

.NET Core: Developing Cross-Platform Web Apps with ASP.NET Core – Workshop*PLUS*

Wael Kdoh - @waelkdoh

Senior Consultant

v3.0

Conditions and Terms of Use

Microsoft Confidential

This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet Web site references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

Copyright and Trademarks

© 2016 Microsoft Corporation. All rights reserved.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see Use of Microsoft Copyrighted Content at
<http://www.microsoft.com/en-us/legal/intellectualproperty/Permissions/default.aspx>

Azure, Internet Explorer, Microsoft, SQL Server, Visual Studio, and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners.

Module 5: Web API

Module Overview

Module 5: Web API

Section 1: RESTful Services

Lesson: REST Fundamentals

What is Representational State Transfer?

- REST defines a set of *design principles and architectural styles* for highly scalable distributed systems
- REST is coined by Roy Fielding in his PhD thesis in 2000
- Roy is also the principle author of the HTTP specification and the co-author of the Uniform Resource Identifier (URI) specification
- The World Wide Web is an existing proof of a massively scalable distributed system that follows REST principles

REST Constraints - I

- Client-Server
 - Clients not concerned with server function (processing of data, storage mechanism)
 - Server not concerned with UI representations of the resource (rendering)
- Stateless
 - Client context is not stored on the server
- Layered
 - Intermediary servers or devices (for example, load balancers) are transparent to the client.

REST Constraints - II

- Cacheable
- Code on Demand (optional)
 - JavaScript executed on the client.
- Uniform Interface
 - URI (addressing the resource)
 - Verbs (manipulation of the resource)
 - Self-Describing (content-type headers)
 - Hyperlinks / HATEOAS

Module 5: Web API

Section 1: RESTful Services

Lesson: REST Principles

REST Principles

- Identifiable resources
 - Give everything an ID
- Resource representations
 - Resources with multiple representations
- Hypermedia as the Engine of Application State (HATEOAS)
 - Link resources together
- Uniform Interface
 - Use standard methods
- Stateless communication
 - Communicate in a stateless fashion

Resource Representations

- Resource can have multiple representations, for example, HTML, XHTML, XML, JSON, plain text, multimedia, Atom, etc.
- Content negotiation allows client to specify the type of content it can accept and preference: "I want XML"
- MIME defines some standard content types:
 - "text/plain"
 - "text/html"
 - "image/jpeg"
 - "audio/mp3" and so on.

Uniform Interface

- Every resource supports the same interface
- In HTTP, this interface is
 - HTTP verbs: GET, PUT, POST, DELETE, HEAD, OPTIONS
 - HTTP status codes
- Safe operations: Idempotence
- Advantages
 - Simplicity
 - Stateless with respect to client and server communication
 - Extensibility
 - Mashups, MIME types, etc.
 - Visibility
 - URIs, Verbs, headers provide for services like caching

Why REST?

- Move towards a resource-based application model
- Interoperability between cloud, on-premises and cross-domains
- Take advantage of Web's infrastructure
- Easy composition of resources via Uniform Interface
- A “must have” if your APIs will be consumed by third-party services

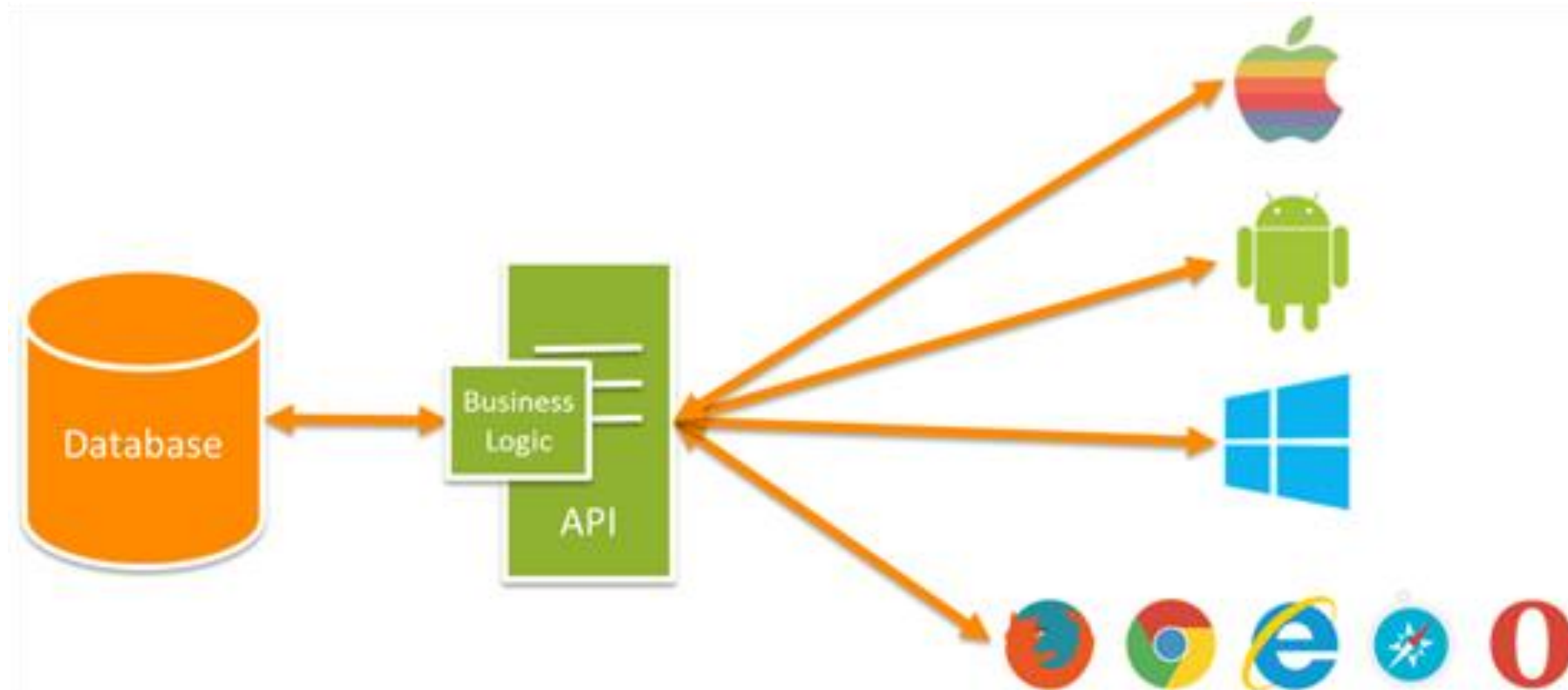
Demo: Online REST Web Service

Module 5: Web API

Section 2: ASP.NET Web API

Lesson: Web API Fundamentals

Web API



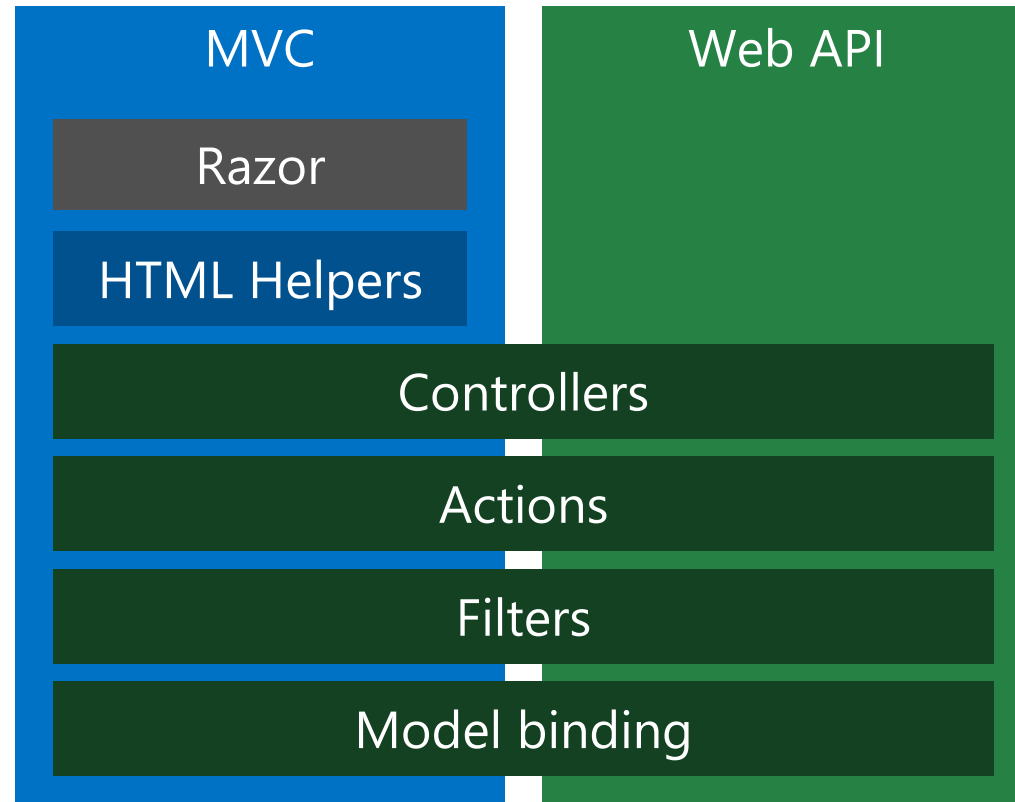
The History

- Windows Communication Foundation (WCF) Support for REST
 - WebHTTPBinding (WCF 3.5)
 - WCF REST Starter Kit (WCF 4)
 - WCF Web API (CodePlex)
- ASP.NET MVC 3
 - Returning JavaScript Object Notation (JSON) from controllers
- ASP.NET Web API takes the best features from WCF Web API and merges them with the best features from MVC

ASP.NET Web API

- ASP.NET Web API is a framework that makes it easy to build HTTP services.
 - Full support for ASP.NET Routing
 - Content negotiation and custom formatters
 - Model binding and validation
 - Filters
 - Query composition
 - Easy to unit test
 - Inversion of Control (IoC)
 - Code-based configuration
 - Self hosting

The World Today with ASP.NET Core



Demo: Adding Web API Controller

Module 5: Web API

Section 2: ASP.NET Web API

Lesson: Routing

Routing - Controllers

- Only one base class for ASP.NET MVC and ASP.NET Web API controllers
 - Both MVC and Web API controllers derive from the very same ***Microsoft.AspNetCore.Mvc.Controller*** base class
- ```
[Route("api/[controller]")]
public class ValuesController : Controller
```
- You can use RESTful style routing with both MVC and Web API controllers
- Same routing framework for both MVC and the Web API
  - Convention-based routes or attribute routes
- Default route template for Web API is "**api/[controller]**"

# ActionResult for UI and API

- Controllers return IActionResult
  - For MVC Controllers, it might be view or data
  - For Web API controllers, it might be data

| UI                | API                  |
|-------------------|----------------------|
| PartialViewResult | BadRequestResult     |
| RedirectResult    | ContentResult        |
| ViewResult        | CreatedAtRouteResult |
| JsonResult        | HttpStatusCodeResult |
|                   | JsonResult           |
|                   | ObjectResult         |
|                   | ChallengeResult      |
|                   | HttpNotFoundResult   |
|                   | FileContentResult    |



# IActionResult for UI and API

- Controllers return IActionResult
  - For MVC Controllers, it might be view or data
  - For Web API controllers, it might be data

```
[HttpGet("{id:int}")]
public IActionResult GetById(int id)
{
 var item = _items.FirstOrDefault(x => x.Id == id);
 if (item == null)
 {
 return NotFound();
 }

 return new ObjectResult(item);
}
```

# Routing - Controllers

- To find the controller, Web API adds "Controller" to the value of the *[controller]* variable.
- Once a matching route is found, Web API selects the controller
- Web API uses the HTTP method, not the URI path, to select the action
- If no route matches, the client receives a 404 error

# Routing - Actions

- Web API looks at the HTTP method, and then looks for an action whose **HTTP attribute** is the same as the method
- This convention applies to GET, POST, PUT, and DELETE methods
- Other placeholder variables in the route template, such as {id} and query strings, are mapped to action parameters.

```
[HttpGet("{id}")]
public string Select(int id)
```

- **[FromBody]** attribute tells the framework to deserialize the parameter from the request body

```
[HttpPut("{id:int}")]
public void Update(int id, [FromBody]string value)
```

# HTTP Verbs

|        | <b>Collection URI</b><br>( <a href="http://api.example.com/v1/resources/">http://api.example.com/v1/resources/</a> ) | <b>Element URI</b><br>( <a href="http://api.example.com/v1/resources/item17">http://api.example.com/v1/resources/item17</a> ) |
|--------|----------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| GET    | List the URIs and perhaps other details of the collection's members.                                                 | Retrieve a representation of the addressed member of the collection.                                                          |
| PUT    | Replace the entire collection with another collection.                                                               | Replace the addressed member of the collection, or if it does not exist, create it.                                           |
| POST   | Create a new entry in the collection.                                                                                | Treat the addressed member as a collection in its own right and create a new entry in it.                                     |
| DELETE | Delete the entire collection.                                                                                        | Delete the addressed member of the collection.                                                                                |

# Routing - Actions

| Verb   | URL          | Action                                                                  |
|--------|--------------|-------------------------------------------------------------------------|
| GET    | api/values   | [HttpGet]<br>public IEnumerable<string> Get()                           |
| GET    | api/values/5 | [HttpGet("{id}")]<br>public string Select(int id)                       |
| PUT    | api/values/5 | [HttpPut("{id}")]<br>public void Update(int id, [FromBody]string value) |
| POST   | api/values   | [HttpPost]<br>public void Create([FromBody]string value)                |
| DELETE | api/values/5 | [HttpDelete("{id}")]<br>public void Delete(int id)                      |

- To prevent a method from getting invoked as an action, use the **NonAction** attribute
- In case multiple Actions match, you get runtime exception:
  - 500 Internal Server Error - Microsoft.AspNetCore.Mvc.AmbiguousActionException

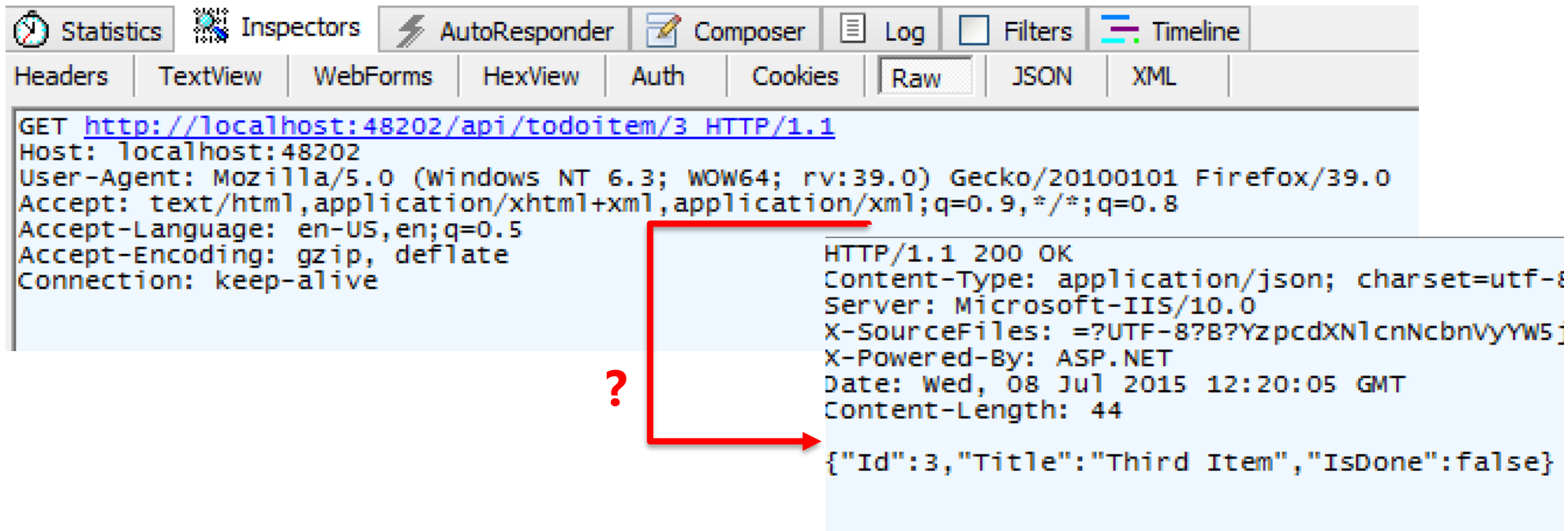
Module 5: Web API

Section 2: ASP.NET Web API

Lesson: Content Negotiation

# Content Negotiation

- Apps consume content in different ways
- JSON and Extensible Markup Language (XML) are the most popular formats
- A good client will request the favored type



```
GET http://localhost:48202/api/todoitem/3 HTTP/1.1
Host: localhost:48202
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:39.0) Gecko/20100101 Firefox/39.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive

HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Server: Microsoft-IIS/10.0
X-SourceFiles: =?UTF-8?B?YzpcdXNlc nNcbnvYyW5j
X-Powered-By: ASP.NET
Date: Wed, 08 Jul 2015 12:20:05 GMT
Content-Length: 44

{"Id":3,"Title":"Third Item","IsDone":false}
```



# JSON Only By Default

- XML formatters (input and output) are now removed by default

| Input formatters        | Output formatters            |
|-------------------------|------------------------------|
| JsonInputFormatter      | JsonOutputFormatter          |
| JsonPatchInputFormatter | StringOutputFormatter        |
|                         | StreamOutputFormatter        |
|                         | HttpNoContentOutputFormatter |

- Custom formatters can be developed

# If You Need XML...

- XML formatters are found in "[Microsoft.AspNetCore.Mvc.Formatters.Xml](#)" package
- Manually add it to the formatters collection in Startup.cs

```
services.Configure<MvcOptions>(options =>
{
 options.OutputFormatters.Add(new XmlSerializerOutputFormatter());
 //options.InputFormatters.Add(new XmlSerializerInputFormatter());
});
```
- Use the [Produces] attribute in the action method

```
[Produces("application/xml")]
public IActionResult GetAll(int id)
```

# Demo: Content Negotiation

Module 5: Web API

Section 2: ASP.NET Web API

Lesson: Swagger Documentation

# Swagger Documentation

- Swagger is a simple yet powerful representation of your RESTful API.
- Swagger is the **most popular** representation of RESTful API.
- Swagger generating good documentation and help pages as a part of your Web API
- Documentation is customizable:
  - XML comment from you code will be included
  - Data Annotation Attributes will be applied

# Swagger configuration

```
18
19 + // This method gets called by a runtime. ...
 - references | Igor Sychev, 120 days ago | 1 author, 2 changes | 1 work item
21 - public void ConfigureServices(IServiceCollection services)
22 {
23 services.AddSwaggerGen();
24 #region
41 }
42
43 // Configure is called after ConfigureServices is called.
 - references | Igor Sychev, 120 days ago | 1 author, 3 changes | 2 work items
44 public void Configure(IApplicationBuilder app,
45 IHostingEnvironment env, ILoggerFactory loggerFactory)
46 {
47 #region
56 app.UseSwaggerUi();
57 app.UseSwagger();
58 }
--
```

# Swagger UI example:

swagger   Explore

## API V1

### Todo

Show/Hide List Operations Expand Operations

GET /api/Todo

Response Class (Status 200)

OK

Model | Model Schema

```
[
 {
 "key": "string",
 "name": "string",
 "isComplete": true
 }
]
```

Response Content Type

Try it out!

POST /api/Todo

DELETE /api/Todo/{id}

GET /api/Todo/{id}

PUT /api/Todo/{id}

[ BASE URL: / , API VERSION: V1 ]



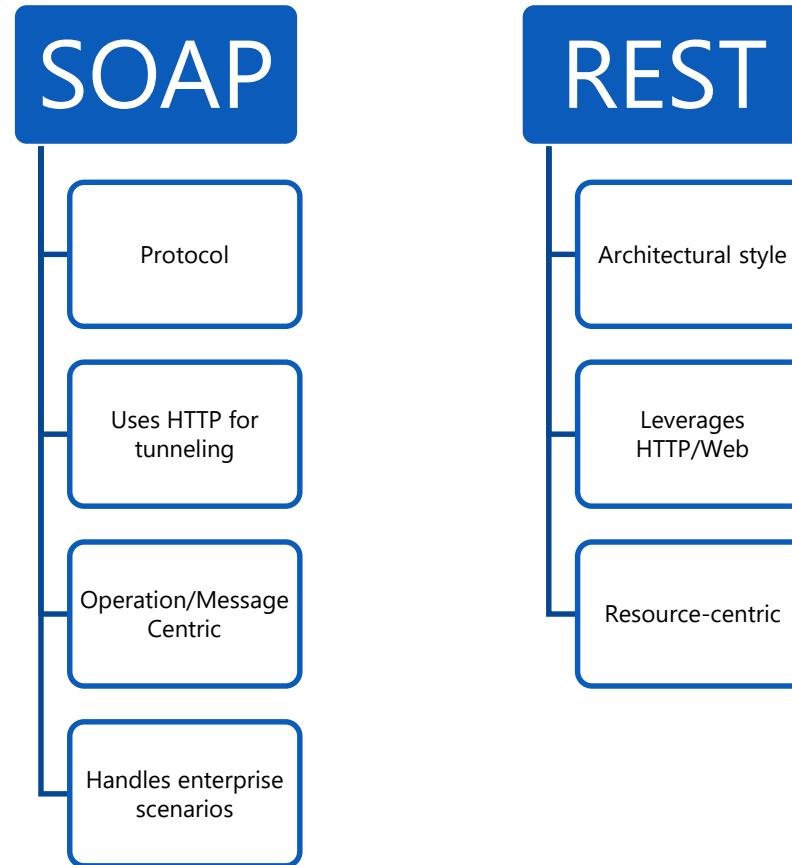
# Demo: Swagger

## Module 5: Web API

### Section 3: Web API Usage Patterns

### Lesson: Which Technology to Use?

# SOAP and REST: Brief Comparison and Contrast



# Which Technology To Use?

| WCF                                                                  | ASP.NET Web API                                      |
|----------------------------------------------------------------------|------------------------------------------------------|
| Multiple transport protocols (HTTP, TCP, UDP, and custom transports) | HTTP only. First-class programming model for HTTP    |
| Web Services Description Language (WSDL) configuration file          | Simple and lightweight                               |
| Multiple encodings (Text, MTOM, and Binary)                          | Wide variety of media types including XML, JSON etc. |

# Which Technology To Use? (continued)

| WCF                                                                    | ASP.NET Web API                                                                                        |
|------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|
| Services with WS-* standards                                           | Basic protocol and formats such as HTTP, WebSocket, and SSL                                            |
| Supports Request-Reply, One Way, and Duplex message exchange patterns. | HTTP is request/response                                                                               |
| WCF SOAP services can be described                                     | Variety of ways to describe a Web API                                                                  |
| Ships with the .NET framework                                          | Ships with .NET framework but is open-source and is also available out-of-band as independent download |

Module 5: Web API

Section 3: Web API Usage  
Patterns

Lesson: Web API Design

# Good API Design

- There is no real standard or method for designing APIs
- Keep your base URL simple and intuitive
- Do not change your URLs over time

# Nouns, No Verbs

- Use nouns for resource names
- Use plural rather than singular nouns, and concrete rather than abstract names
- Keep verbs out of your base URLs
- Use HTTP verbs to operate on the collections and elements

*Tip: Nouns are good, verbs are bad*

- Actions that do not fit into the world of CRUD operations can be endpoint
  - GitHub's API
    - star a gist with PUT /gists/:id/star
    - unstar with DELETE /gists/:id/star



# Simplify Associations

- Nested resources(Association)
- User URI navigation (/resource/identifier/resource)
  - GET /api/Customers/123/Invoices
  - GET /api/Customers/123/Payments
  - GET /api/Customers/123/Shipments
- Keep the base resource URLs lean by implementing query parameters
  - GET /api/Customers?state=GA
  - GET /api/Customers?hasOpenOrders=true

# Use Meaningful HTTP Status Codes

- Use HTTP status codes and map them cleanly to relevant standard-based codes

| Group | Comment                      |
|-------|------------------------------|
| 1xx   | Informational (100, 101)     |
| 2xx   | Success (200, 201)           |
| 3xx   | Redirect (301, 304)          |
| 4xx   | Client Error (401, 404, 405) |
| 5xx   | Server Error (500, 503)      |

# Versioning

- Never release an API without a version and make the version mandatory
- Maintain at least one version back
- Versioning with Content Negotiation and custom headers are popular now
  - Content-Type: application/json;v=2
  - X-MyApp-Version = 2
- Versioning with URI components are more common
  - Easier to implement
  - GET /api/v2/Customers/123/Invoices

# Caching

- Implement caching carefully
- Time-based (Last-Modified)
  - When generating a request, include a HTTP header Last-Modified
  - If an inbound HTTP requests contains a If-Modified-Since header, the API should return a 304 Not Modified status code instead of the output representation of the resource
- Content-based (ETag)
  - Useful when the last modified date is difficult to determine

# Demo: ETag

# Security

- Do you need to secure your API?

| Are you?                                   | Secure? |
|--------------------------------------------|---------|
| Using private or personalized data         | Yes     |
| Sending sensitive data across the wire?    | Yes     |
| Using credentials of any kind              | Yes     |
| Trying to protect overuse of your servers? | Yes     |

# Security

- Do not expose your domain model
- Always use SSL
- Secure the API itself
  - HTTP Basic Authentication
  - API token keys
  - Custom authentication mechanism with cookies or token
  - OAuth

**PayPal**  
Permissions Service API

**Facebook**  
OAuth 2.0

**Twitter**  
OAuth 1.0a

# Demo: Web API Security



# Module Summary

- In this module, you learned about:
  - REST principles
  - Web API fundamentals
  - Web API routing
  - Content negotiation
  - Web API Design





# Lab: ASP.NET Web API



