Rest API

Hands on

Note on Final Project

- MEAN framework

OR

- **(if you are welling to explore features to support the requirement for the final project)**
  - **I may allow Java with written approval from the instructor**

IS 2560: Web Services Using SOAP, REST, WSDL, and UDDI

Graduate Program Information Science and Technology

School of Information Sciences

University of Pittsburgh

What’s a Web Service?

- A web service is just a web page meant for a computer to request and process
- More precisely, a Web service is a Web page that’s meant to be consumed by an *autonomous* program as opposed to a Web browser or similar UI tool

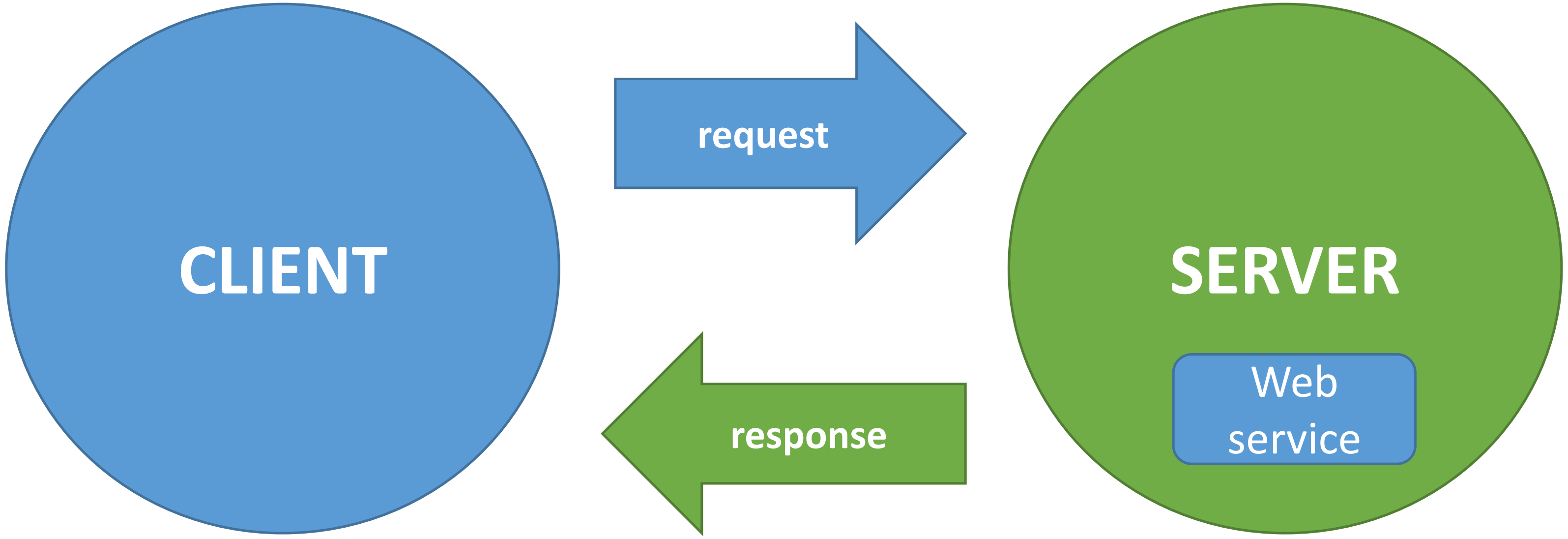
What Is A Web Service?

```
{
 "statuses": [
  {
   "coordinates": null,
   "favorited": false,
   "truncated": false,
   "created_at": "Mon Sep 24 03:35:21 +0000 2012",
   "id_str": "250075927172759552",
   "entities": {
    "urls": [

    ],
    "hashtags": [
     {
      "text": "freebandnames",
      "indices": [
       20,
       34
      ]
     }
    ],
    "user_mentions": [

    ]
   },
```

https://dev.twitter.com/docs/api/1.1/get/search/tweets

```
<breakfast_menu>
    <food>
        <name>Belgian Waffles</name>
        <price>$5.95</price>
        <description>
            Two of our famous Belgian Waffles with
            plenty of real maple syrup
        </description>
        <calories>650</calories>
    </food>
    <food>
        <name>Strawberry Belgian Waffles</name>
        <price>$7.95</price>
        <description>
            Light Belgian waffles covered with
            strawberries and whipped cream
        </description>
        <calories>900</calories>
    </food>
</breakfast_menu>
```

http://www.w3schools.com/xml/simple.xml



CLIENT

request

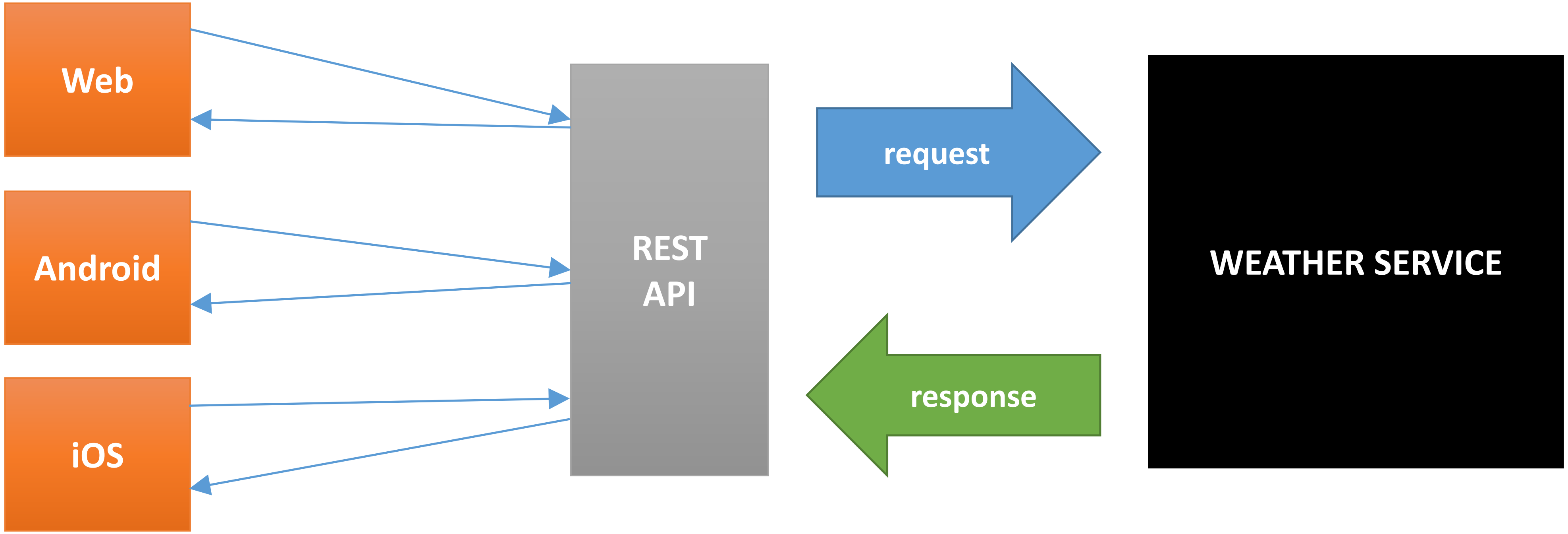
response

SERVER

Web service

What Is A Web Service

- Web services hide complexity
- Client agnostic
- Every modern programming language offers libraries for working with web services
- Many open-source frameworks that abstract working with web services
    - Angular.js
    - Backbone.js
    - Knockout.js
    - JQuery


Web
Android
iOS
REST API
request
response
WEATHER SERVICE

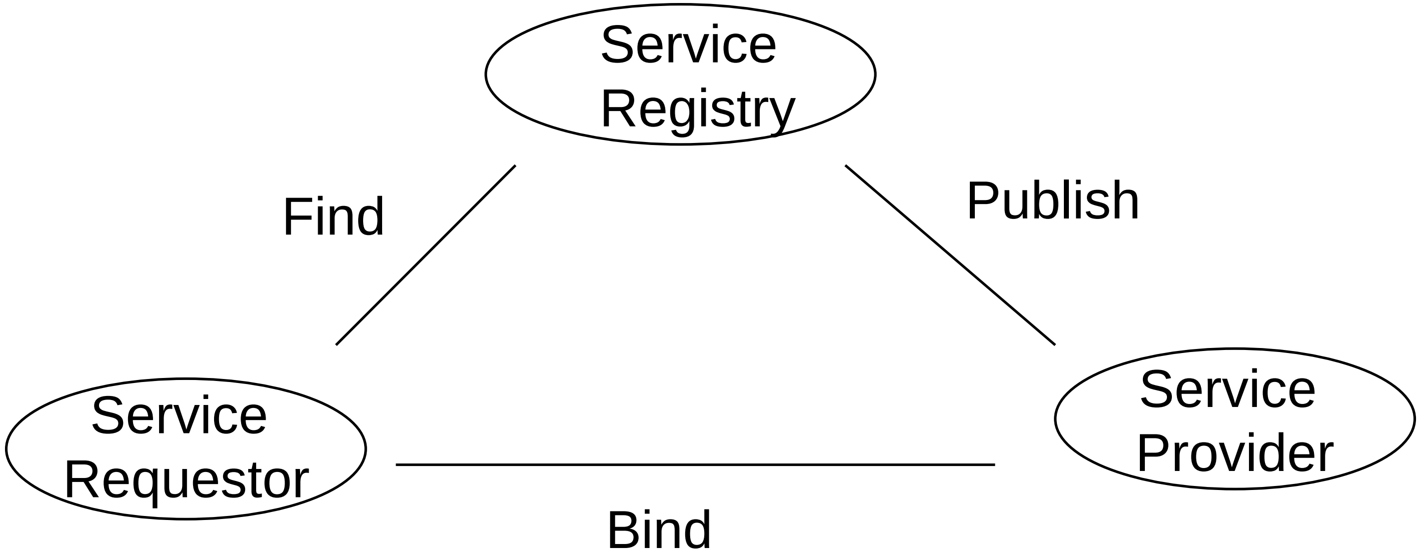
Why Web Services?

- From business standpoint
  - Integration
    - Within an organization
    - Between companies
    - Allows time/cost efficiencies
      - Purchase orders
      - Answering inquiries
      - Processing shipment requests
    - Do this without locking in to a single partner

Example

- Eastman Company
  - Obtain catalog information through
    - Web scraping
    - Email from Eastman with files
  - Catalog updates regularly –never on schedule basis
    - Distributors are left with outdated information
  - Solution -> Web service
    - Distributor can get access to product catalog
    - Push that access to their customers so everyone has the same catalog

Web Service Architecture



- Service-Oriented Architecture

Metaphor

- Restaurant
- Customer -> Service Requester
- Restaurant it self-> Service Provider
- Restaurant Menu ->WSDL/ Service Registry
- Staff-> UDDI Locate resources

XML Leveraging Features

- XML Namespaces
  - Collision
    - Common XML element names
      - Application specific or embedded in message?
  - Allows composition of multiple XML documents
    - Identifies elements belonging to the same document type

XML Leveraging Features II

- XML Schemas
    - Alternative to DTDs for describing document structure
    - Written in XML
        - Simple types
        - Complex types
    - Reusable
        - Intended to be used with namespaces

Web service Message Protocol

- SOAP (Simple Object Access Protocol)
  - XML-based protocol
- REST (REpresentational State Transfer)
  - HTTP Based (Resources and URI)

SOAP

- **S**imple **O**bject **A**ccess **P**rotocol
- Web service messaging and invocation
- 2nd Generation XML Protocol
    - Takes advantage of
        - XML Namespaces
        - XML Schema

First Generation XML Protocol

- Based on XML 1.0
- Example: XML-RPC
  - Introduced by Userland in 1998
  - Uses HTTP as underlying transport

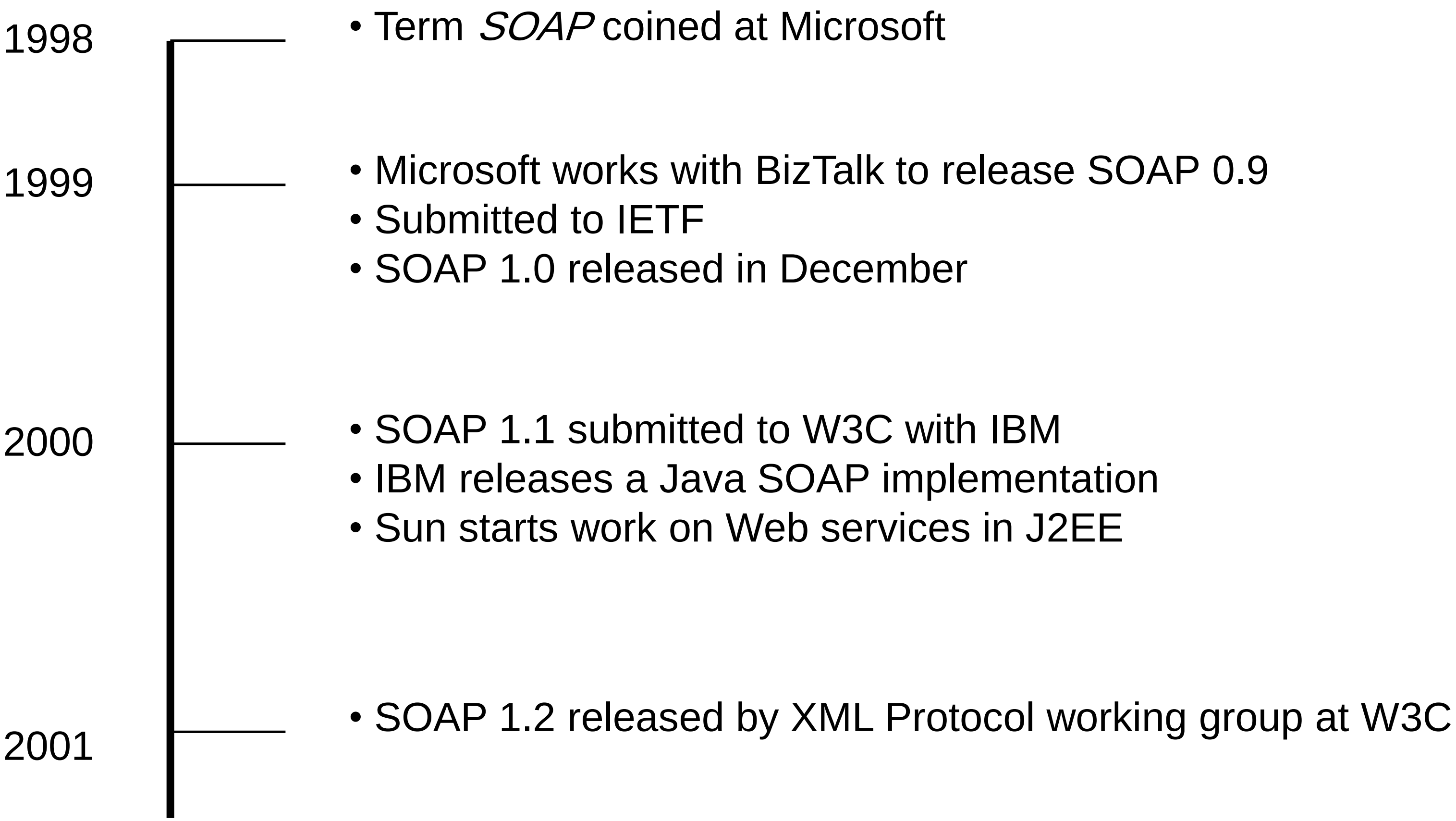
Call

```
<methodCall>
  <methodName>NumberToText</methodName>
   <params>
       <param>
          <value><i4>28</i4></value>
       </param>
   </params>
</methodCall>
```

Response

```
<methodResponse>
  <params>
    <param>
      <value>
        <string>twenty-eight</string>
      </value>
    </param>
  </params>
</methodResponse>
```

SOAP History



1998

- Term *SOAP* coined at Microsoft

1999

- Microsoft works with BizTalk to release SOAP 0.9
- Submitted to IETF
- SOAP 1.0 released in December

2000

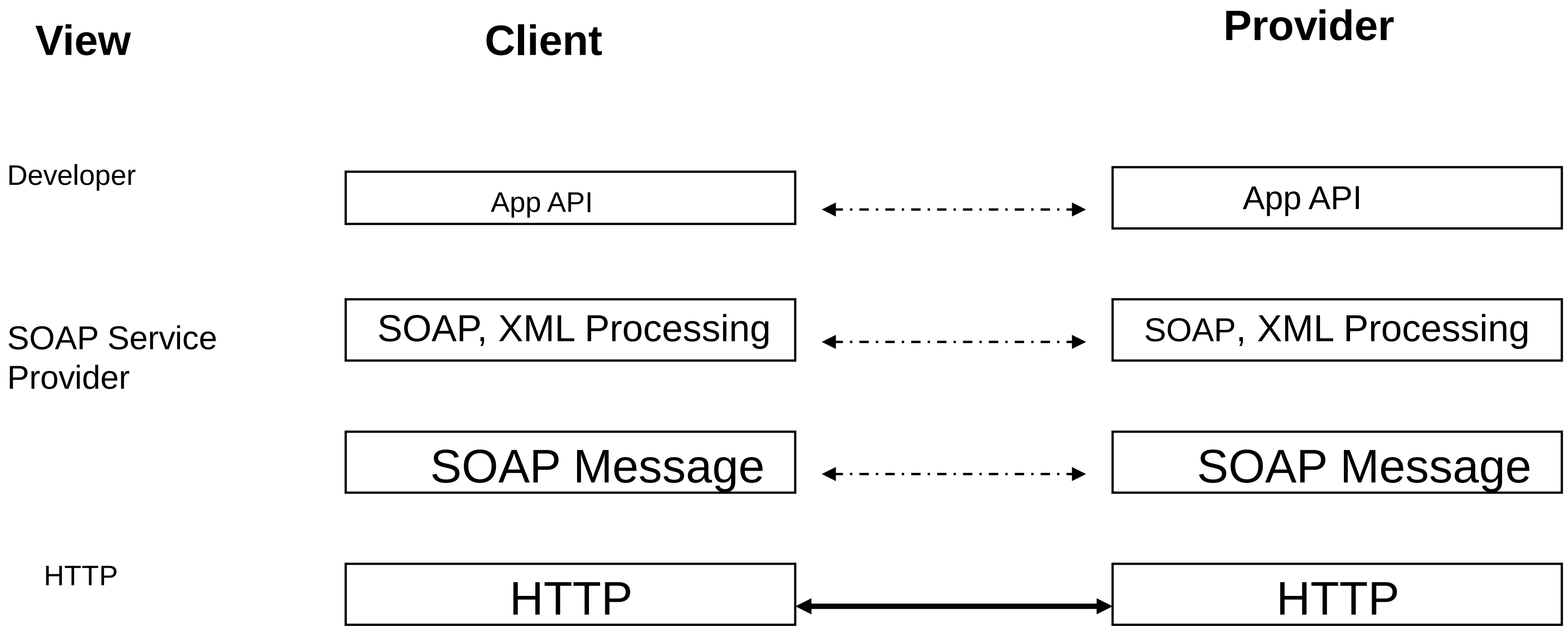
- SOAP 1.1 submitted to W3C with IBM
- IBM releases a Java SOAP implementation
- Sun starts work on Web services in J2EE

2001

- SOAP 1.2 released by XML Protocol working group at W3C

Currently, about 80+ SOAP implementations available including Apple…

SOAP Messaging Layers



View

Client

Provider

Developer

App API

App API

SOAP Service Provider

SOAP, XML Processing

SOAP, XML Processing

SOAP Message

SOAP Message

HTTP

HTTP

HTTP

SOAP Message

```
  <?xml version="1.0" encoding="UTF-8"?>
  <Envelope>

     <Header>


      </Header>


      <Body>


     </Body>




</Envelope>
```

SOAP Envelope

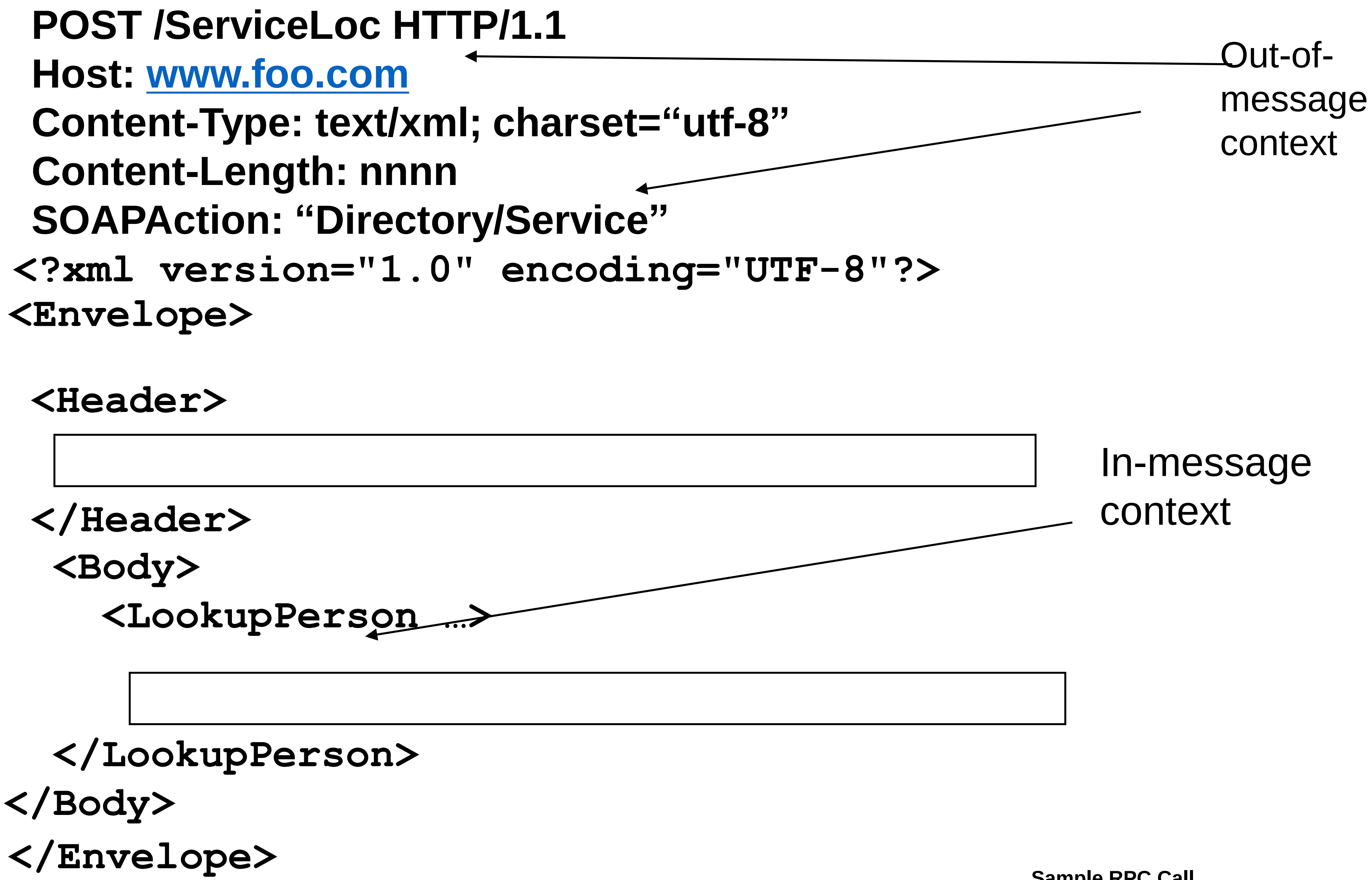
- Root element
- Mandatory
- Does not expose any protocol versions
  - Protocol version is the URI of SOAP envelope namespace
  - encodingStyle attribute for complex types

```
<SOAP-ENV:Envelope
   SOAP-ENV:encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
   xmlns="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
```

SOAP Body

- Can contain arbitrary XML
- Conventions for
    - RPCs
    - Faults
        - Faultcode – lookup string
        - Faultstring – human readable string
        - Faultactor – where in the message path
        - Detail – optional
    - Data encoding

SOAP Protocol Binding: HTTP



```
POST /ServiceLoc HTTP/1.1
Host: www.foo.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Directory/Service"
<?xml version="1.0" encoding="UTF-8"?>
<Envelope>

 <Header>

 </Header>
  <Body>
    <LookupPerson …>

  </LookupPerson>
</Body>
</Envelope>
```

Out-of-message context

In-message context

Sample RPC Call

Other SOAP Protocol Bindings

- HTTPS
  - Similar to HTTP
    - Use POST
    - Return 200 for success
    - 500 for failure + SOAP fault
    - SOAPAction HTTP header for hint
    - MIME media type: text/html
- SMTP
- SOAP messages with Attachments

WSDL

- **W**eb **S**ervice **D**efinition **L**anguage
- Predecessors include
  - COM, CORBA IDLs
  - Network Accessible Service Specification Language (IBM)
  - SOAP Contract Language (Microsoft)
  - First submitted to W3C in Sep 2000
- Current version is 2.0
- Changed the definition to
- Web Service Description language

WSDL

- Define a web service in WSDL by
  - Writing an XML document conforming to the WSDL specs
- Describes three fundamental properties
  - What a service does
    - Operations (methods) provided by the service
  - How a service is accessed
    - Data format and protocol details
  - Where a service is located
    - Address (URL) details

WSDL Components

definitions	
types	All the data types used by the Web service
message	Parameters and messages used by method
operation	Abstract interface definition – each *operation* element defines a method signature
binding	Binds abstract methods to specific protocols
service ***port***	A service is a collection of ports. A port is a specific method and its URI

Sample WSDL: getQuote

```
<?xml version="1.0" encoding="UTF-8" ?>

<definitions name="net.xmethods.services.stockquote.StockQuote"
targetNamespace="http://www.themindelectric.com/wsdl/net.xmethods.s
ervices.stockquote.StockQuote/"

xmlns:tns="http://www.themindelectric.com/wsdl/net.xmethods.service
s.stockquote.StockQuote/"
   xmlns:electric="http://www.themindelectric.com/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns="http://schemas.xmlsoap.org/wsdl/">

<message name="getQuoteResponse1">
    <part name="Result" type="xsd:float" />
</message>

<message name="getQuoteRequest1">
    <part name="symbol" type="xsd:string" />
</message>
```

Sample WSDL: getQuote

```
<portType name="net.xmethods.services.stockquote.StockQuotePortType">
   <operation name="getQuote" parameterOrder="symbol">
     <input message="tns:getQuoteRequest1" />
     <output message="tns:getQuoteResponse1" />
   </operation>
</portType>

<binding name="net.xmethods.services.stockquote.StockQuoteBinding“
         type="tns:net.xmethods.services.stockquote.StockQuotePortType">
    <soap:binding style="rpc"
                  transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="getQuote">
       <soap:operation soapAction="urn:xmethods-delayed-quotes#getQuote" />
         <input>
           <soap:body use="encoded" namespace="urn:xmethods-delayed-quotes"
             encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
         </input>
         <output>
           <soap:body use="encoded" namespace="urn:xmethods-delayed-quotes"
             encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
         </output>
    </operation>
</binding>
```

Sample WSDL: getQuote

```
<service
name="net.xmethods.services.stockquote.StockQuoteService">
  <documentation>net.xmethods.services.stockquote.StockQuote web
service
  </documentation>
   <port name="net.xmethods.services.stockquote.StockQuotePort"

binding="tns:net.xmethods.services.stockquote.StockQuoteBinding">
     <soap:address location="http://64.39.29.211:9090/soap" />
   </port>
</service>

</definitions>
```

Overall Issues

- Interoperability
- Web Services Everywhere
  - Peer to peer vs centralized

Web Service API

- Message format
- Request syntax
- Parameters
- HTTP methods
- Authentication
- Data format
- Content and metadata

Message Format

- **SOAP** – Simple Object Access Protocol. http://en.wikipedia.org/wiki/SOAP
- **XML** – eXtensible Markup Language. http://www.w3schools.com/xml/
- **JSON** – JavaScript Object Notation

Request Methods, Syntax, Parameters

- HTTP methods: Get vs. Post
- Method calls in SOAP
  - Method signatures (names, arguments)
  - Return data types
  - Return data format
- URIs in RESTful
  - Query parameters
  - XML or JSON
  - Return data format

Data format

- XML
- JSON
- Custom?

XML

- XML - a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.
- Tag - A markup construct that begins with < and ends with >. Tags come in three flavors:
  - start-tags; for example: <section>
  - end-tags; for example: </section>
  - empty-element tags; for example: <line-break />

http://en.wikipedia.org/wiki/Xml

XML

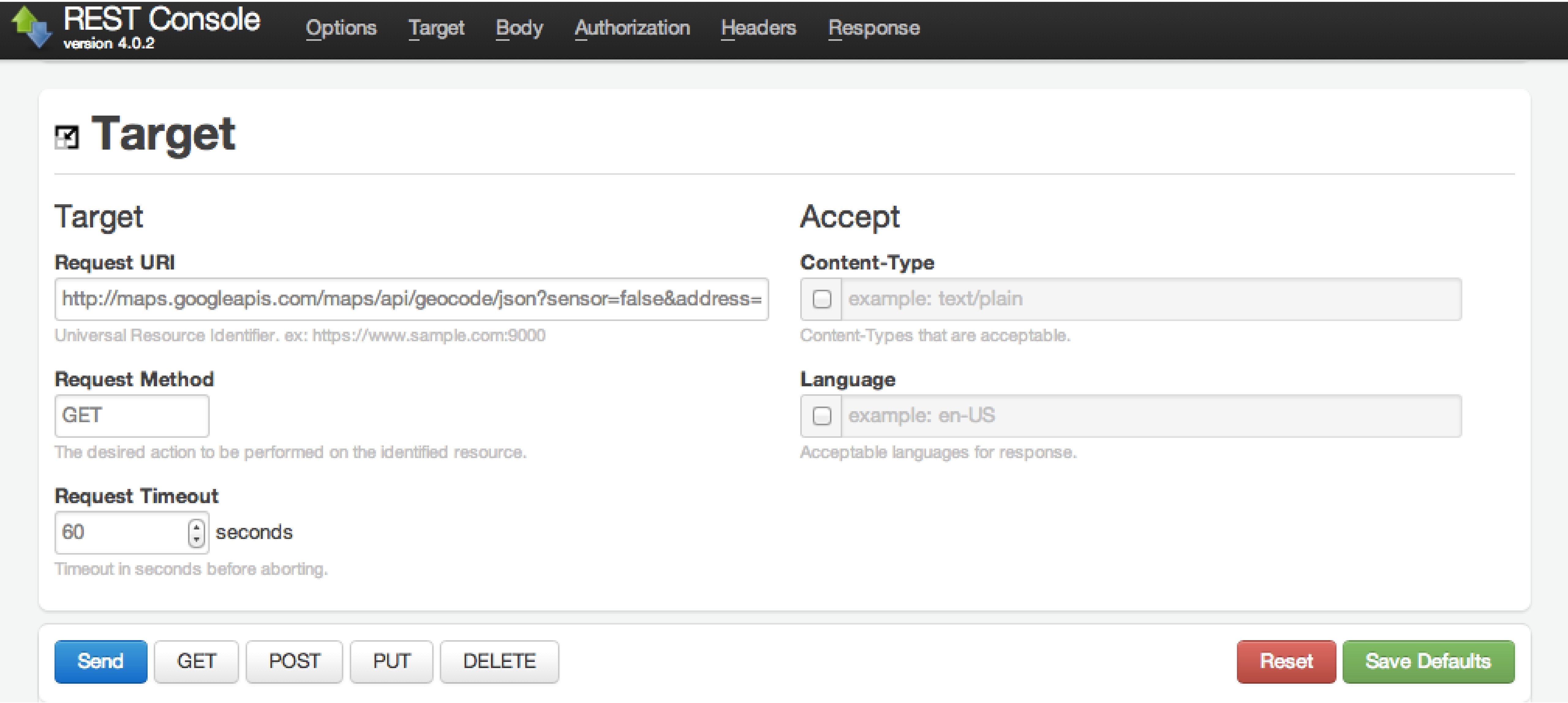
- Element - A logical document component which either begins with a start-tag and ends with a matching end-tag or consists only of an empty-element tag.
  - An example of an element is <Greeting>Hello, world.</Greeting>
- Attribute - A markup construct consisting of a name/value pair that exists within a start-tag or empty-element tag.
  - <img src="madonna.jpg" alt='Foligno Madonna, by Raphael'/>
  - <step number="3">Connect A to B.</step>

http://en.wikipedia.org/wiki/Xml

```xml
<prescriptionList>
    <prescription>
        <prescriptionID>0001</prescriptionID>
        <prescriptionDate>04/11/2013</prescriptionDate>
        <patientID>14343</patientID>
        <medicalProviderID>45465</medicalProviderID>
        <medicationList>
            <medication>
                <medicationID>12345</medicationID>
                <medicationName>Tylenol</medicationName>
                <usageDirections>Take 2 for headache</usageDirections>
                <maximumDosage>1000</maximumDosage>
            </medication>
            <medication>
                <medicationID>12346</medicationID>
                <medicationName>Thyrogen</medicationName>
                <usageDirections>Take 1 at least an hour before a meal</usageDirections>
                <maximumDosage>175</maximumDosage>
            </medication>
        </medicationList>
        <instruction>Do not take with food.  Best taken in the morning before breakfast.</instruction>
    </prescription>
</prescriptionList>
```

Request Syntax

- Install a REST console plugin for Google Chrome
- Copy and paste the following URI into the **Target Request URI** field http://maps.googleapis.com/maps/api/geocode/json?sensor=false&address=Pittsburgh
- Set **Request Method** field to GET
- Scroll down and click **Send**



Target

Target

Request URI

http://maps.googleapis.com/maps/api/geocode/json?sensor=false&address=

Universal Resource Identifier. ex: https://www.sample.com:9000

Request Method

GET

The desired action to be performed on the identified resource.

Request Timeout

60 seconds

Timeout in seconds before aborting.

Accept

Content-Type

□ example: text/plain

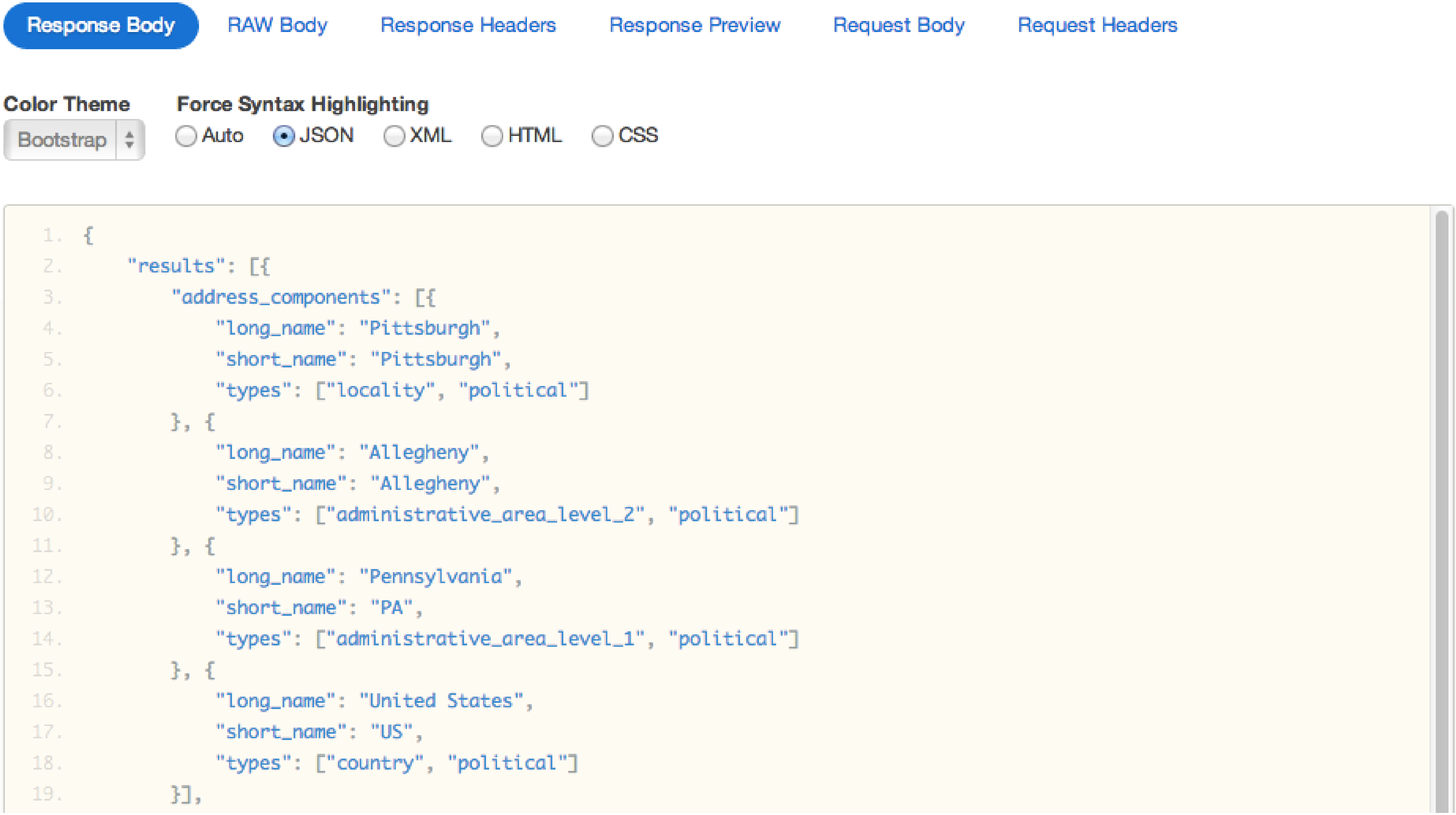
Content-Types that are acceptable.

Language

□ example: en-US

Acceptable languages for response.

You will see the response body:



Color Theme

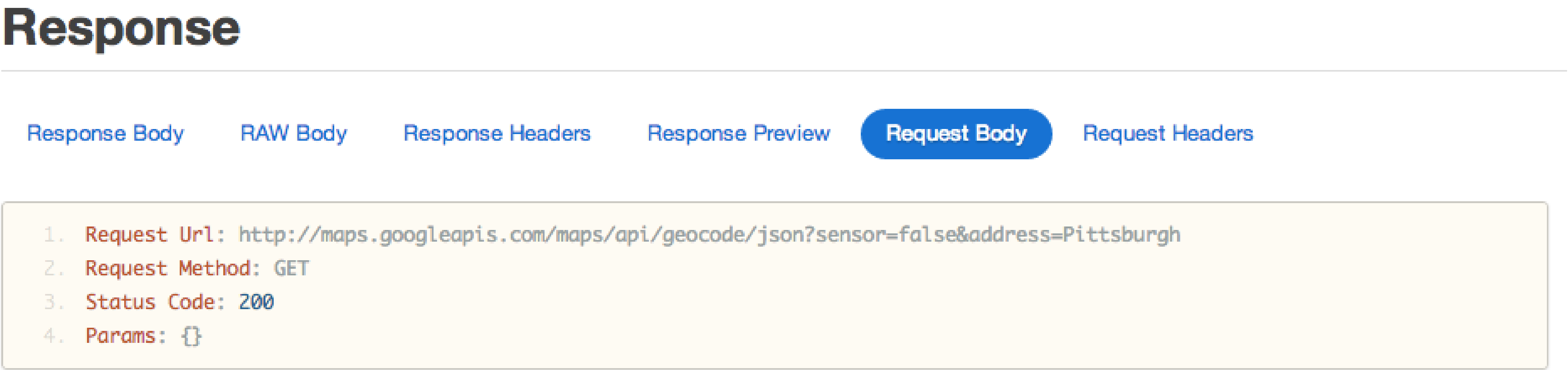
Bootstrap

Force Syntax Highlighting

○ Auto ● JSON ○ XML ○ HTML ○ CSS

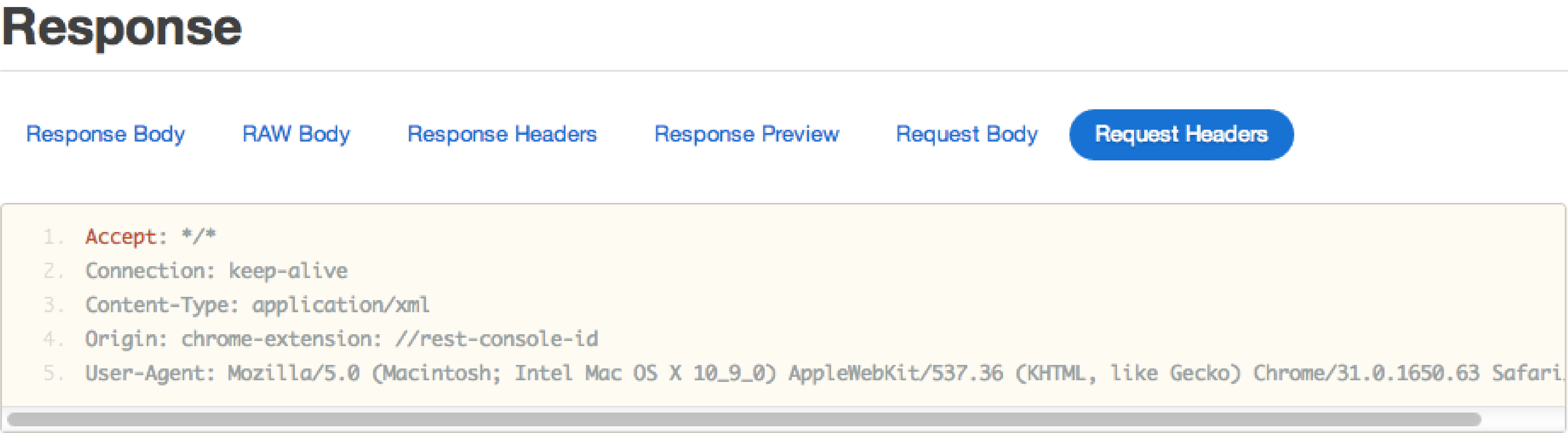
```
{
    "results": [{
        "address_components": [{
            "long_name": "Pittsburgh",
            "short_name": "Pittsburgh",
            "types": ["locality", "political"]
        }, {
            "long_name": "Allegheny",
            "short_name": "Allegheny",
            "types": ["administrative_area_level_2", "political"]
        }, {
            "long_name": "Pennsylvania",
            "short_name": "PA",
            "types": ["administrative_area_level_1", "political"]
        }, {
            "long_name": "United States",
            "short_name": "US",
            "types": ["country", "political"]
        }],
```

You can look at the request body (to see what was actually sent):



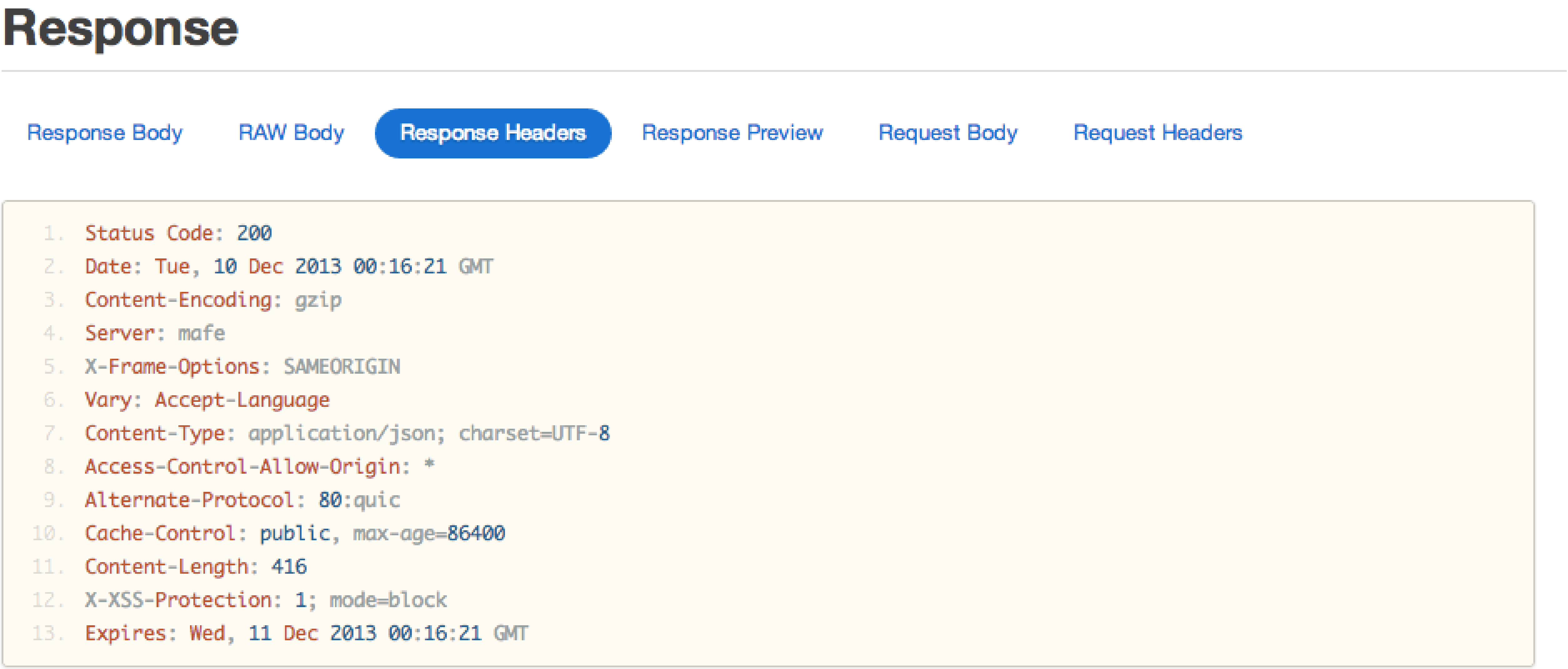
```
Request Url: http://maps.googleapis.com/maps/api/geocode/json?sensor=false&address=Pittsburgh
Request Method: GET
Status Code: 200
Params: {}
```

… and at the request header:



```
Accept: */*
Connection: keep-alive
Content-Type: application/xml
Origin: chrome-extension: //rest-console-id
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/31.0.1650.63 Safari
```

You can also see response headers:



```
Status Code: 200
Date: Tue, 10 Dec 2013 00:16:21 GMT
Content-Encoding: gzip
Server: mafe
X-Frame-Options: SAMEORIGIN
Vary: Accept-Language
Content-Type: application/json; charset=UTF-8
Access-Control-Allow-Origin: *
Alternate-Protocol: 80:quic
Cache-Control: public, max-age=86400
Content-Length: 416
X-XSS-Protection: 1; mode=block
Expires: Wed, 11 Dec 2013 00:16:21 GMT
```

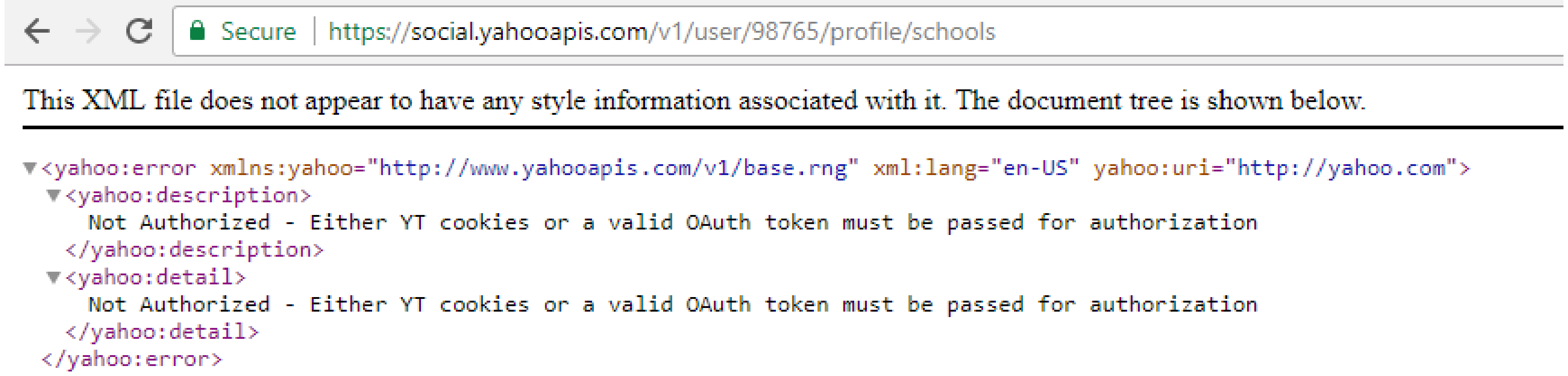
RESTful Web Services

REST

Representational **S**tate **T**ransfer

RESTful Web Services

- Style of software architecture
- Highly scalable
- Generality of interfaces
- Independent deployment of components



https://social.yahooapis.com/v1/user/98765/profile/schools

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<yahoo:error xmlns:yahoo="http://www.yahooapis.com/v1/base.rng" xml:lang="en-US" yahoo:uri="http://yahoo.com">
  ▼<yahoo:description>
     Not Authorized - Either YT cookies or a valid OAuth token must be passed for authorization
   </yahoo:description>
  ▼<yahoo:detail>
     Not Authorized - Either YT cookies or a valid OAuth token must be passed for authorization
   </yahoo:detail>
 </yahoo:error>
```

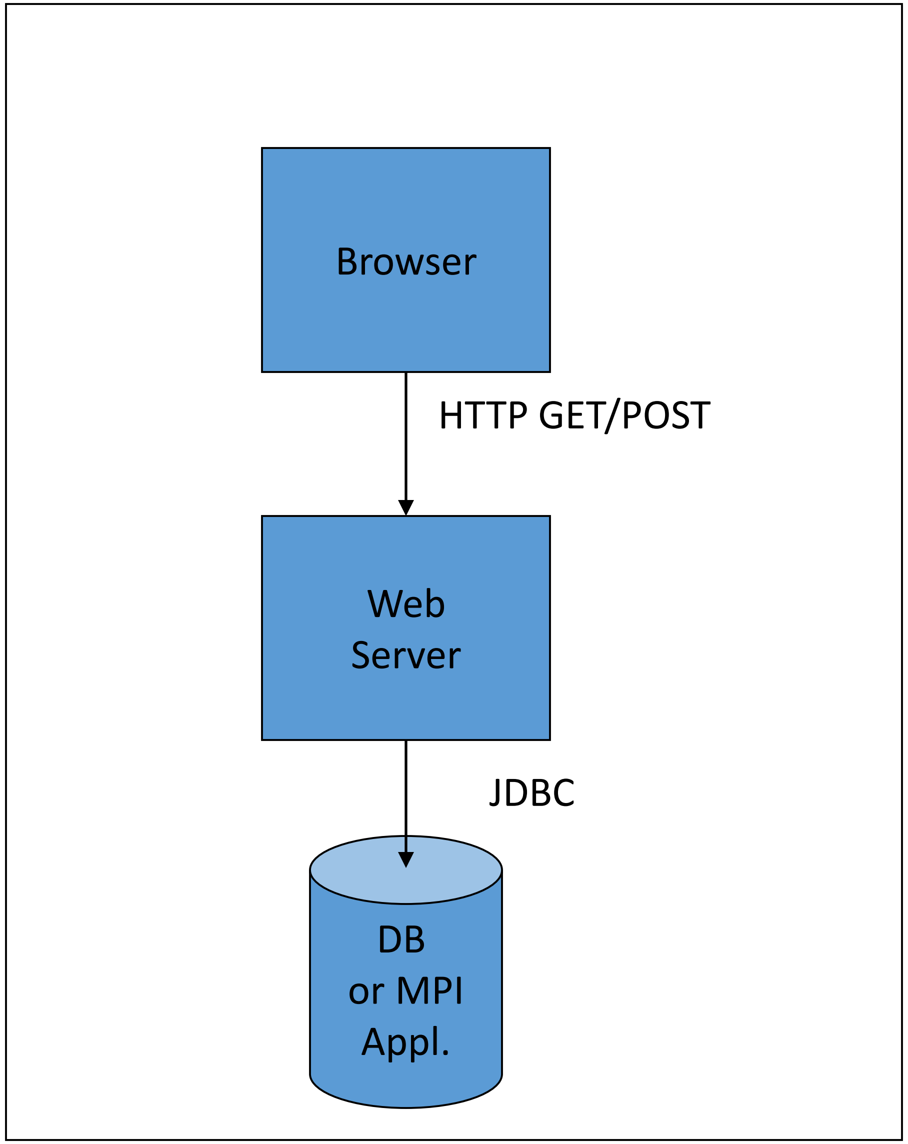
IS 2560: Web Services

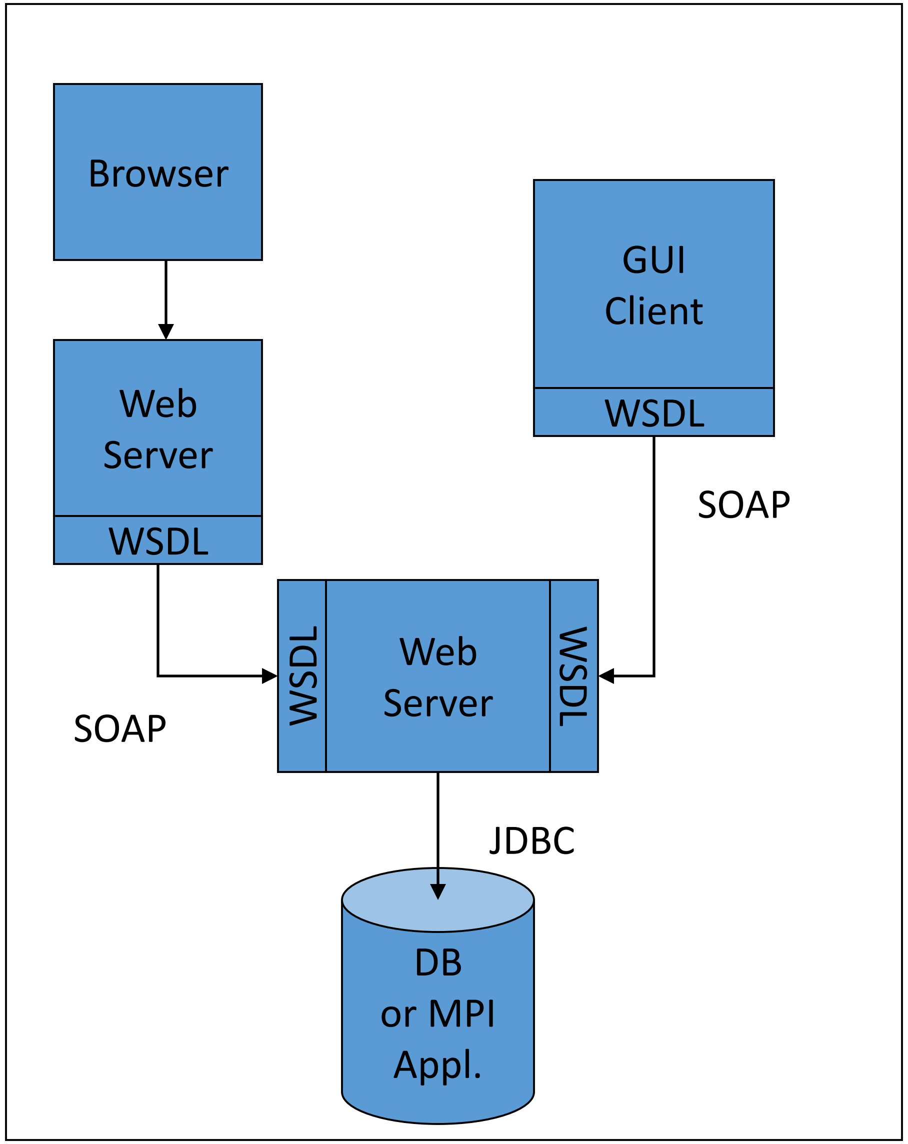
Graduate Program Information Science and Technology

School of Information Sciences

University of Pittsburgh

Basic Architectures: Servlets/CGI and Web Services





Web Services are Hard

- Java – OO Programming Paradigm
- Web Services – Message Exchange Paradigm

…

creates an

….

- Impedance Mismatch

REST vs. SOAP

	REST	SOAP
Message Format	XML[1]	SOAP
Interface Definition	None[2]	WSDL
Transport	HTTP	HTTP[3], FTP, MIME, JMS, SMTP, etc.

[1] Also uses HTTP headers and query string.

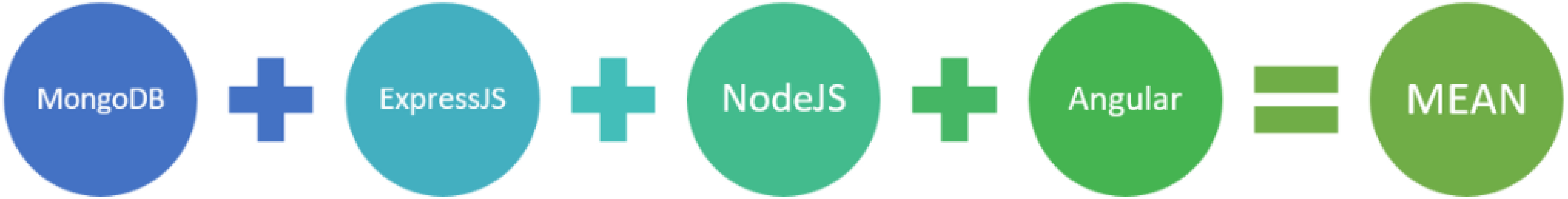
[2] XML Schema sometimes provided. And “out of band” documentation.

[3] Without WS-Addressing, SOAP relies on the message transport for dispatching (e.g., HTTP context path).

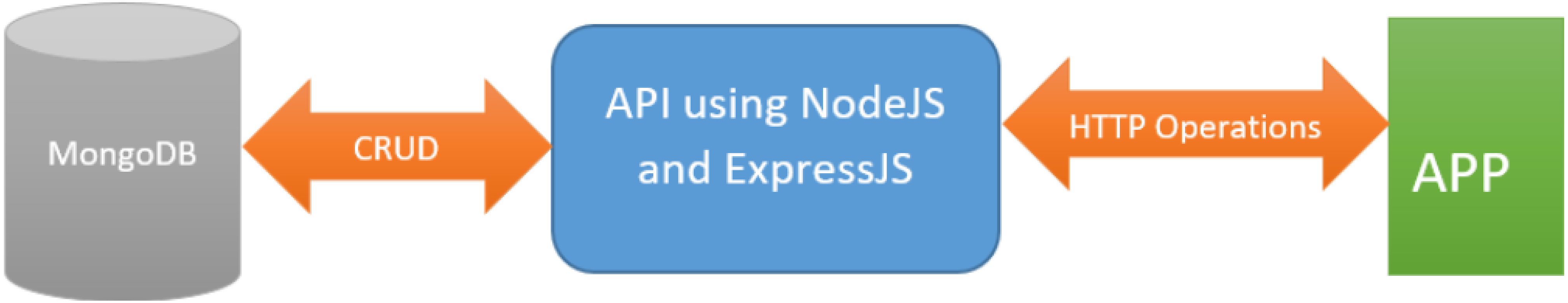
REST vs. SOAP

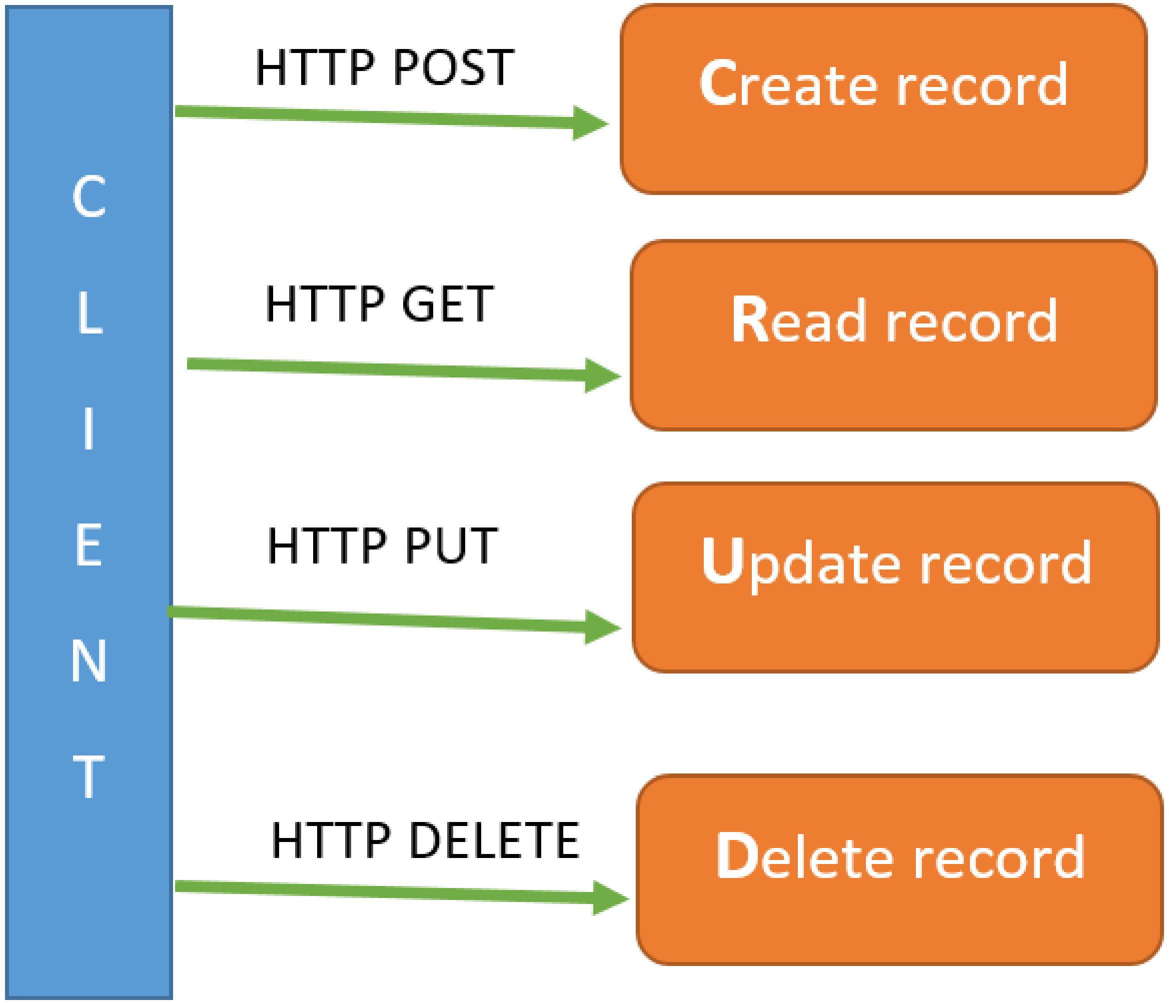
- REST is best when …
  - Rapid prototyping and quick demos for endusers are important.
  - Data is not highly structured or well defined by a schema – so you want to experiment and see the data in a browser and write code based on that.
- SOAP is best when …
  - Bullet-proof integration of systems is important.
  - Well defined application interfaces are needed.
  - Data conforms to a schema.
  - QoS (e.g., guaranteed delivery) issues are important.

Demo Step by Step



MongoDB + ExpressJS + NodeJS + Angular = MEAN


MongoDB
CRUD
API using NodeJS and ExpressJS
HTTP Operations
APP



CLIENT

HTTP POST

Create record

HTTP GET

Read record

HTTP PUT

Update record

HTTP DELETE

Delete record

Mongo db compass

- Visual Interface for Mongodb

REST API Demo

Computer to Computer communication

- Our Application will
- Handle CRUD for an item (we will use Product)
- Have a standard URL
  - http://example.com/api/product
- Support proper HTTP verbs to make it RESTful
  - GET,POST,PUT and delete
- Return JSON data
- Log all request to console

Before you begin

- You should have nodejs and npm installed
  - Npm come with Nodejs
- You should have MongoDB installed
  - You can install MongoCompass (visual Interface)

- Command
  - node –v
  - npm -v

Step 1

- Command Line- navigate to where you want you project Directory
- Instructor preference
  - Under the user directory c:\users\Alawya\
  - Mkdir restAPIDemo
  - Cd restAPIDemo

Now you should be in our working directory restAPIDemo

Step 2 using EXPRESS to create web app

- Run the command npm init
- This utility will ask you to input the information for Package.json
  - Name
  - Description
  - Version
  - etc

package.json

- Important file
- Tells npm
  - What your project is
  - What are the dependencies
  - Other metadata, project description
- Written in JSON
- Located at the root directory of your project

- When you call the command npm install=> create your program and create all subdirectories related to your project
  - Dependencies are installed under node_module

Example

```
{
"name" : "underscore",
"description" : "JavaScript's functional programming helper library.",
"homepage" : "http://documentcloud.github.com/underscore/",
"keywords" : ["util", "functional", "server", "client", "browser"],
"author" : "Jeremy Ashkenas <jeremy@documentcloud.org>",
"contributors" : [],
"dependencies" : [],
"repository" : {"type": "git", "url":
"git://github.com/documentcloud/underscore.git"},
"main" : "underscore.js",
"version" : "1.1.6"
}
```

Install

```
$ npm install express –save
```

We used **--save** flag to put it into **package.json** file as a dependency for this project

```
$ npm install --save mongoose node-restful
$ npm install --save body-parser
```

Nodemon

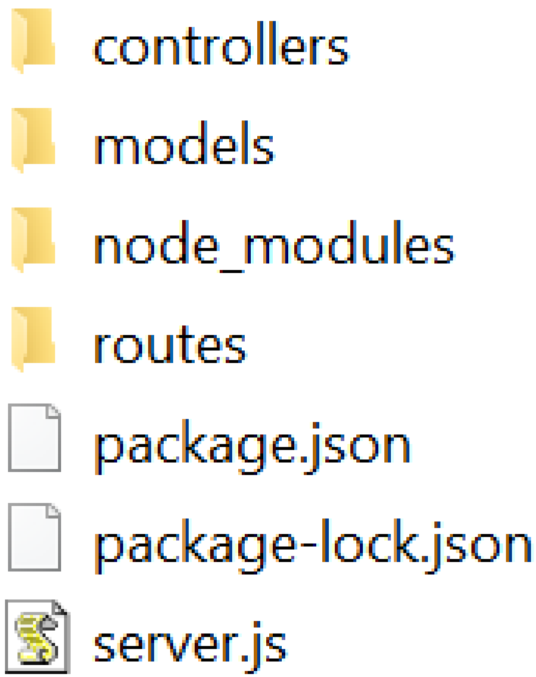
This utility will help you, it will restart your app automatically each time you make a change

- $npm install -g nodemon

Files we will create

- MEANS framework utilizes MVC architecture
- We need three folder
  - controllers
  - models
  - routes
- We will create three files
- server.js
- routes\api.js
- models\product.js

Directory structure



controllers
models
node_modules
routes
package.json
package-lock.json
server.js

Server.js

```
// Dependencies
var express = require('express');
var mongoose = require('mongoose');
var bodyParser = require('body-parser');
// MongoDB
mongoose.connect('mongodb://localhost/rest_test');
// Express
var app = express();
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
// Routes
app.use('/api', require('./routes/api'));
// Start server
app.listen(3000);
console.log('Listening on port 3000...');
```

Models/product.js

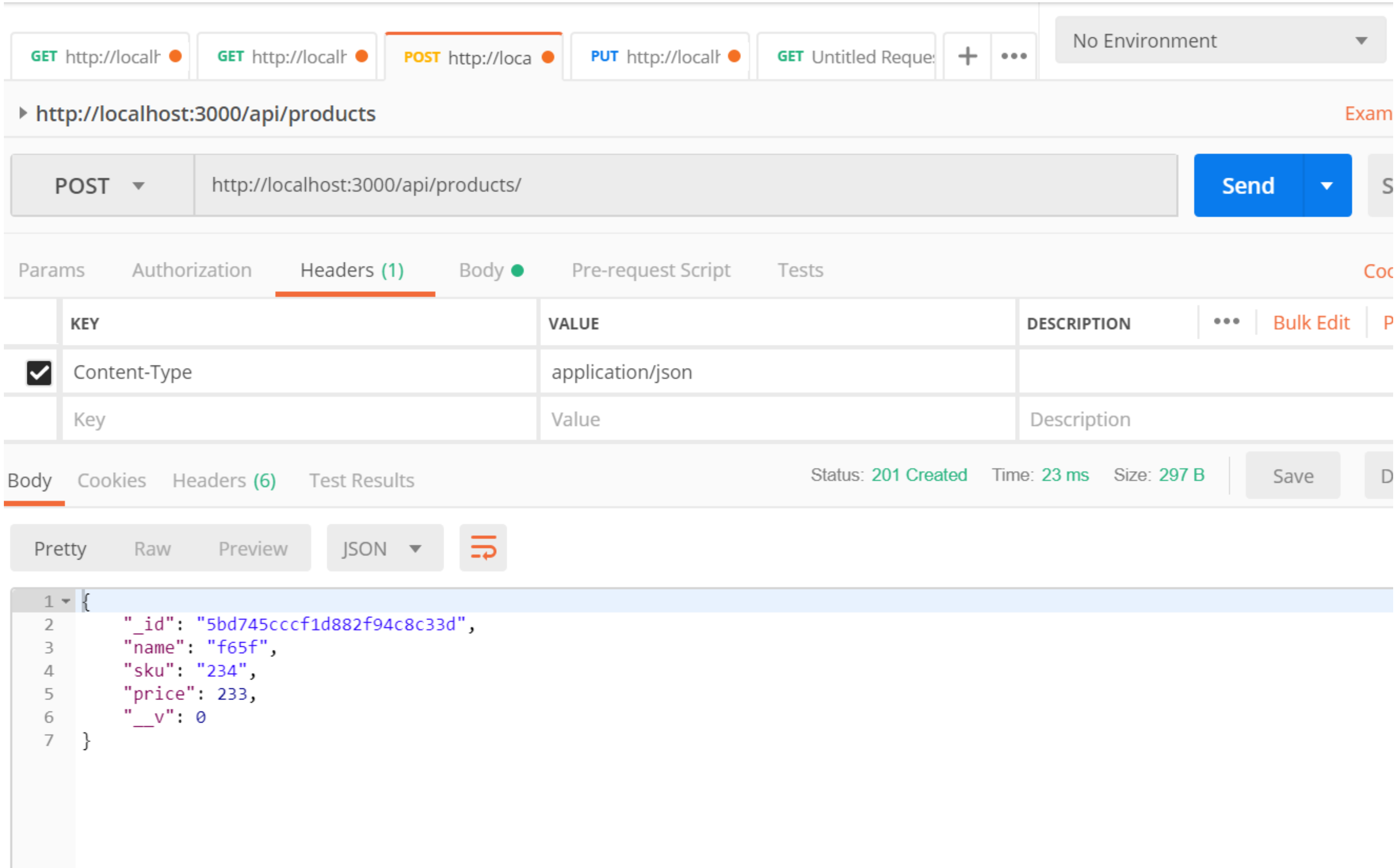
```
// Dependencies
var restful = require('node-restful');
var mongoose = restful.mongoose;

// Schema
var productSchema = new mongoose.Schema({
    name: String,
    sku: String,
    price: Number
});

// Return model
module.exports = restful.model('Products', productSchema);
```

Routes/api.js

```
// Dependencies
var express = require('express');
var router = express.Router();

// Models
var Product = require('../models/product');
// Routes
Product.methods(['get', 'put', 'post', 'delete']);
Product.register(router, '/products');

// Return router
module.exports = router;
```

Postman

- Helps with testing the API



http://localhost:3000/api/products

POST http://localhost:3000/api/products/

KEY	VALUE	DESCRIPTION
☑ Content-Type	application/json	
Key	Value	Description

Status: 201 Created Time: 23 ms Size: 297 B

```
{
    "_id": "5bd745cccf1d882f94c8c33d",
    "name": "f65f",
    "sku": "234",
    "price": 233,
    "__v": 0
}
```

Resources

- There are a lot of resources in the web
  - Google RESTFUL API using MEAN
- https://medium.com/@debug_mode/step-by-step-building-node-js-based-rest-api-to-perform-crud-operations-on-mongodb-ab18835111d7

Assignment 4 (Last assignment)

- Modify the API we did in the class which has one route for all http request.
- Create different routes for different requests.

router.route('/products/:product_id').put(function (req, res)

- Your app should
  - Add authentication. (optional)
  - Get Allow for finding product by ID
  - Put to update specific record by ID
  - Delete
  - POST
- You can follow [this tutorial]()