Module Three

WCF (SOAP), REST, OData, and GraphQL

WCF and SOAP

The “Simple” Object Access Protocol

- Windows Communication Foundation (WCF) “Indigo” circa 2005
- Handle variety of protocols, transports, etc.
- Key web-based scenario was SOAP-based
- SOAP established in 1998
- XML-based

PASS

WCF: Discovery – the “WSDL”

```
<definitions name="HelloService"

targetNamespace="http://www.examples.com/wsdl/HelloService.w
sdl"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:hs="http://www.examples.com/wsdl/HelloService.wsdl"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"></definitions>
```

WCF: Discovery

```
<message name="SayHelloRequest">
       <part name="firstName" type="xsd:string"/>
</message>

<message name="SayHelloResponse">
       <part name="greeting" type="xsd:string"/>
</message>
```

WCF: Discovery

```
<portType name="Hello_PortType">
      <operation name="sayHello">
            <input message="hs:SayHelloRequest"/>
            <output message="hs:SayHelloResponse"/>
      </operation>
</portType>
```

WCF: Discovery

```
<binding name="Hello_Binding" type="hs:Hello_PortType">
   <soap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="sayHello">
      <soap:operation soapAction="sayHello"/>
      <input>
         <soap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="urn:examples:helloservice"
            use="encoded"/>
      </input>

      <output>
         <soap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="urn:examples:helloservice"
            use="encoded"/>
      </output>
   </operation>
</binding>
```

WCF: Discovery

```
<service name="Hello_Service">
   <documentation>WSDL File for HelloService
   </documentation>
   <port binding="hs:Hello_Binding" name="Hello_Port">
      <soap:address
         location="http://www.examples.com/SayHello/" />
   </port>
</service>
```

WCF: Security with WS-Security

Making SOAP less Simple

- “Envelope” with header and message body
- Security:
- Security Token
- Assertions
- Signatures
- Keys
- Encrypted content

SOAP Stopped Being Simple.

JSON is overtaking XML.

- Mobile phenomenon – SOAP was complex and difficult to implement on mobile devices
- For JavaScript clients, parsing and building XML was difficult
- Instead, they turned to existing protocols (HTTP) and transporting simpler structures (text, JSON, CSV)
- HTTP has built-in support for "verbs" GET, POST, PUT, and DELETE
- There has been a slow and steady move from SOAP to REST

PASS

REST

The “Built-in” Web API

A Brief History

Two Decades of REST

- Proposed in 2000 (Roy Fielding)
- Not tied to HTTP but this is by far the most common implementation
- Objects and Services are *Resources*
- Methods identify actions (get, post, delete, put)
- Stateless
- Collections and relationships

The REST Matrix

URI	POST	GET	PUT	DELETE
/orders	New order	Retrieve list of orders	Bulk update	Delete everything
/orders/O1	N/A	Properties of order "O1"	Update "O1" if exists	Delete order "O1"
/orders/O1/items	New line item on order "O1"	Retrieve list of line items for "O1"	Bulk update line items for Order "O1"	Delete all line items for order "O1"

Request and Response

Freedom of Content

- The request can provide a list of accepted formats, for example:
  - application/json
  - application/xml
- The response should honor the request
  - Custom types are allowed
  - Example: comma or tab separated values

PASS

Demo

REST in Action

Stay tuned…

There’s much more to REST.

- HATEOAS
- Idempotency
- Security
- Versioning
- … we’ll cover these in later modules

OData

REST for Data Sets

Introducing OData

"REST done Right"

- Everything is a resource
- Unique identifiers with ('id') format
- Queries with $search, $filter, $count, $orderby, $skip, $top
- Creation with POST
- Update with PUT or PATCH
- Relationships
- Method invocations

PASS

Demo

OData in Action

OData on the Server (Entity Framework)

```
ODataModelBuilder builder = new
ODataConventionModelBuilder();
builder.EntitySet<Product>("Products");
config.MapODataServiceRoute(
        routeName: "ODataRoute",
        routePrefix: null,
        model: builder.GetEdmModel());
```

OData on the Server (Entity Framework)

```
public class ProductsController : ODataController
{
   ProductsContext db = new ProductsContext();

   [EnableQuery]
   public IQueryable<Product> Get()
   {
      return db.Products;
   }

   [EnableQuery]
   public SingleResult<Product> Get([FromODataUri] int key)
   {
      IQueryable<Product> result = db.Products.Where(p => p.Id == key);
      return SingleResult.Create(result);
   }
}
```

OData on the Server (Entity Framework)

```
public async Task<IHttpActionResult> Post(Product product)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }
    db.Products.Add(product);
    await db.SaveChangesAsync();
    return Created(product);
}
```

OData Tools

The "Easy Button" for REST

- .NET Support: nuget Microsoft.AspNet.OData, Microsoft.AspNetCore.OData
- Visual Studio Code: https://marketplace.visualstudio.com/items?itemName=stansw.vscode-odata
- Validation: http://services.odata.org/validation/

GraphQL

Powered by JSON

Graph Query Language

"Ask for what you need, get exactly that."

- Client makes requests
- Requests include attributes desired
- Supports aliases for queries that request multiple results
- Fragments to reuse requests for specific fields
- Variable definitions and more

PASS

GraphQL Examples

```
{
    me {
        name
    }
}
```

```
{
    "data": {
        "me": {
            "name": "Jeremy"
        }
    }
}
```

GraphQL Examples

```
{
   me {
      name
      email
   }
}
```

```
{
   "data": {
      "me": {
         "name": "Jeremy"
         "email": 
"Jeremy.Likness@Microsoft.com"
      }
   }
}
```

GraphQL Examples

```
{
    me {
        associates {
            name
        }
    }
}
```

```
{
    "data": {
        "me": {
            "associates": [
                { "name": "Scott Cate" },
                { "name": "John Papa" }
            ]
        }
    }
}
```

GraphQL Examples

```
{
   people(id: "001") {
      name
   }
}
```

```
{
   "data": {
      "people": {
         "name": "Scott Cate" }
      }
   }
}
```

GraphQL Examples

```
{
   boss: people(id: 001) {
      ...personFields
   }
   me: people(id: 002) {
      ...personFields
   }
}
fragment personFields on Person
{
   name
   twitterFollowers
}
```

```
{
   "data": {
      "boss": {
           "name": "Scott Cate",
          "twitterFollowers": alot
      },
      "me": {
           "name": "Jeremy Likness",
          "twitterFollowers": afew
      }
   }
}
```

Demo

GraphQL in Action: GitHub

Challenges

"With great power comes great responsibility."

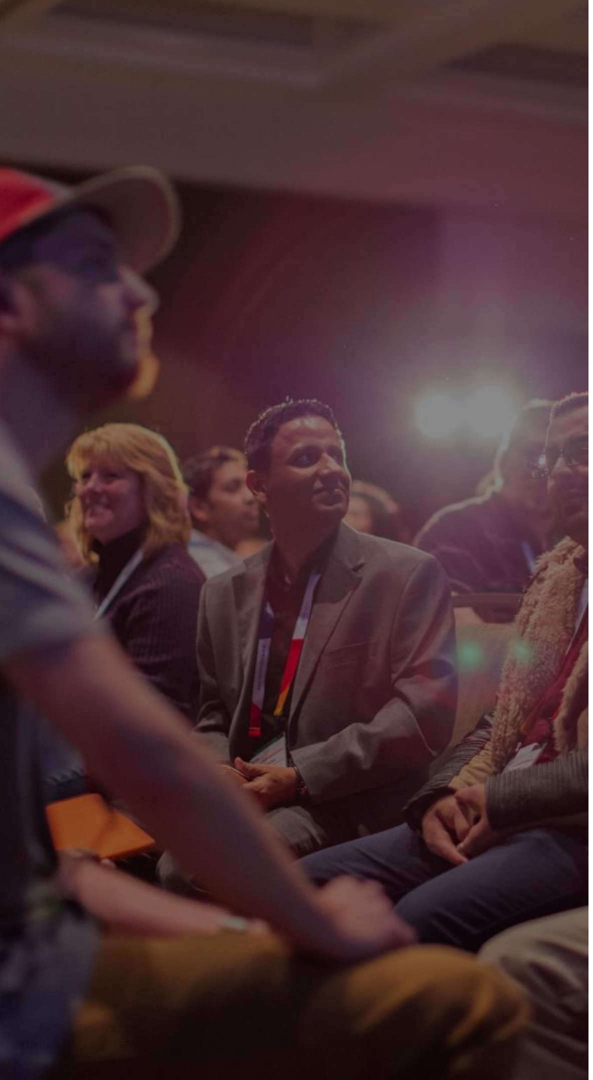
- Server must be able to parse query and return results
- Not all queries will be optimal
- No protection against asking for extremely large datasets
- Translation between query and backend database can be challenging depending on the type of data
- Works best with document database, but relational can be viewed as documents

PASS

Tools

GraphQL for the World!

- GraphQL to SQL queries: https://github.com/stems/join-monster
- GraphQL to IQueryable: https://github.com/ckimes89/graphql-net
- Implementing in ASP.NET Core: http://asp.net-hacker.rocks/2017/05/29/graphql-and-aspnetcore.html



Thank You

Learn more from Jeremy Likness

@JeremyLikness

Jeremy.Likness@microsoft.com

PASS