Web Services

david.lynch@raglansoftware.com

Web Services

Definition Communication	A software system designed to interoperable machine to machine over a network. Effectively, web services enable **RPC over the Web** using well defined standards.
Interface	A Web Service typically has a definition that is described in a machine-processable definition language.
Interaction	Is facilitated by messages that are defined in well-formed messages that are serialized and transmitted over the HTTP protocol.
"Big"	XML Messages that follow the SOAP protocol to invoke Web Services that are defined using the Web Services Description Language WSDL.
RESTFul protocol.	Representational State Transfer. More closely affiliated with the HTTP Facilitates stateless interaction and uses more lightweight ETF defined standards such as JSON.

Why Web Services?

Well defined protocols such as HTTP

Very loose coupling of consumers and producers

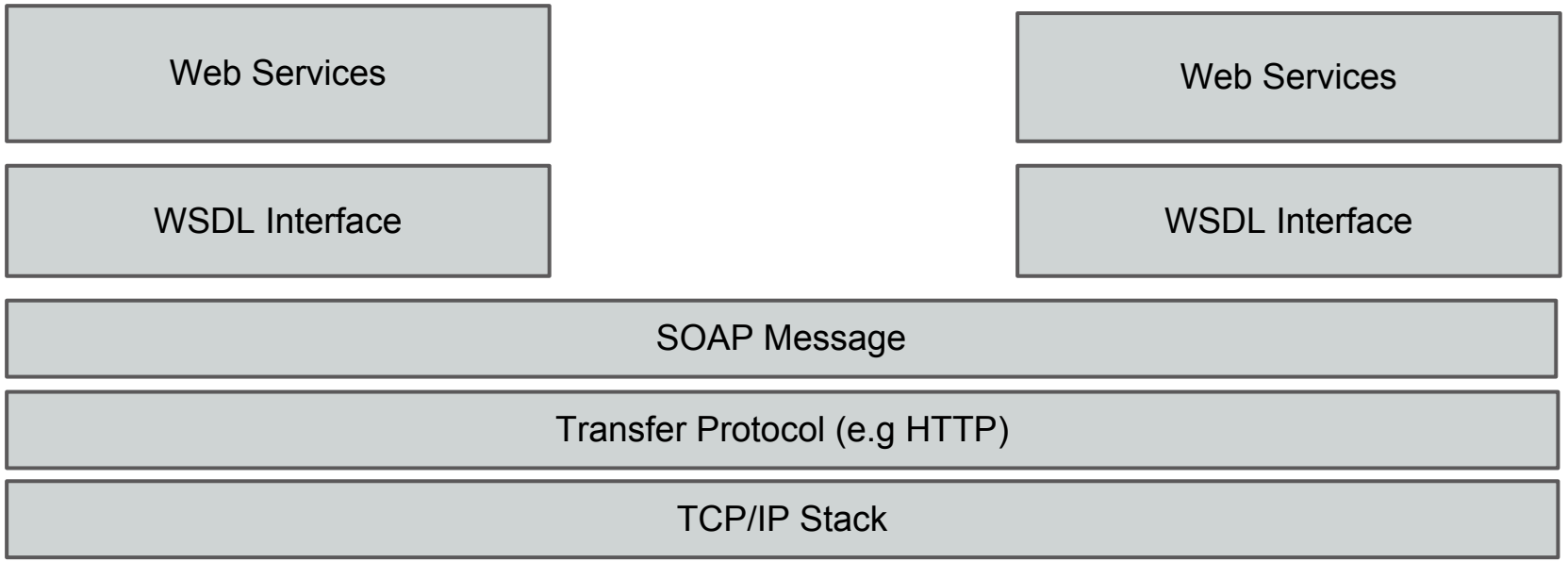
Well defined security mechanisms

Big Web Services

Objectives	Loose coupling of ***service consumer*** and ***service provider*** are goals of a more general ***services oriented architecture*** approach.
Assumptions	Interaction occurs by means of documents that make no assumptions about the technical capability of the service consumer or the service provider.
SOAP	Simple Object Access Protocol is an XML language defining a message and message formats. A SOAP document contains a top level XML element known as an envelope, which contains a header and body. XML Schema describes the structure of the SOAP messages at both endpoints and facilitates ***marshalling*** and ***unmarshalling*** of the message content at both endpoints.
WSDL	Web Service description language allows for definition of interfaces syntactically. ***port-types*** contain multiple ***abstract operations*** which are associated with incoming and outgoing *messages*. ***Bindings*** link the sets of abstract operations with concrete transport protocols and serialization formats.

SOAP

Message Format	Defining how a message can be packed into an XML document.
Description	Of how a message should be transported over the web
Set of Rules entities	That must be followed in processing of message, including definition of
Set of Conventions	How to turn an RPC call into a SOAP message and back.



Structure of a SOAP Message

Header processing	An optional element that contains blocks of information that are relevant to
Body	Main end-to-end payload.

Request

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-
envelope"
soap:encodingStyle="http://www.w3.
org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.
org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

Response

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-
envelope"
soap:encodingStyle="http://www.w3.
org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.
org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

Binding

```
POST /item HTTP/1.1
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 250
```

WSDL & JAX WS

Web Services Definition Language

Types	Container for type definitions used by the web service
Message	A typed definition of the data being communicated
PortType	A set of operations supported by one or more endpoints
Binding	Protocol and Data format spec for a port type

JAX-WS	Java API for creating web services that is part of the JEE specification. Uses annotations and tooling to facilitate the creation and deployment of web services clients and endpoints.
Implementation	JBoss provides an implementation of JAX-WS and also a set of tools for creating web-services. This removes a lot of the heavy lifting involved in creating Big web services from the developer.

WSDL Example -http://www.w3.org/2001/03/14-annotated-WSDL-examples.html

Why not Big Web Services

Too many layers of indirection - bloated and over engineered

Provides lots of standards but no implementation guidance

Fixing problems with tools is a symptom of a problem in itself

RESTFul Web-Services

REST	Representational State Transfer
URI	Resource identification through a uniform resource identifier
Interface	Resources are manipulated using a fixed set of CRUD operations PUT - Create a new resource GET - Retrieve the current state of a resource POST - Transfer new state onto a resource DELETE - Remove a resource
Messages	Messages are self descriptive and simple.
Stateless	Every interaction with a resource is stateless.

RESTFul Web-Services

Types — Much more loosely typed than SOAP/WSDL approach. This brings flexibility. Generally much less strictly implemented.

HTTP — Leverages the HTTP protocol, which gives us useful thing such as

- Caching
- Request/Response Model
- GET/PUT/POST/DELETE methods
- URI formatting using hyperlinks
- Lots of message formats and encodings
- Well formed error handling

Lightweight — RESTFul API's tend to be much more lightweight and agile than their "Big" web services counterparts.

The HTTP Protocol

Hypertext Transfer Protocol - HTTP 1.1 Standard

Request/Response Dialog that provides means of manipulation of remote **resources**

Well defined **methods** define the required action to take on the resource

Uniform resource identifiers to name and locate resources

Status codes for indicating success, failure or other interesting conditions

Stateless Protocol

Sessions via parameters or modification method bodies

The HTTP Protocol Methods

Safe Methods

GET - Retrieve the full resource state

HEAD - Retrieve header information related to the resource state only

TRACE - Echoes exactly the submitted request

OPTIONS - Returns the methods that are supported by the server

Methods that potentially Modify State

POST - Accept the entity submitted as a new subordinate of a resource

PUT - Store the entity at the required URI

DELETE - Delete the specified resource

PATCH - Apply partial modifications to an existing resource

Methods PUT and DELETE should be **idempotent**

The HTTP Protocol Request

Request Line

Defines the method and the resource

```
POST http://localhost:8080/users-1.0
```

Headers

Define meta information about the request

```
Content-Type: application/json
AuthToken: 837a9552-3903-3b3e-9fe0-88ca885f14c4
```

Body

Various information that results in the modification of a resource

```
{
     "name":"David Lynch",
     "email":"david.lynch@raglansoftware.com",
     "password" : "dave",
     "id":"2"
}
```

The HTTP Protocol Response

Request

```
GET http://localhost:8080/users-1.0/user/1
Content-Type: application/json
AuthToken: 837a9552-3903-3b3e-9fe0-88ca885f14c4
```

Response Status

```
200 OK                                          # Status codes give clues as to
```

Response Headers

```
                                                # the nature of the response
Content-Type:application/json
Date: Sat, 23 Mar 2013 11:04:54 GMT             # Headers contain useful info
Server: Apache-Coyote/1.1
```

Response Body

```
                                        # The resource body
{
    "name": "Stock Test User",
    "email": "stock@testuser.com",
    "id": "1",
    "password": "test",
}
```

The HTTP Protocol Status Codes

Well defined codes and groups that, particularly when combined with response headers, give the the client clues as to the nature of the progress of the request response dialog

1xx	Informational	101 Continue	103 Processing		
2xx	Success	200 OK	201 Created	202 Accepted	206 Partial
3xx	Redirection	300 Multiple Choice	301 Moved Permanently		
		303 See Other			
4xx	Client Error	400 Bad Request	401 Unauthorized	406 Not Acceptable	
		403 Forbidden	404 Not Found	405 Method Not Allowed	
		429 Too Many Requests		408 Request Timeout	
5xx	Server Error	500 Internal Server Error	503 Temporarily Unavailable		

JSON - Javascript Object Notation

Text based open standard - human readable

Types

String	Boolean
Array	Object
Null	

Very popular alternative to XML

Jackson is a Java <=> JSON parser that includes object mapping support

Can be used in conjunction with standard JAX-Bindings for automated **marshalling** and **unmarshalling** in Java Applications

```
{
    "firstName": "John",
    "lastName": "Smith",
    "age": 25,
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": 10021
    },
    "phoneNumbers": [
        {
            "type": "home",
            "number": "212 555-1234"
        },
        {
            "type": "fax",
            "number": "646 555-4567"
        }
    ]
}
```

Example - A User Management API

URI	**Method**	**Needs**	**Returns**	**Errors**
/users	POST	Entity	Status Code Only	User Exists `CONFLICT 409`

/users/{userId}	PUT	Auth Token, Entity	Status Code Only	No Credentials
				`NOT AUTHORIZED 401`
				User Doesn't Exist
				`NOT FOUND 404`

/users	GET	Auth Token	JSON List of Users	No Credentials `NOT AUTHORIZED 401`

/users/{userId}	DELETE	Auth Token	Status Code Only	No Credentials,
				`NOT AUTHORIZED 401`
				User Doesn't Exist
				`NOT FOUND 404`

/users/login	POST	Credentials	Auth Token and User ID	No credentials `NOT AUTHORIZED 401`

JAX-RS

Definition	Java for Restful Web Services Part of the Java EE 6 Specification
Annotations	@Path specifies the relative path for a resource class or method. @GET, @PUT, @POST, @DELETE and @HEAD specify the HTTP request type @Produces specifies the response Internet media types @Consumes specifies the accepted request Internet media types. @PathParam allows injection of URI path parameters @HeaderParam allows injection of HTTP header values
Implementations	JBoss Resteasy Well supported and actively maintained. Good Docs. Apache Jersey Well supported reference implementation. Slow. Apache CXF Obscure and poor documentation.

JBoss Resteasy Application

A web container such as JBoss AS 7 and a maven web-application configuration.

https://github.com/lynchd/rest-api/blob/master/pom.xml

Main

```
<dependency>
	<groupId>org.jboss.resteasy</groupId>
	<artifactId>resteasy-jaxrs</artifactId>
	<version>2.3.1.GA</version>
</dependency>
```

JBoss Repo

```
<repository>
	<id>JBoss repository</id>
	<url>http://repository.jboss.org/nexus/content/groups/public/</ur
</repository>
```

Plugin

```
<plugin>
	<groupId>org.jboss.as.plugins</groupId>
	<artifactId>jboss-as-maven-plugin</artifactId>
	<version>7.4.Final</version>
	<executions>
		<execution>
			<phase>install</phase>
			<goals>
				<goal>deploy</goal>
			</goals>
		</execution>
	</executions>
</plugin>
```

Testing

```
<dependency>
	<groupId>com.jayway.restassured</groupId>
	<artifactId>rest-assured</artifactId>
	<version>1.6</version>
	<scope>test</scope>
</dependency>
```

Application

Extend Application to Bootstrap

Define Interesting Objects in the HashSet

https://github.com/lynchd/rest-api/blob/master/src/main/java/ie/dit/users/UsersApplication.java

Resources

Each Resource should be represented by a single class

Annotations provide a means to instruct the web-container how to behave

https://github.com/lynchd/rest-api/blob/master/src/main/java/ie/dit/users/resources/LoginResource.java

https://github.com/lynchd/rest-api/blob/master/src/main/java/ie/dit/users/resources/UserResource.java

Model

The model should be a distinct layer, annotated with JAXB and JSON Marshalling related constructs.

https://github.com/lynchd/rest-api/tree/master/src/main/java/ie/dit/users/model

Data

I use the Singleton Pattern to fudge persistence

https://github.com/lynchd/rest-api/blob/master/src/main/java/ie/dit/users/data/repository/LoginRepository.java

https://github.com/lynchd/rest-api/blob/master/src/main/java/ie/dit/users/data/repository/UserRepository.java

Handling Exceptions

By default all exceptions that do not extend WebApplicationException are thrown back as default errors

We can intercept this by implementing an Exception Mapper

https://github.com/lynchd/rest-api/tree/master/src/main/java/ie/dit/users/exception

Testing - Rest Easy

Provides a Java DSL for integration testing against restful endpoints

https://github.com/lynchd/rest-api/tree/master/src/test/java/ie/dit/users/integration

Resources

Restful Java with JAX-RS - Bill Burke
http://shop.oreilly.com/product/9780596158057.do

Example Project
https://github.com/lynchd/rest-api

JBoss Rest-Easy
http://www.jboss.org/resteasy/

Jackson JSON Processor
http://jackson.codehaus.org/

Post-Man REST Client
https://chrome.google.com/webstore/detail/postman-rest-client/fdmmgilgnpjigdojojpjoooidkmcomcm?hl=en

The HTTP Protocol
http://www.w3.org/Protocols/rfc2616/rfc2616.html

Lab Session

Implement your **own version** of the API on slide 17

Replace my fudged persistence layer with something more permanent