Building RESTful services with WebAPI

Mark Smith
mark@julmar.com

julmar.com
SOFTWARE DEVELOPMENT, ARCHITECTURE & TRAINING

Who Am I?

- My name is Mark Smith
  - mark@julmar.com
  - @marksm
  - julmar.com/blogs/mark
  - linkedin.com/in/marksm
- Consultant through my company julmar.com focused primarily on UI design, parallel programming and debugging
- DEVELOPMENTOR author and instructor
- Wintellect consultant, author and instructor

Microsoft Most Valuable Professional

What are REST services?

- REST (Representational State Transfer) is an architectural style to create distributed applications modeled around the HTTP specification


UNIVERSITY OF CALIFORNIA, IRVINE

Architectural Styles and the Design of Network-based Software Architectures

DISSERTATION

submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in Information and Computer Science

by

Roy Thomas Fielding

2000

Why use REST?

- REST is designed to capitalize on the architecture and operation of the World Wide Web
  - CRUD operations are implemented as HTTP verbs
  - URLs represent accessible resources
  - read operations are cachable

REST vs. SOAP [comparison]

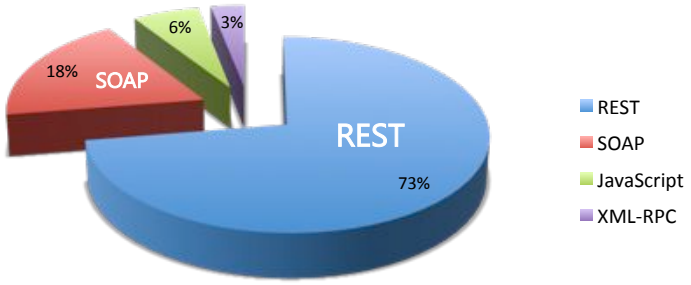
REST

SOAP

	REST	SOAP
Standardized	No	Yes (WS-*)
Transport	HTTP	HTTP, FTP, Pipes, MSMQ etc
Data Format	Any	XML
Reach	Excellent	Good
Development tools	Basic	Good
Flexibility	Flexible	Rigid
Scalability	Excellent	Reasonable
Security	HTTPS	WS-Security
Operations	Stateless	Often statefull

REST vs. SOAP [popularity]

- REST tends to be easier to work with for Javascript clients – as such it has become the most popular access strategy



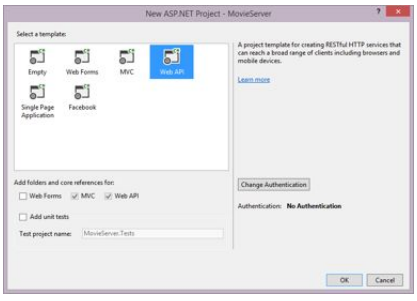
Internet APIs, Data: Programmable Web, Feb 10, 2012
REST (71%), SOAP (18%), JavaScript (6%), and XML-RPC (3%)

The Web API framework

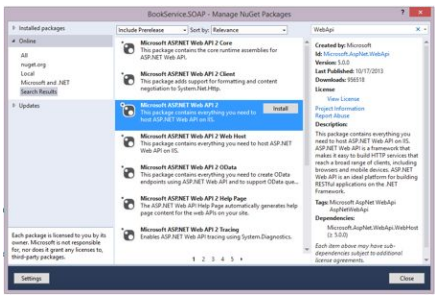
- WebAPI was introduced with ASP.NET MVC and was designed to simplify the creation of REST-based services
  - Routing is used for URIs
  - Controllers model resources
  - HTTP methods map to methods on Controller
  - `HttpRequestMessage` provides access to request
  - `HttpResponseMessage` provides API for response
  - Content negotiation allows for client requested media types
  - `MediaTypeFormatters` serialize/deserialize formats
- Published under the MS-PL open source license
  - `https://aspnet.codeplex.com/`

Getting WebAPI

- Two primary ways to add WebAPI support to your projects



Can create a new web site with WebAPI



.. or add it to existing project with Nuget

Mapping URLs to code

- URL mappings are registered with system – will then locate based on template matching just like ASP.NET MVC

{placeholders} are used to locate specific elements – here the **{controller}** will map to a specific **`ApiController`** class in the **Controllers** folder

```
public static void RegisterRoutes(RouteCollection routes)
{
   ...
   routes.MapHttpRoute(
        name: "DefaultApi",
        routeTemplate: "api/{controller}/{id}",
        defaults: new { id = RouteParameter.Optional }
    );
}
```

can make portions of the route optional – so they are not required to locate a matching method (action) in the controller

Creating WebAPI services

- WebAPI controllers represent resources and HTTP requests are mapped onto methods of controller – model binding is used to map body and query string to parameters

```
public class MovieController : ApiController
{
    public IEnumerable<Movie> Get() { ... }

    public Movie Get(int id) { ... }

    public Movie Post(Movie movie) { ... }

    public Movie Put(Movie movie) { .... }

    public void Delete(int id) { .... }
}
```

WebAPI route mapping

- Can also use attribute model to setup routes (WebAPI2)

```
public class BooksController : ApiController
{
    [Route("api/movies/")]
    [HttpGet]
    public IEnumerable<Movie> Get() { ... }

    [Route("api/movies/filterby/{filter}"]
    [HttpGet]
    public IEnumerable<Movie> GetWithFilter(string filter) { ... }

    [Route("api/movies/{id}")]
    [HttpPost, HttpPut]
    public Movie Update(Movie movie) { ... }
}
```

Request/Response management

- Can access raw message body and response through specific request / response types

```
public class MoviesController : ApiController
{
   public HttpResponseMessage Update(HttpRequestMessage request)
   {
      try
      {
         var jo = input.Content.ReadAsAsync<JObject>().Result;
         UpdateMovieFromJSON(jo);
      }
      catch (Exception ex)
      {
         return Request.CreateErrorResponse(
                    HttpStatusCode.BadRequest, ex.Message);
      }
      ...
   }
}
```

Returning specific status codes

- Can return statuses either by throwing exception or by returning HttpResponseMessage

```
public Movie Get(int id)
{
   var movie = _netflixDbService.FindById(id);
   if (movie == null)
      throw new HttpResponseException(HttpStatusCode.NotFound);
   ...
}

public HttpResponseMessage Get (int id)
{
   var movie = _netflixDbService.FindById(id);
   if (book == null)
      return new HttpResponseMessage(HttpStatusCode.NotFound);
   ...
}
```

WebAPI2 – returning status codes

- WebAPI2 adds a new IHttpActionResult return type and base return methods for capturing status and typed result

```
public IHttpActionResult Get(int id)
{
   var movie = _netflixDbService.FindById(id);
   if (movie == null)
      return NotFound();
   return Ok(movie);
}
```

Resource formats

- Clients can request a specific format through the **Accept** HTTP header, WebAPI will attempt to honor the client request

```
GET http://localhost:2100/api/movies/5 HTTP/1.1
Accept: application/json

HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Content-Length: 556

{"ID":503321,"Title":"Star Wars: A New Hope", "Director" : "..."}
```

Content negotiation

- MediaTypeFormatters serialize and de-serialize formats
  - WebAPI supports XML and JSON formats directly
  - configure existing formatters or create custom formats by deriving from MediaTypeFormatter and adding to the WebAPI configuration

```
static void Register(HttpConfiguration config)
{
   // Turn on camel casing and enum strings for JSON output
   var jsonFormatter = GlobalConfiguration.Configuration
                          .Formatters.JsonFormatter;
   jsonFormatter.SerializerSettings.Converters.Add(
         new StringEnumConverter());
   jsonFormatter.SerializerSettings.ContractResolver =
         new CamelCasePropertyNamesContractResolver();
   ...
}
```

Summary

- REST services embrace HTTP
  - resources model the data we work with and are exposed via URIs
  - HTTP methods map to the CRUD actions we are used to
  - HTTP status codes used to indicate success or failure
- REST is more flexible than SOAP and more accessible to a larger number of clients, but SOAP is still useful in some scenarios