Sheridan | Get Creative

Advanced .NET Server Development: Web API

Instructor: Amandeep S. Patti

Objectives

- Intro to Web API
- Web API Controllers
- Web API Routing
- Web API Model Binding
- Web API Content Negotiation
- Building a Web API Service
- Hosting a Web API Service
- Consuming a Web API Service from .NET
- Consuming a Web API Service from JavaScript

Introduction to Web API

- New method for building HTTP Web Services
- Great support for XML, JSON and RESTful services
- Uses HTTP verbs for controller action names including:
  - GET, POST, PUT, DELETE, etc.
- Methods can return raw data which will automatically be converted into
  - either JSON or XML format depending on the HTTP request's Accept Header
- Not directly related to ASP.NET MVC
  - although some concepts are shared
- Web API favors Convention over Configuration
  - so endpoints and contracts are not used as in WCF
- Works great with JavaScript (HTML5) and other mobile apps!

Web API

- Can add support for more formats beyond JSON and XML if necessary
- Can be hosted by any .NET application without using IIS!
- Alternative approach to WCF Web Services which are more complicated
  - WCF doesn't provide native support for HTTP services
- Big part of Microsoft's Cloud Strategy going forward

Web API Controllers

- Controllers inherit from System.Web.Http.ApiController
- Method names match HTTP verbs (Get, Post, Put, Delete)
- ASP.NET Routing maps URIs and HTTP Verbs to Controller actions
  - This default routing can be overridden by annotating method names with attributes such as [HttpGet], [HttpPost], etc
- Method parameters are automatically bound to the request's parameters by the Model Binder
- Actions must be public, can't be static, no ref or out parameters
- Can return the following from an action:
  - POCO type which is automatically converted to an HttpResponseMessage
  - HttpResponseMessage by calling the Request.CreateResponse method or the Request.CreateErrorResponse method.
  - IHttpActionResult that will ultimately create an HttpResponseMessage

REST: HTTP Verbs

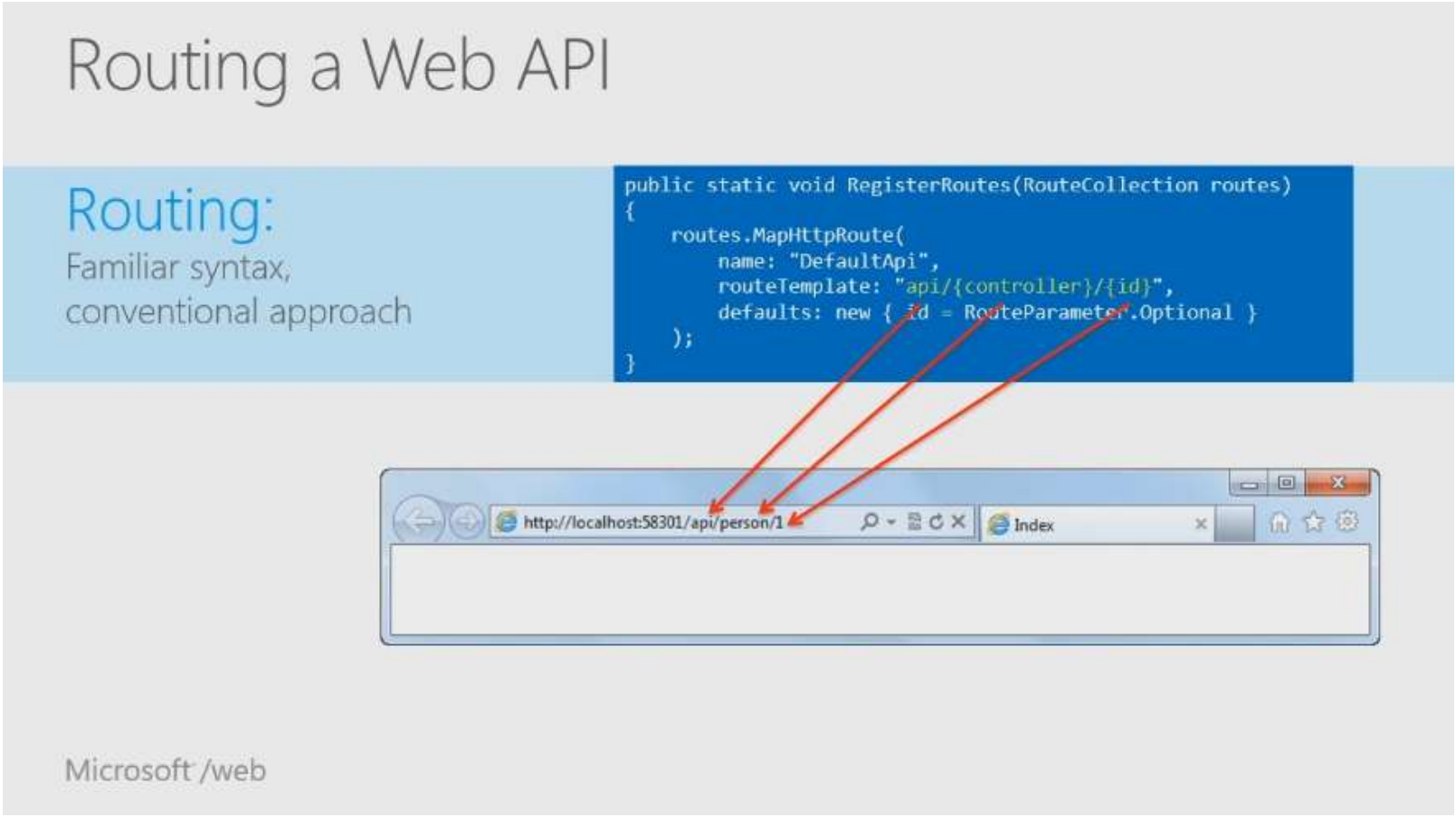
- Delete
  - Remove Data
- Post
  - Add or Create new data
- Put
  - Update or Replace existing data
- Get
  - Retrieve existing Data

Common HTTP Status Codes

- Common Groups
  - 2xx (Successful)
  - 3xx (Redirected)
  - 4xx (Request error)
  - 5xx (Server error)
- Common StatusCodes
  - 200 OK: Success
  - 201 Created - Used on POST request when creating a new resource.
  - 304 Not Modified: no new data to return.
  - 400 Bad Request: Invalid Request.
  - 401 Unauthorized: Authentication.
  - 403 Forbidden: Authorization
  - 404 Not Found – entity does not exist.
  - 406 Not Acceptable – bad params.
  - 409 Conflict - For POST / PUT requests if the resource already exists.
  - 500 Internal Server Error
  - 503 Service Unavailable
- https://support.google.com/webmasters/answer/40132?hl=en

Reuben, Josh. "ASPNET WebAPI REST Guidance." *Geekswithblogs.net*. N.p., n.d. Web. 14 Oct. 2013. <http://geekswithblogs.net/JoshReuben/archive/2012/10/28/aspnet-webapi-rest-guidance.aspx>.

Default Web API Routing



"ASP.NET Web Apps development, Create Windows Apps -Microsoft Virtual Academy." *Microsoft Virtual Academy – Free IT Training, Online Learning of Microsoft Technologies*. N.p., n.d. Web. 5 Oct. 2013. <http://www.microsoftvirtualacademy.com/training-courses/create-web-apps-with-asp-net#fbid=UzNPsZ0j8kX>.

Content Negotiation

- Web API can respond with data formatted in XML or JSON.
- Client can decide which format it will receive from a Web API service through its HTTP Request Headers
- HTTP Accept Header tells the Web API service how to format its response (XML or JSON)
- HTTP Content-Type Header tells the Web API service how to interpret the request's data (XML, JSON or HTML form)
- Different Browsers may display data from a Web API service differently (XML or JSON) depending on their preferences (HTTP Headers)

Model Binding

- Maps incoming data from the HTTP Body and/or query string (URI) to method parameters
- MediaTypeFormatters transform input and output data to/from .NET objects
- Transforms supplied data (even if it is JSON, XML or Form data) to the parameters
- You can use the [FromBody] or [FromUri] attributes for action parameters in a POST or PUT method to force Web API where to look for the object

```
// POST api/car
public void Post([FromBody]Car value)
{
    inventory.Add(value);
}

// PUT api/car/5
public void Put(int id, [FromBody]Car value)
{
    inventory[id] = value;
}
```

Default Sample Web API Code

```
10  public class ValuesController : ApiController
11  {
12      // GET api/values
13      public IEnumerable<string> Get()
14      {
15          return new string[] { "value1", "value2" };
16      }
17
18      // GET api/values/5
19      public string Get(int id)
20      {
21          return "value";
22      }
23
24      // POST api/values
25      public void Post([FromBody]string value)
26      {
27      }
28
29      // PUT api/values/5
30      public void Put(int id, [FromBody]string value)
31      {
32      }
33
34      // DELETE api/values/5
35      public void Delete(int id)
36      {
37      }
38  }
```

Video

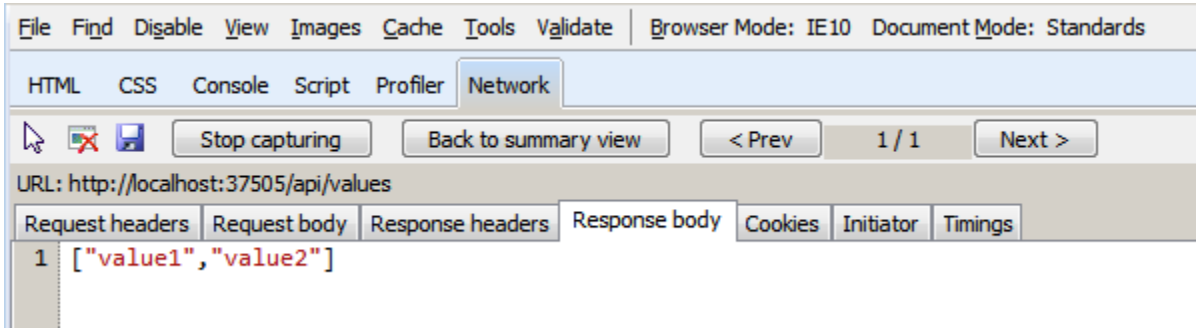
- Let's watch a great video (11.2) from the "ASP.NET MVC 4 Essential Training" series on Lynda

Video

- Let's watch a great video on Web API on Pluralsight from the free "Introduction to the ASP.NET Web API" Course

Testing Web API Services

- For testing/sniffing Web API we have a few options:
  - Advanced REST Client for Chrome
  - Fiddler (remember to set Content-Type: application/json for POSTs and PUTs)
  - WebApiTestClient NuGet package
  - FireBug for Firefox (monitoring only)
  - Developer Tools in most browsers (F12) (monitoring only)
  - Writing client test code in .NET or JavaScript



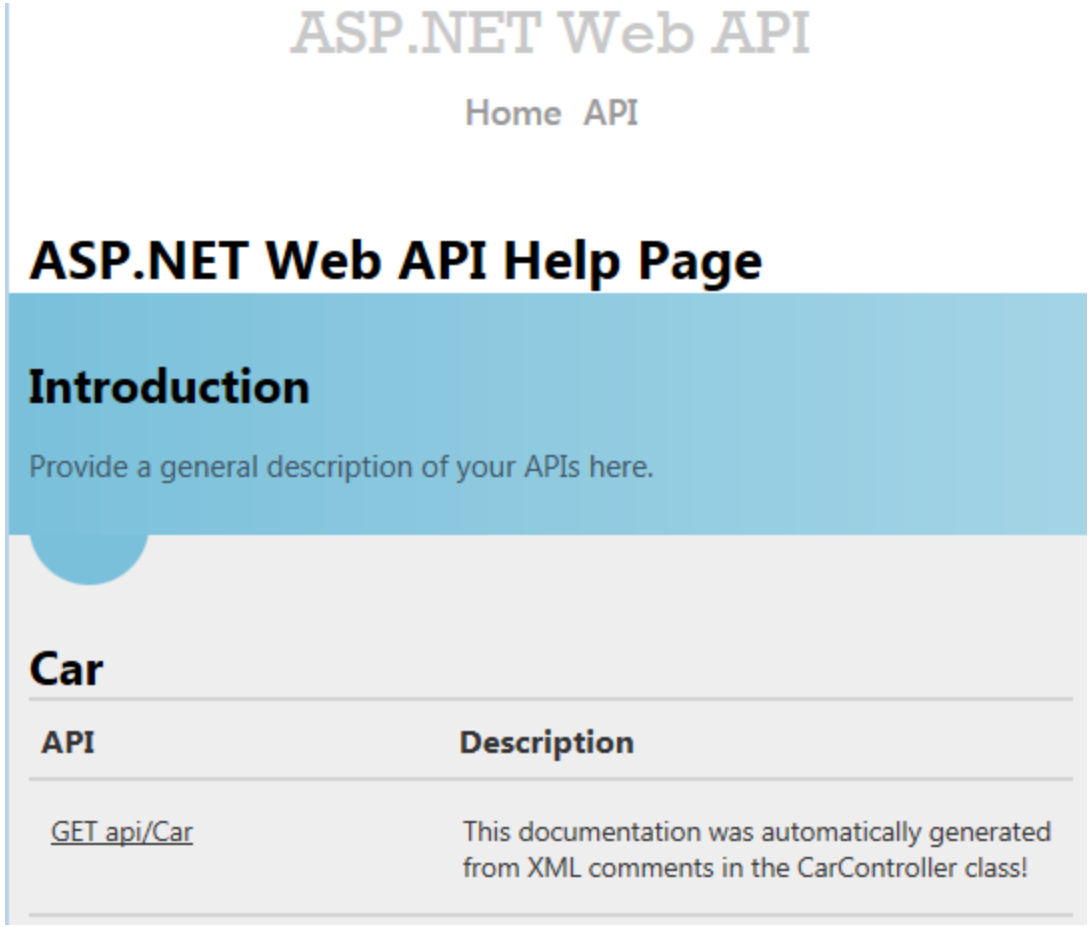
Web API Exceptions

- By default unhandled Web API Exceptions generate a HTTP response with a status code of 500 for Internal Server Error
- Throwing an HttpResponseException allows
  - Customize the response (HttpResponseMessage)
  - Provide a status code
  - throw new HttpResponseException(HttpStatusCode.NotFound);
- Alternatively, the Request.CreateErrorResponse method can be used to list errors (ex. validation) in the response
  - return Request.CreateErrorResponse(HttpStatusCode.NotFound, ModelState);
- The HttpError class can be used to provide an HTTP body which describes the error details
- Exception Filters can be used to customize Exception Handling

Web API Help Controller

- The Web API Help Controller allows you to easily and automatically add Help to your Web API Controllers!
- Available by default from the "API" navigation link in all MVC pages
- API Help is dynamically generated for Web API projects at run time using the Help View (Areas\HelpPage\Views\Help\Index.cshtml)
- To use automatic help generation:
  - Uncomment the config.SetDocumentationProvider call in the HelpPageConfig.Register method.
  - Enable "XML documentation file" in the Build properties for the project and set the name to "App_Data\XmlDocument.xml"
  - Add XML comments to your controller actions
  - Run the project and click on the API link

Web API Help Controller



Web API & Entity Framework

- Web API works well with Entity Framework
- However, two issues commonly arise:
  - Associations between entities can cause Circular References which is problematic when Web API serializes objects to a HttpResponseMessage. A dedicated ViewModel like class can be used to address this issue.
  - Entity Framework's Lazy Loading can also be problematic for Web API. This issue can often manifest itself as the first issue.

Web API & Entity Framework

- When using Web API with Entity Framework note that:
  - The virtual keyword for the navigation model properties generates a proxy class used for lazy loading in Entity Framework!
  - Web API does not work well with these proxy classes so they need to be disabled by setting the data context's Configuration.LazyLoadingEnabled and Configuration.ProxyCreationEnabled flags to false as follows:
    - public CountryApiController() : base() {
    - db.Configuration.LazyLoadingEnabled = false;
    - db.Configuration.ProxyCreationEnabled = false;
    - }
  - After disabling Lazy Loading you may need to add an "Include" call to your LINQ queries to fetch the navigation properties
    - var customers = from customer in salesContext.Customers.Include("Orders")
    - where customer.Orders.Count > 0
    - select customer;

Web API Scaffolding

- Web API 2 was recently released and it had a significant impact on the scaffolding which uses Entity Framework
- Rather than return POCOs and HttpResponseMessages many actions now return IHttpActionResult which makes it easier to test actions
- The new scaffolding makes it harder to directly call and work with Web API return values in .NET code
- Some new handy generic controller methods were also introduced in Web API 2 such as Ok<T>(T), NotFound() and BadRequest()

Web API and MVC

- The easiest way to consume a Web API service that returns a POCO is by calling it directly from the bundled MVC site
- The Web API Service and MVC site in the same project are hosted in the same ASP.NET process
- From an MVC controller, just create the ApiController and call one of its methods directly.

```
public class CountryMVCController : Controller
{
    // GET: /CountryMVC/
    0 references
    public ActionResult Index()
    {
        CountryApiController cac = new CountryApiController();

        return View(cac.Get());
    }
```

```
public class CountryApiController : ApiController
{
    private OlympicsContext db = new OlympicsContext();

    // GET api/readwriteapi
    1 reference
    public IEnumerable<Country> Get()
    {
        return db.Countries;
    }
```

Consuming Web API

- Often we want to return something other than a POCO or we need to consume the service from another project
- The HttpClient class in System.Net.Http namespace can be used by .NET applications to consume a Web API service.
- HttpClient is the newer alternative to WebClient and HttpWebRequest
- Use HttpResponseMessage with HttpClient
- Supports asynchronous programming
- Used for consuming HTTP services including Web API
- See the following sample at asp.net

Consuming Web API from another .NET Project

```
HttpClient client = new HttpClient();
client.BaseAddress = new Uri("http://localhost:60527/");
client.DefaultRequestHeaders.Accept.Add(new
        System.Net.Http.Headers.MediaTypeWithQualityHeaderValue("application/json"));

HttpResponseMessage response = client.GetAsync("api/CountryApi/" +
                                                id.ToString()).Result;
if (response.IsSuccessStatusCode)
{
    Country country = response.Content.ReadAsAsync<Country>().Result;
    return View(country);
}
else
    return View("Error", "The specified country was not found" as object);
```

Simple blocking version!

Hosting

- Web API Services can be hosted:
  - From ASP.NET application running on IIS
  - Self hosted within any .NET application using Self Hosting API (create HttpSelfHostConfiguration and HttpSelfHostServer)
- HttpConfiguration can be accessed via GlobalConfiguration.Configuration property
- Can use the same ASP.NET routing and configuration settings for self hosting
- Web API Projects automatically come with ASP.NET MVC support which makes hosting easy!

Invoking Web API from JavaScript

- Web API services can be easily consumed from JavaScript and other languages since they can emit JSON and XML.

Invoking Web API from JavaScript

```
<h2>Consuming a Web API Service using JavaScript</h2>
<div id ="cars">
    <button onclick ="GetCars()">Get Cars</button>
    <ul id="carList" />
</div>

<script type="text/javascript">
function GetCars() {
       $.getJSON("http://localhost:17667/api/Car",
            function (data) {
                $.each(data, function(key, val) {
                    var str = val.Make + ' ' + val.Model;
                    $('<li/>', {text:
str}).appendTo($('#carList'));
                });
        });
    };
</script>
```

Web Services and Capstone

- Web API Web Services are ideal for your Capstone project for the following reasons:
  - Cross platform
  - Leaves the heavy processing on the server where it belongs
  - Support JSON and work great with JavaScript and other languages used on mobile devices
  - They are a growing field worth exploring
- ASP.NET works well with mobile browsers but Web API will provide more flexibility for mobile apps
- Note: You will be required to build a Web API Service for your Capstone projects in this course!

OData

- Open Data Protocol (OData) is a RESTful data access protocol which leverages AtomPub and JSON
- Provides a data model, query language and client libraries for various clients
- Allows queries to be placed in URIs
- Used by Microsoft Azure for data access
- Web API supports it as long as a method returns IQueryable<T> and has the [IQueryable] attribute applied