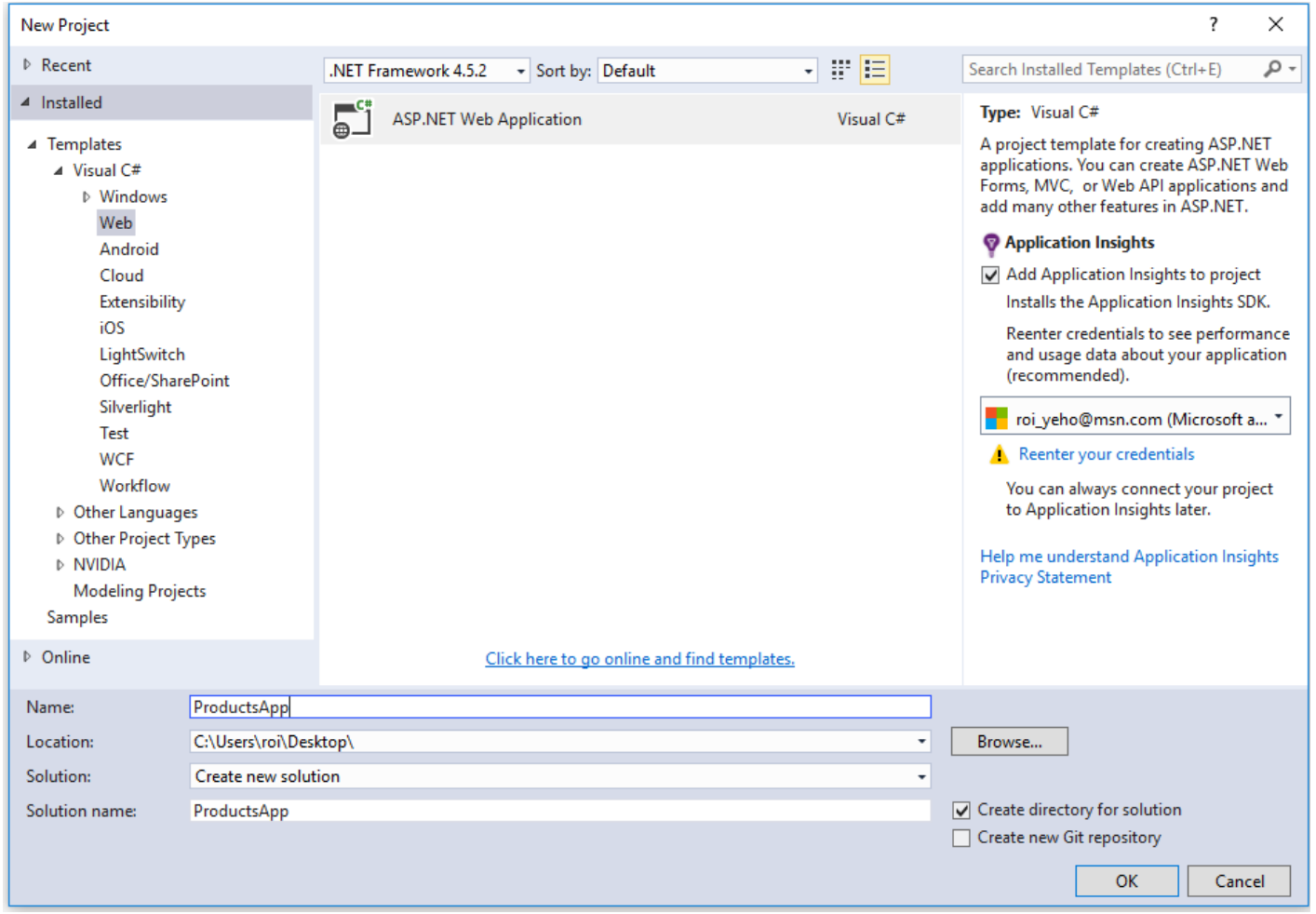


What is REST?

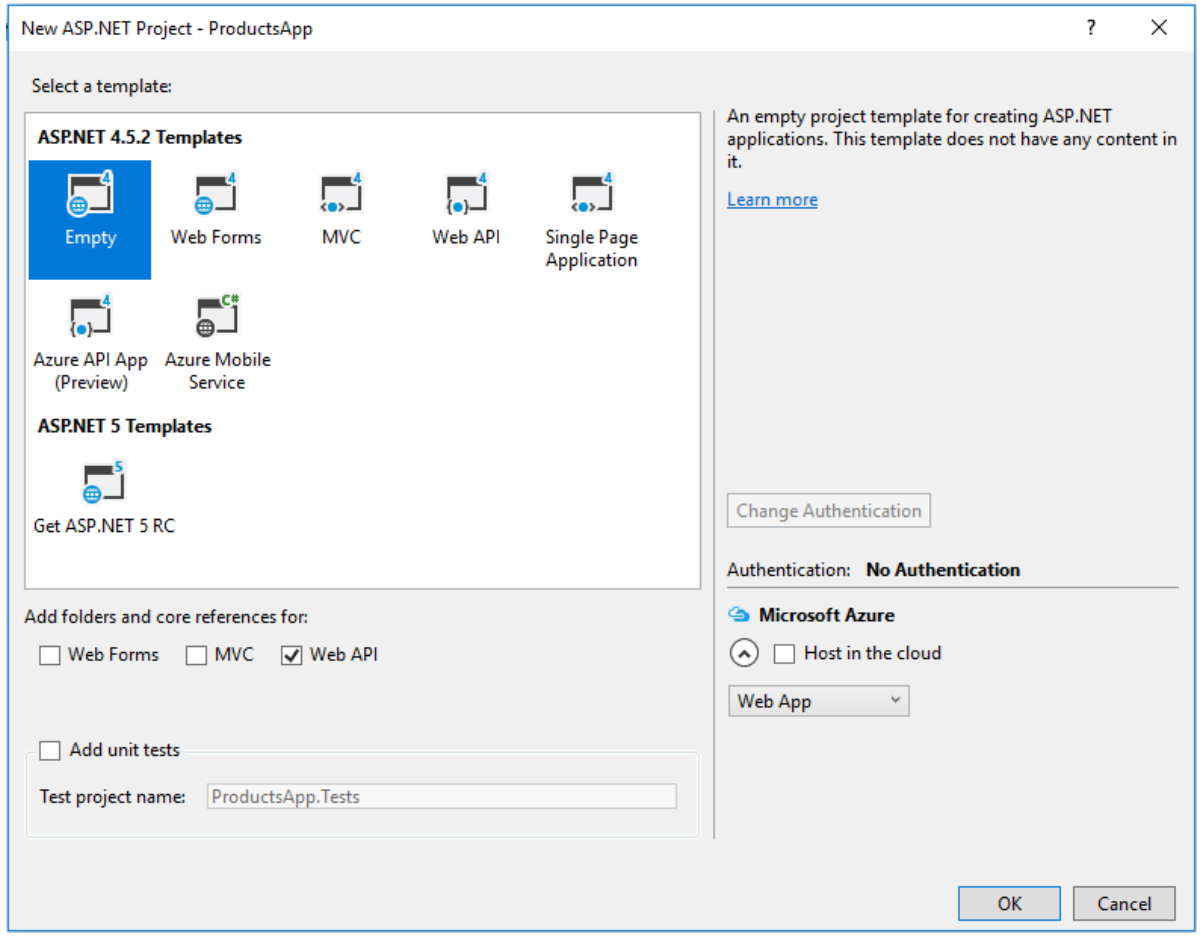
- REST is an architectural pattern for creating an API that uses HTTP as its underlying communication method
- Resources typically represent the data entities (i.e. ‘Product’, ‘Order’)
- The HTTP verb that is sent with the request informs the API what to do with the resource, e.g.:

Resource	Verb	Expected Outcome	Response Code
/Products	GET	A list of all products in the system	200/OK
/Products?Colour=red	GET	A list of all products in the system where the colour is red	200/OK
/Products	POST	Creation of a new product	201/Created
/Products/81	GET	Product with ID of 81	200/OK
/Products/881(a product ID which does not exist)	GET	Some error message	404/Not Found
/Products/81	PUT	An update to the product with an ID of 81	204/No Content
/Products/81	DELETE	Deletion of the product with an ID of 81	204/No Content
/Customers	GET	A list of all customers	200/OK

Creating a Web API Project

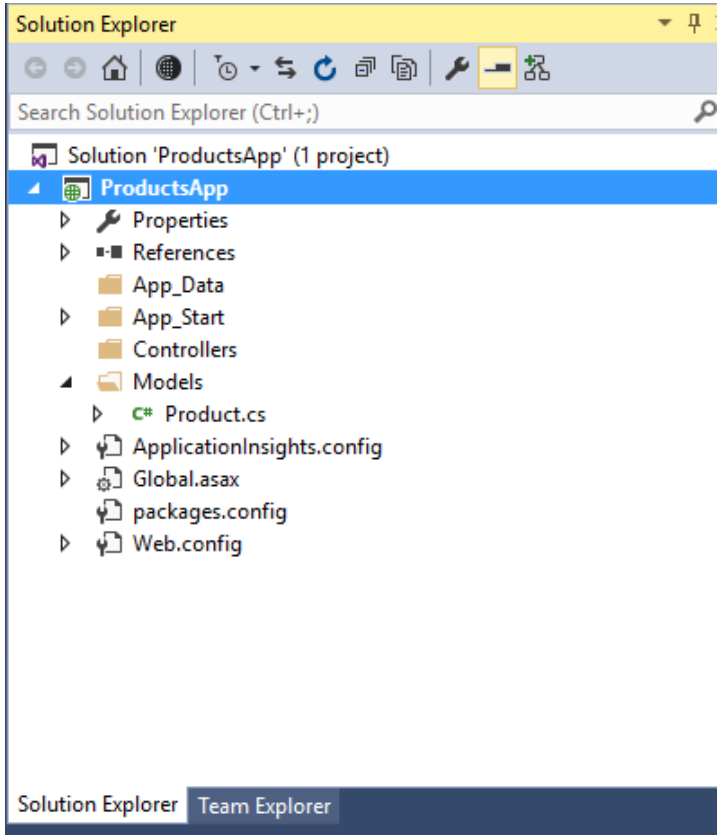


Creating a Web API Project



Web API Project Structure

- App_Data – contains the data files (e.g., .mdf files)
- App_Start – contains startup tasks (e.g., defining the routing table)
- Controllers – the Controller layer
- Models – the Models layer
- Global.asax - an optional file that contains code for responding to application-level events raised by ASP.NET, e.g., Application_Start
- Web.config – the main settings and configuration file for an ASP.NET web application



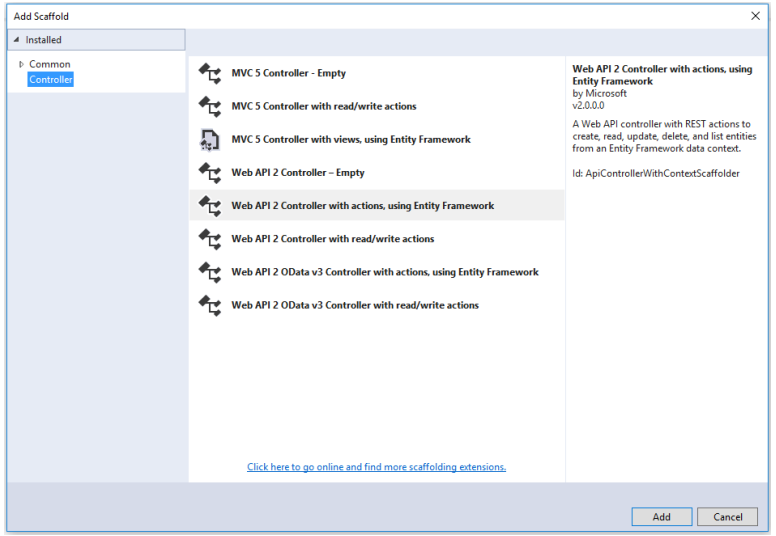
Adding a Model

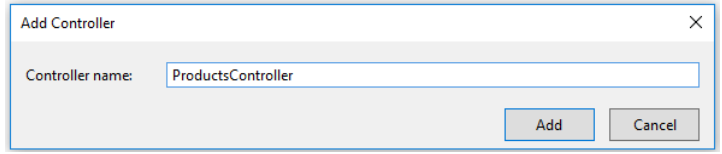
- A **model** is an object that represents the data in your application
- ASP.NET Web API can automatically serialize your model to JSON, XML, or some other format, and write the serialized data into the HTTP response message.
- The client can indicate which format it wants by setting the **Accept** header in the HTTP request message
- Let's create a simple model that represents a product
  - In Solution Explorer, right-click the Models folder
  - From the context menu, select **Add** then select **Class**

```
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

- A Web API **controller** is an object that handles HTTP requests
- You would typically have a different controller for each of your main data entities (Product, Person, Order etc)
- The public methods of the controller are called **actions**
- The .net routing engine to map URLs to controllers and actions
- We'll add a products controller that will enable the user to get the list of products or add a new product
  - 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**.
  - In the **Add Controller** dialog, name the controller "ProductsController". Click **Add**.
  - The scaffolding creates a file named ProductsController.cs in the Controllers folder.





Adding a Controller

If the name of the action starts the words "Get", "Post", "Put", etc. use the corresponding HTTP method

If there is an attribute applied via [HttpGet], [HttpPost], [HttpPut], etc, the action will accept the specified HTTP method

```
public class ProductsController : ApiController
{
    private static List<Product> products = new List<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)
    {
        Product product = products.FirstOrDefault(p => p.Id == id);
        if (product == null)
        {
            return NotFound();
        }
        return Ok(product);
    }
    [HttpPost]
    public void AddProduct(Product p)
    {
        products.Add(p);
    }
}
```

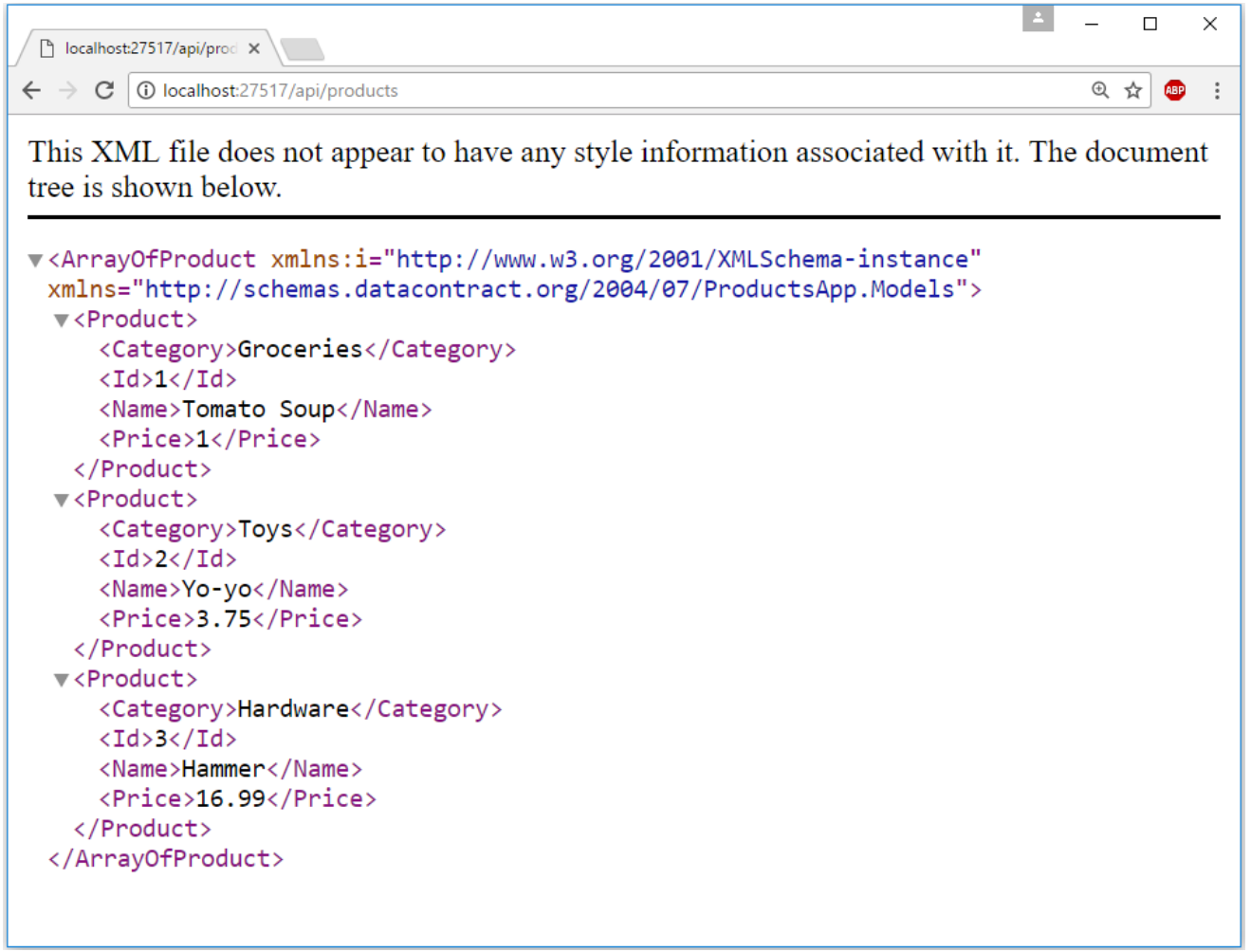
Routing and Action Selection

- To determine which action to invoke, the framework uses a **routing table**
- The Visual Studio project template for Web API creates a default route:

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

- Each entry in the routing table contains a **route template**
- The default route template for Web API is "api/{controller}/{id}“
  - "api" is a literal path segment, and {controller} and {id} are placeholder variables
- For example, the following URIs match the default route:
  - /api/contacts, /api/contacts/1, /api/products/gizmo1
- To find the controller, Web API adds "Controller" to the value of the {controller} variable.
- To find the action, Web API looks at the HTTP method, and then looks for an action whose name begins with that HTTP method name
- Placeholder variables such as {id} are mapped to action parameters

Testing the Get Actions from the Browser

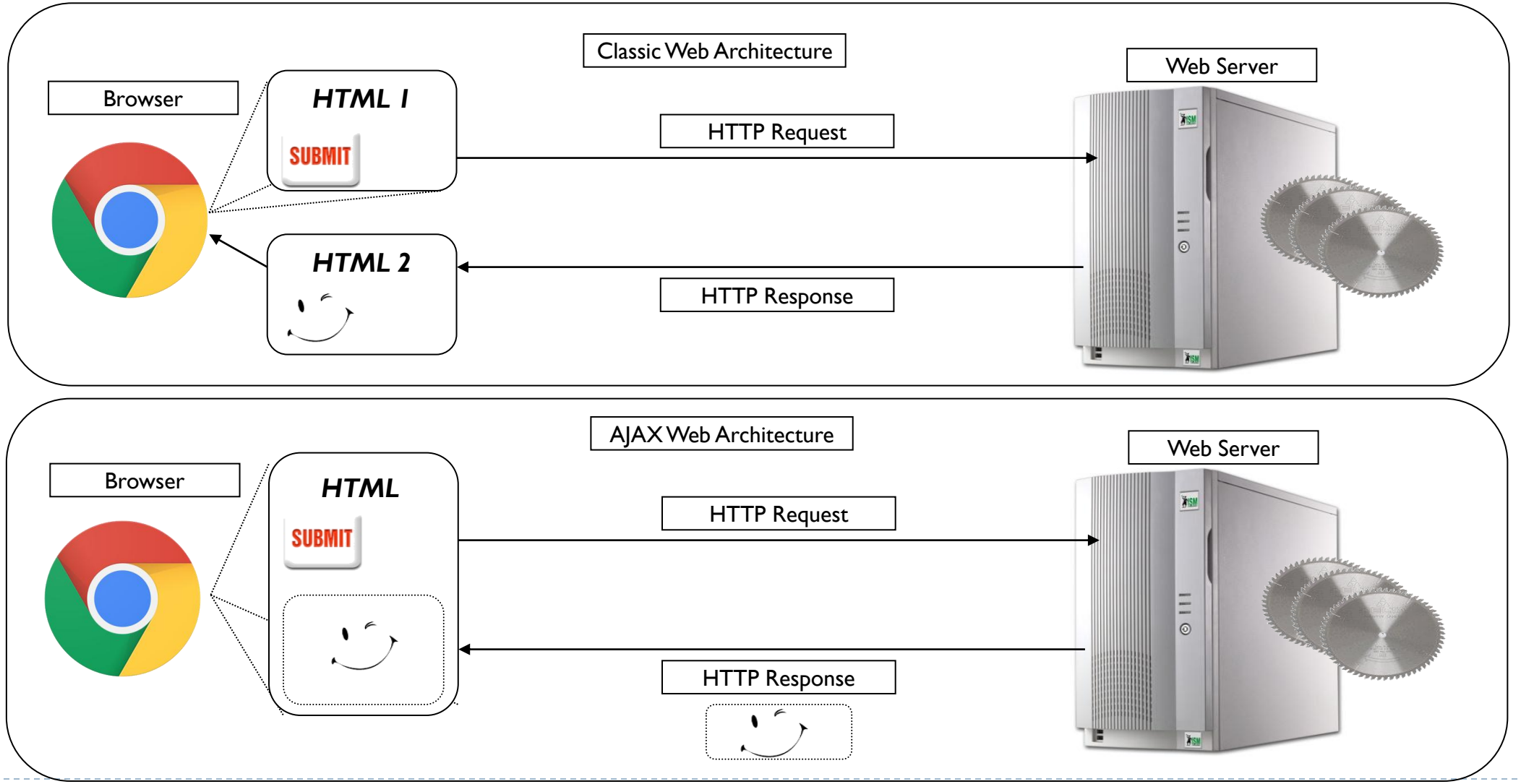


AJAX

AJAX

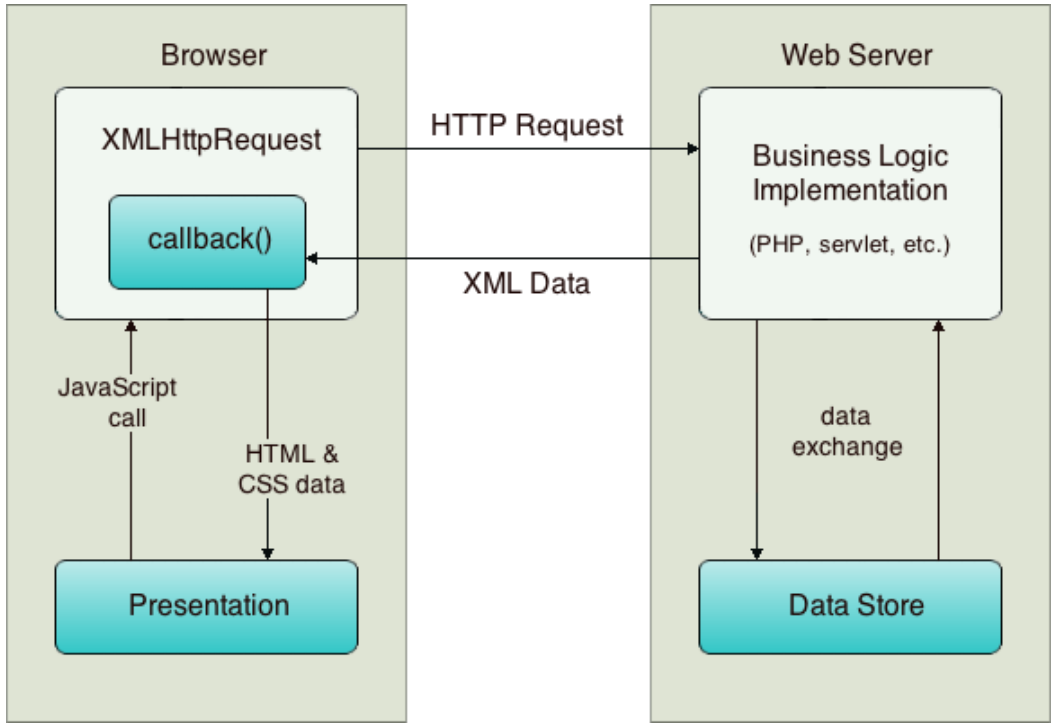
- **A**synchronous **J**avascript **A**nd **X**ML
- A **combination** of **technologies**
  - XML – data exchange (+ JSON, HTML & plain text)
  - JavaScript –XMLHTTPREQUEST object (**XHR**)
- **Send & receive** data on the **background**
  - Parts of the page are "refreshed“
  - Smoother – no need to wait for a new page to load
- Requires neither XML nor async requests
- Advantages:
  - Makes web pages more responsive
  - Instead of reloading an entire web page each time a user makes a change – only the required data is retrieved

Classic vs. Modren Web Architectures



AJAX Architecture

- **XMLHttpRequest (XHR)** is a JavaScript object that allows you to transfer data between a web browser and a web server



Generating AJAX Requests

url – the address of this ajax call. May target a Servlet or JSP. May send parameters just like any HTTP request

When a response is received stateCange() function will be asynchronously called

open() - Setting request data format:
- method (GET/POST)
- url
- asynchronous call – true enables it & is the default value

Send method takes a DOM Object.
DOM object hosts XML documents and Fragments available via DOM API.
Null value is also permitted, usually when string values are sent as a request header.

stateChange() method will be called asynchronously.
Will be explained later.

```
function generateRequest()
{
    var url="http://localhost:8080/ajax";
    url+="?command=doIt";
    xmlhttp.onreadystatechange=stateChange;
    xmlhttp.open("GET",url,true);
    xmlhttp.send();
}

function stateChange()
{
    if (xmlhttp.readyState==4)
     // ...some code here...
    else
        alert("Problem retrieving XML data")
 }
...
```

Server Side - Generating AJAX Response

- Response might be a:
  - String
  - XML Document
  - JSON String
- Generating AJAX response with servlets:

```
String command=request.getParameter (“command”);
if(command.equals(“AJAXTranslateAction”)){
        String wordToTranslate=request.getParameter(“word”);
        String translation=translate(wordToTranslate);
        PrintWriter out=response.getWriter();
        out.print(translation);
        out.flush();
}
```

Handling Server Response

- XMLHttpRequest response related methods & fields:

Method / Field	Description
onreadystatechange	Event handler for an event that fires at every state change
readyState	Object status integer: 0 = uninitialized 3 = interactive 1 = loading 4 = complete 2 = loaded
responseText	String version of data returned from server process
responseXML	DOM-compatible document object of data returned from server process
status, statusText	Numeric code & description returned by server such as 404 for "Not Found" or 200 for "OK"

Handling Server Response

- Processing a response with an XML element

stateChange() method will be called asynchronously.
It checks the 'readyState'
0- uninitialized
1- loading
2- loaded
3- interactive
4- complete

```
function generateRequest() {
    var url="http://localhost:8080/ajax";
    url+"?command=doIt";
    xmlhttp.onreadystatechange=stateChange;
    xmlhttp.open("GET",url,true);
    xmlhttp.send(null);
}

function stateChange() {
    if (xmlhttp.readyState==4)
        var doc=xmlhttp.responseXML;
        var element = doc.getElementsByTagName('root').item(0);
        alert(element.nodeName);
    else
        alert("Problem retrieving XML data")
 }
...
```

AJAX & jQuery

- **.ajax()**
  - The **main AJAX** method
  - Provides precise control over your Ajax call
    - GET or POST method
    - Error callback
    - Different formats of data

```
$.ajax({
  method: "POST",
  url: "some.php",
  data: { name: "John", location: "Boston" }
})
  .done(function( msg ) {
    alert( "Data Saved: " + msg );
  });
```

- **AJAX Shortcuts**
  - **.get()** – Load data from the server using a HTTP GET request
  - **$.getJSON()** – Gets a JSON object from the server using a HTTP GET request
  - **.post()** - Load data from the server using a HTTP POST request
  - **.load()** - Load data from the server and place the returned HTML into the matched element

AJAX Limitations

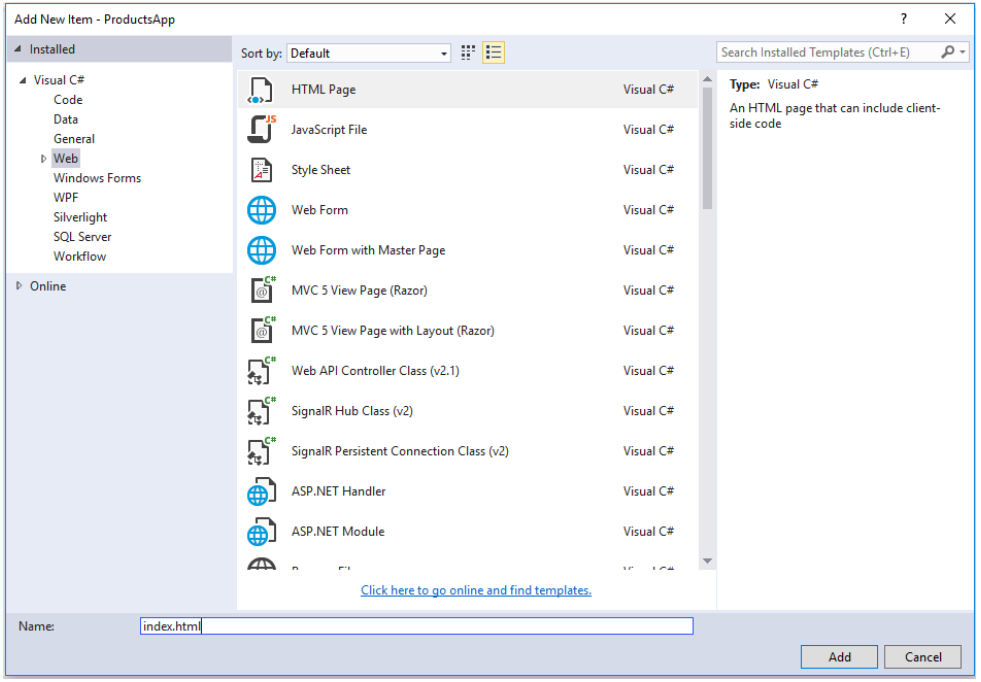
- The same-origin policy
- Prevents loading of resx from one site to another
- It prevents attacks like hijacking user's session

http://www.company.com/page.html

URL	Same origin?	Reason
http://www.company.com/-page2.html	YES	
http://www.company.com/dir/-page.html	YES	
http://blog.company.com/page.html	NO	Different host
https://www.company.com/-page2.html	NO	Different protocol
http://www.company.com:8080/-page2.html	NO	Different port

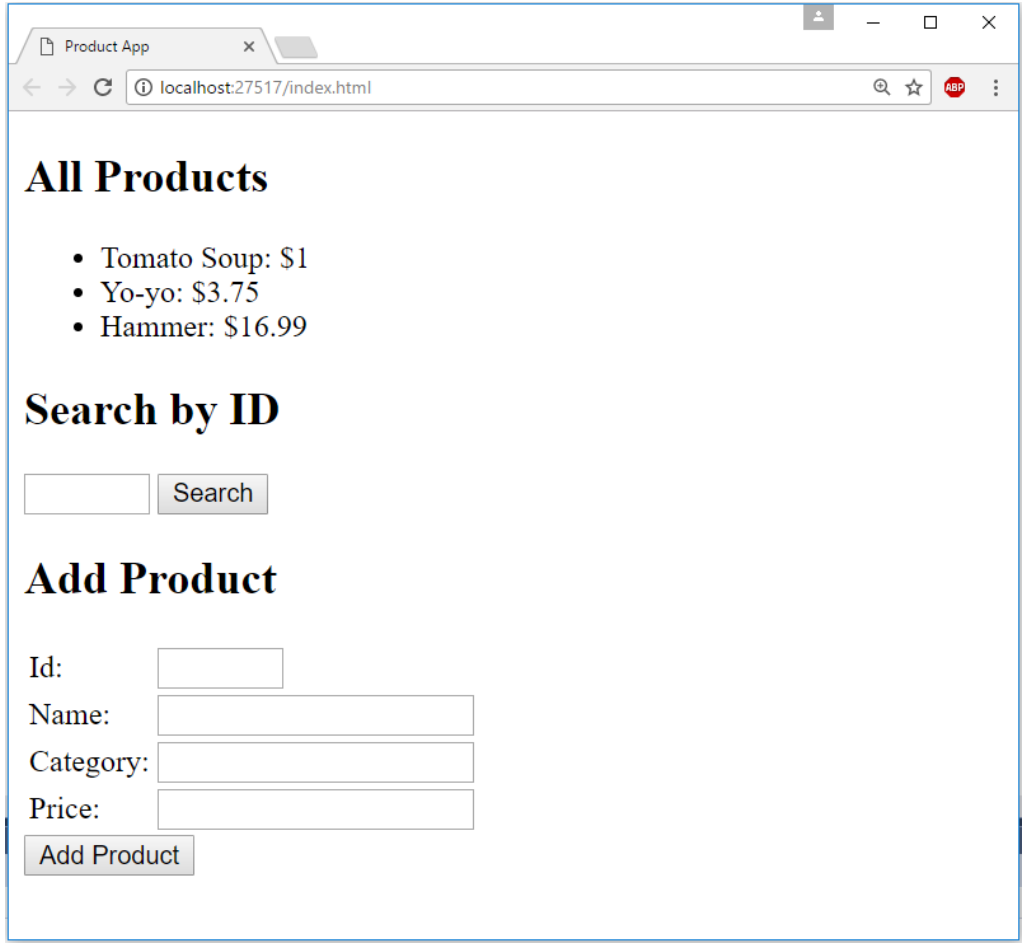
Calling the Web API with Javascript and jQuery

- In this section, we'll add an HTML page that uses AJAX to call the web API
  - 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#**
  - Then select the **HTML Page** item. Name the page "index.html".



Calling the Web API with Javascript and jQuery

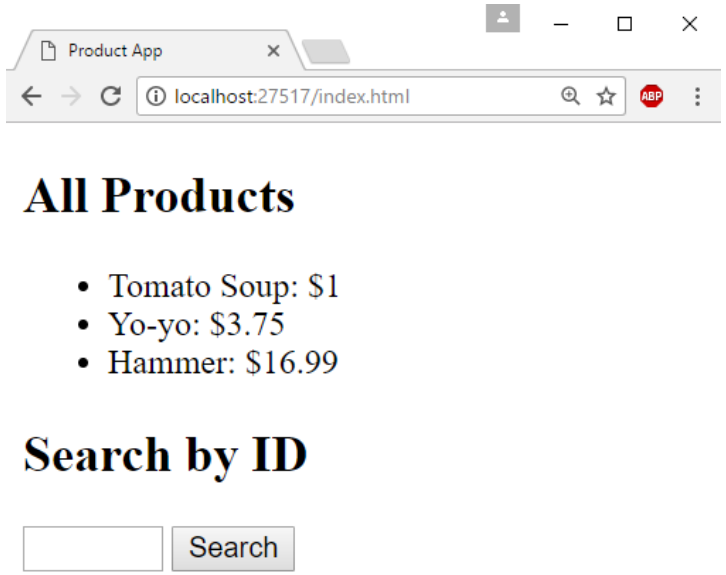
```
<!DOCTYPE html>
<html>
<head>
    <title>Product App</title>
</head>
<body>
    <div>
        <h2>All Products</h2>
        <ul id="lstProducts"></ul>
    </div>
    <div>
        <h2>Search by ID</h2>
        <input type="text" id="txtProductId" size="5" />
        <input type="button" id="btnSearch" value="Search" />
        <p id="product" />
    </div>
    <div>
        <h2>Add Product</h2>
        <table>
            <tr><td>Id: </td>
                <td><input type="text" id="prodId" size="5" /></td></tr>
            <tr><td>Name: </td>
                <td><input type="text" id="prodName" /></td></tr>
            <tr><td>Category: </td>
                <td><input type="text" id="prodCategory" /></td></tr>
            <tr><td>Price: </td>
                <td><input type="text" id="prodPrice" /></td></tr>
        </table>
        <input type="button" id="btnAddProduct" value="Add Product" />
    </div>
</body>
</html>
```



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 and returns a JSON object
- The done() function specifies a callback that is called if the request succeeds
- In the callback, we update the DOM with the product information

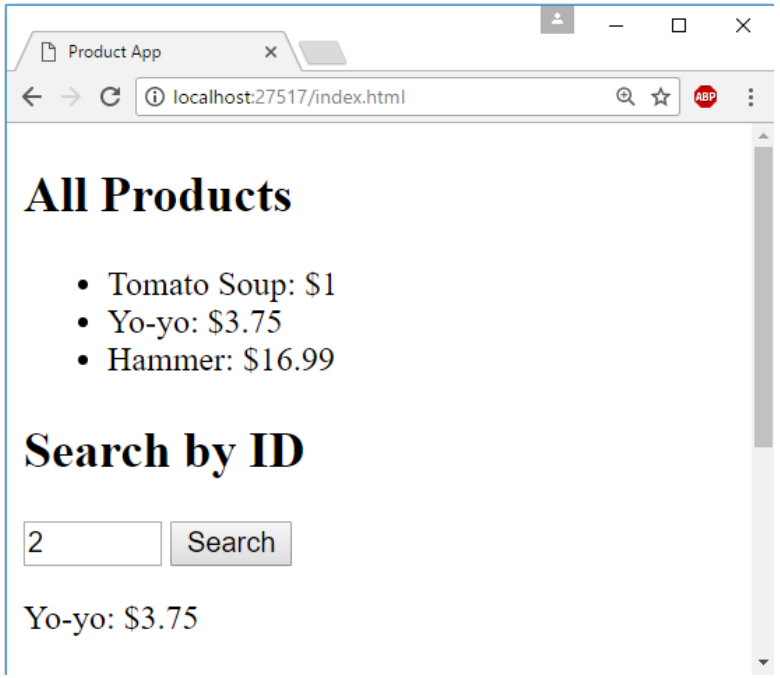
```
<script>
    var apiUrl = "api/Products";
    // Send an AJAX request
    $.getJSON(apiUrl).done(function (data) {
        data.forEach(function (product) {
            // Add a list item for the product
            $("<li>" + product.Name + ": $" + product.Price + "</li>")
                .appendTo("#lstProducts");
        });
    });
</script>
```



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
- The response from this request is a JSON representation of a single product

```
$("#btnSearch").click(function () {
    var id = $("#txtProductId").val();
    $.getJSON(apiUrl + "/" + id)
    .done(function (product) {
        $("#product").text(product.Name + ": $" + product.Price);
    })
    .fail(function (jqXHR, textStatus, err) {
        $("#product").text("Error: " + err);
    });
});
```



Adding a Product

- To add a product, send an HTTP POST request to "/api/products“
- The body of the HTTP request is a string representation of the new product

```
$("#btnAddProduct").click(function () {
    var product = {
        Id: $("#prodId").val(),
        Name: $("#prodName").val(),
        Category: $("#prodCategory").val(),
        Price: $("#prodPrice").val()
    };

    $.post(apiUrl, product)
    .done(function () {
        alert("Product added successfully");
    });
});
```