Service-Oriented Programming

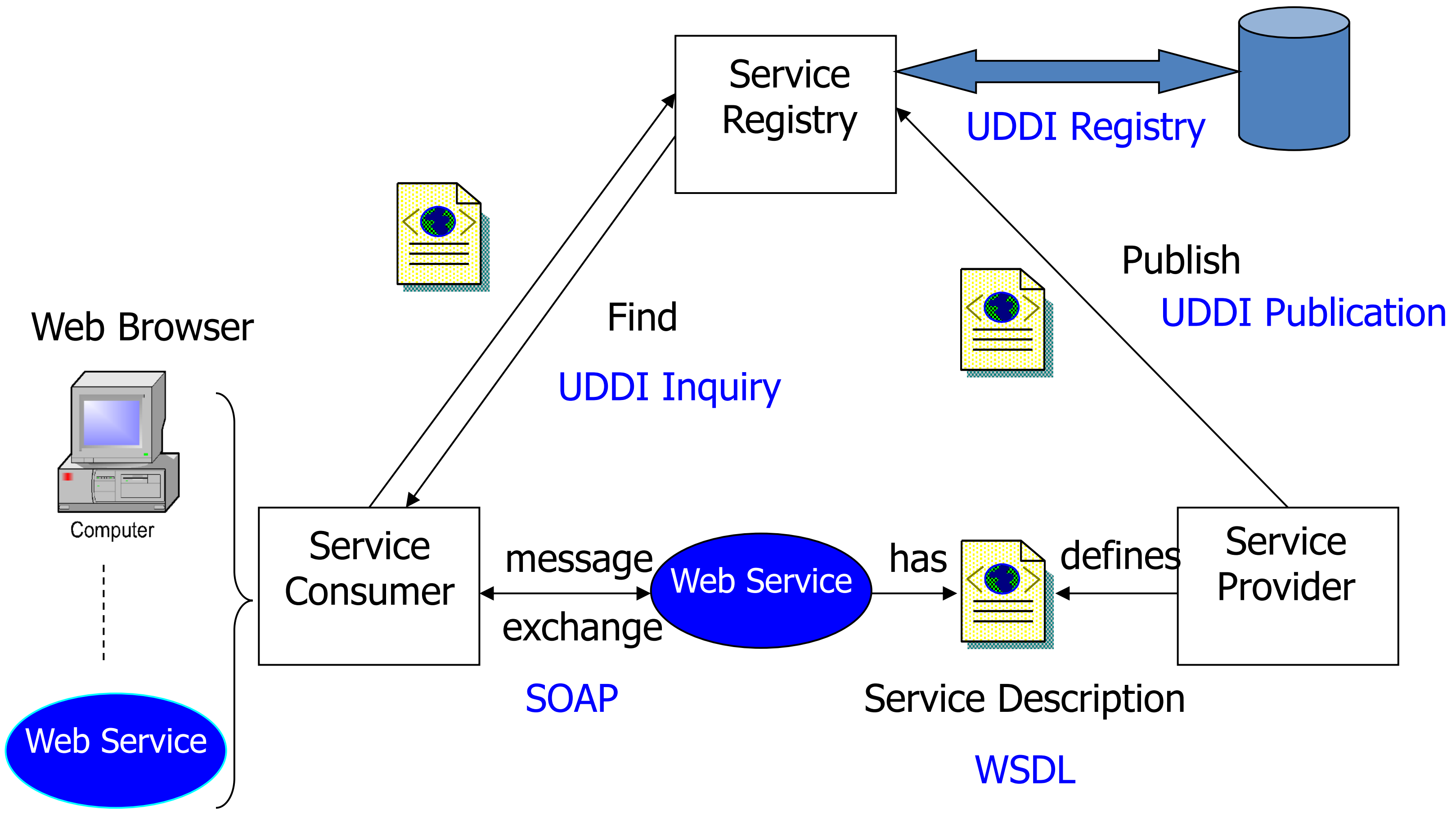
Basic Concepts/Standards/Technologies

SOA, XML, WSDL, SOAP, REST

https://github.com/eduardocoelholima/lecture-webapi-intro

Eduardo Lima

SOP Basis: Service-Oriented Architecture (SOA)

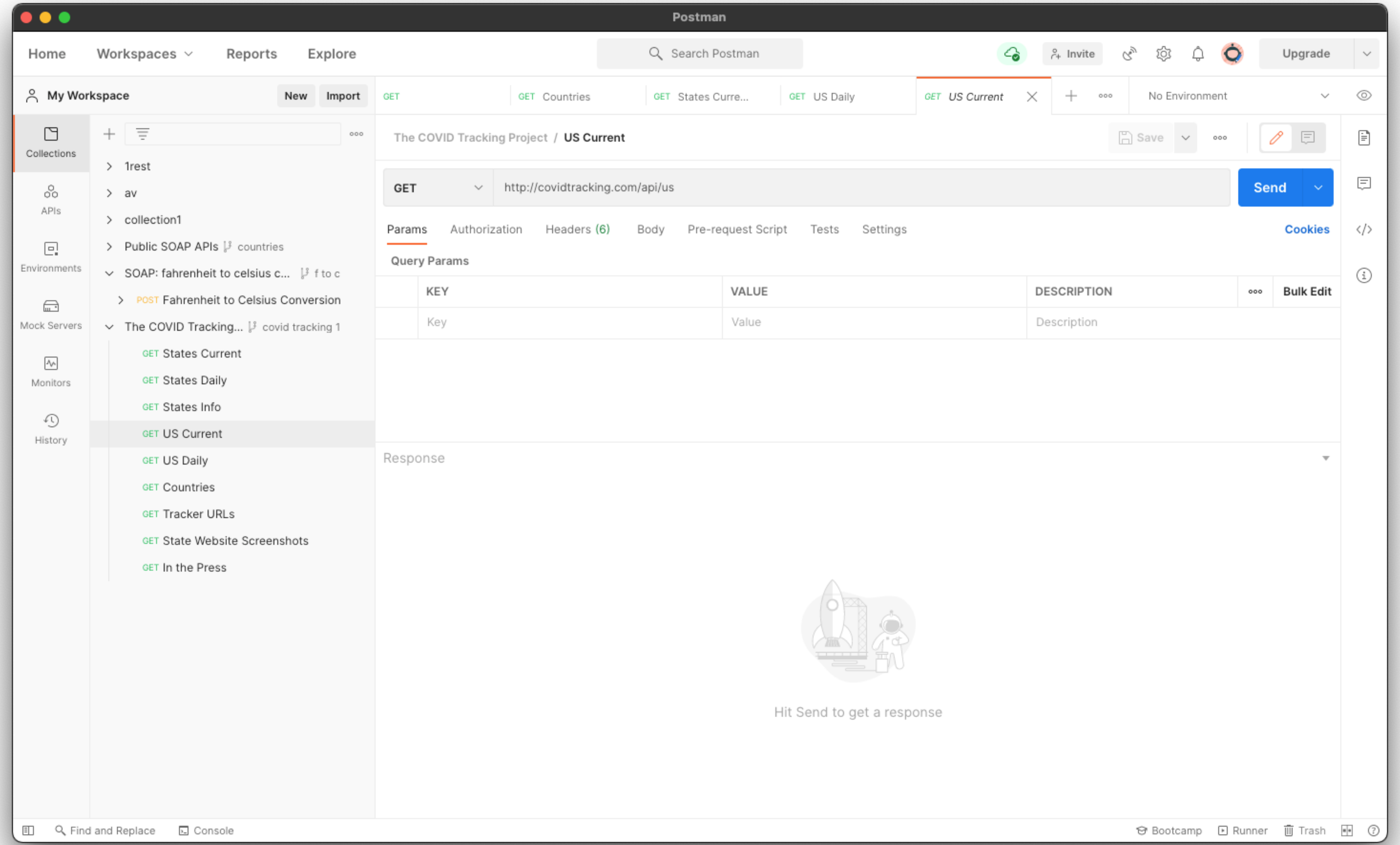


SOA: Roles of Interaction

- Web services provider
  - Owns Web service and implements business logic
  - Hosts and controls access to the service
    - Examples: Microsoft, Amazon, Facebook, …
- Web services requestor
  - Requires the certain functions to be satisfied
  - Looks for and invokes the service
    - Examples: a client, a server, or another web service
- Web services registry
  - Searchable directory where service descriptions can be published and searched
    - Examples: UDDI registry

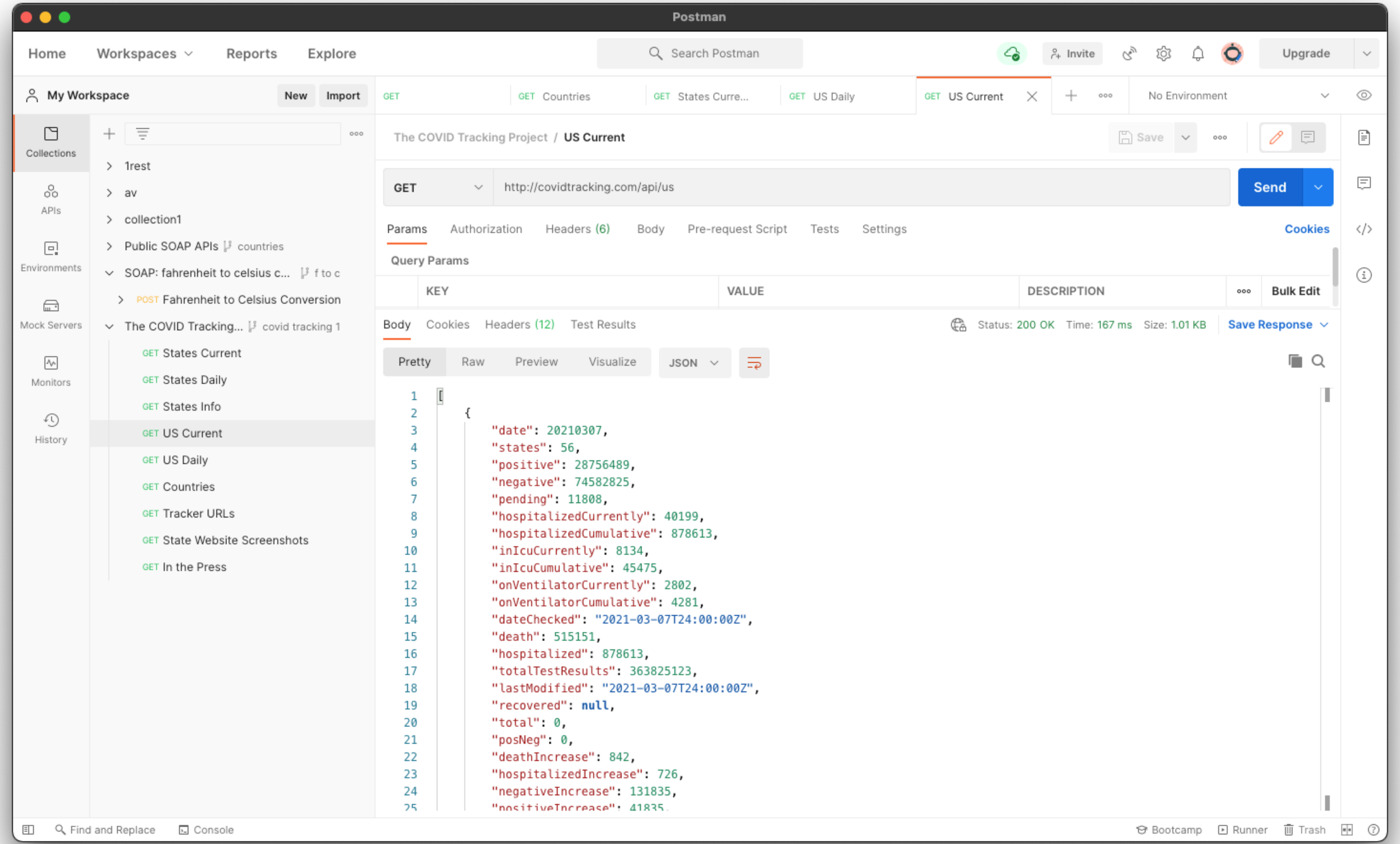
DEMO

- Postman consuming a SOAP web service



GET http://covidtracking.com/api/us

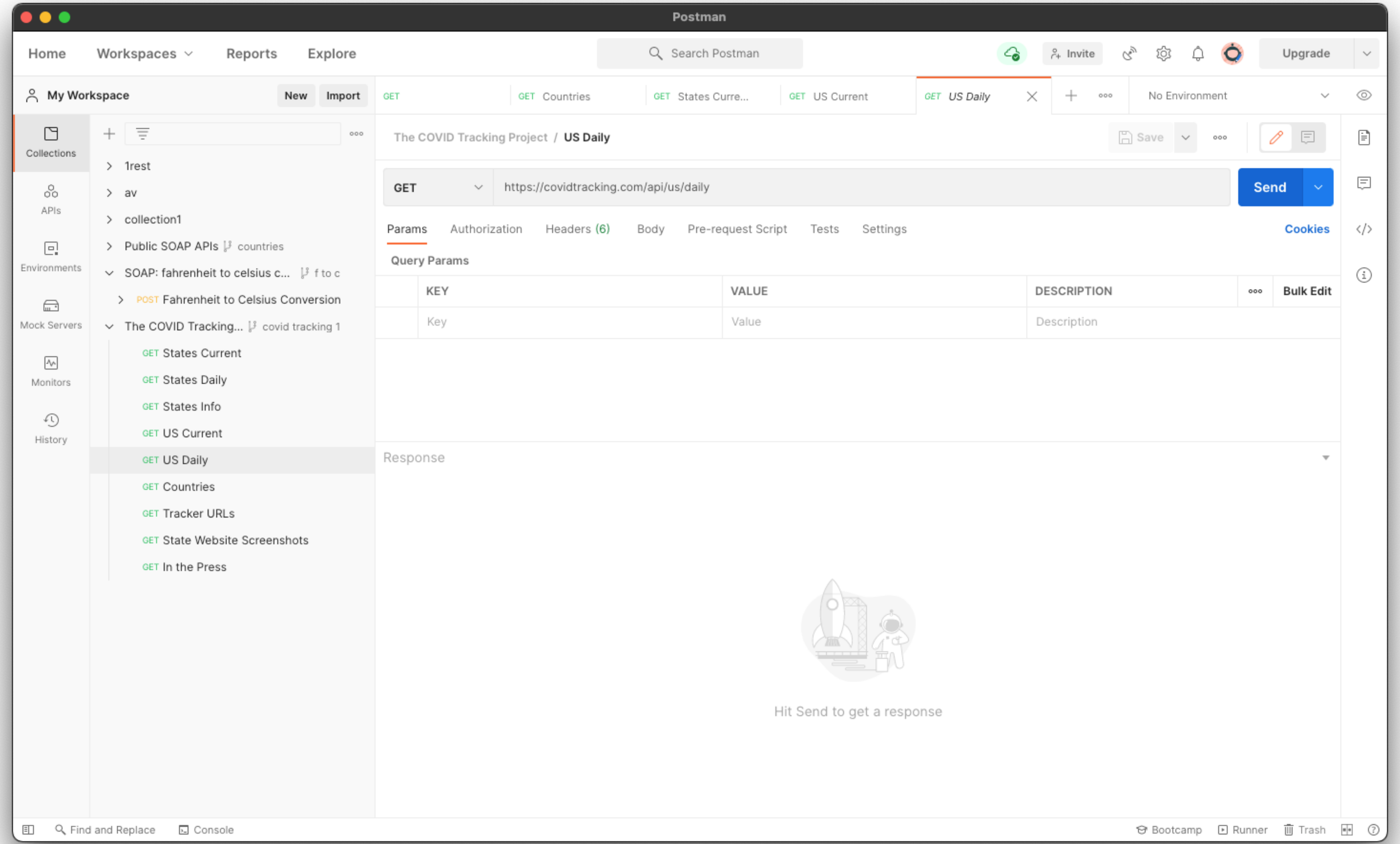
KEY	VALUE	DESCRIPTION
Key	Value	Description



GET http://covidtracking.com/api/us

Status: 200 OK Time: 167 ms Size: 1.01 KB

```
[
    {
        "date": 20210307,
        "states": 56,
        "positive": 28756489,
        "negative": 74582825,
        "pending": 11808,
        "hospitalizedCurrently": 40199,
        "hospitalizedCumulative": 878613,
        "inIcuCurrently": 8134,
        "inIcuCumulative": 45475,
        "onVentilatorCurrently": 2802,
        "onVentilatorCumulative": 4281,
        "dateChecked": "2021-03-07T24:00:00Z",
        "death": 515151,
        "hospitalized": 878613,
        "totalTestResults": 363825123,
        "lastModified": "2021-03-07T24:00:00Z",
        "recovered": null,
        "total": 0,
        "posNeg": 0,
        "deathIncrease": 842,
        "hospitalizedIncrease": 726,
        "negativeIncrease": 131835,
        "positiveIncrease": 41835,
```



The COVID Tracking Project / US Daily

GET https://covidtracking.com/api/us/daily

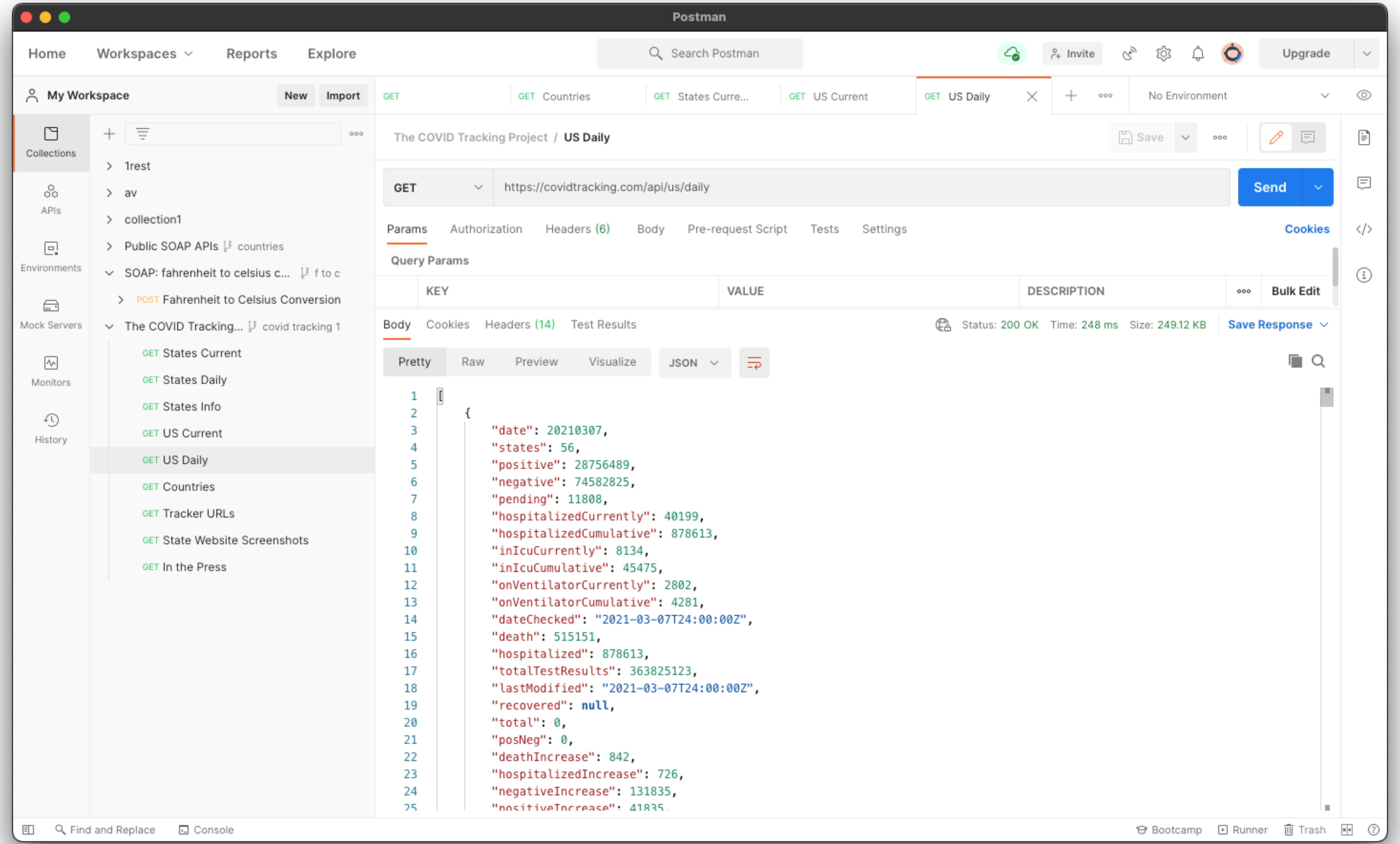
Params | Authorization | Headers (6) | Body | Pre-request Script | Tests | Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Response

Hit Send to get a response

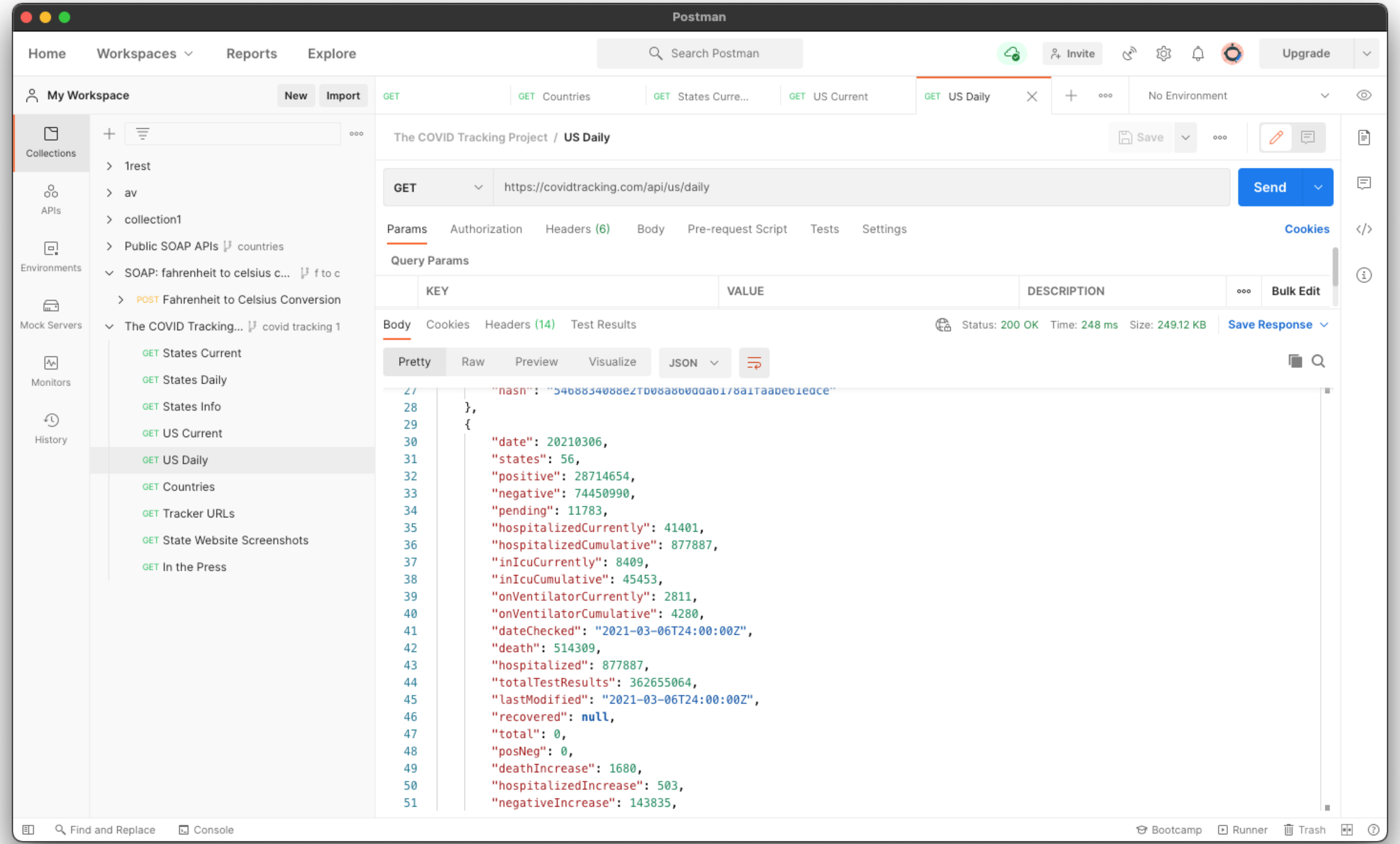


The COVID Tracking Project / US Daily

GET https://covidtracking.com/api/us/daily

Status: 200 OK Time: 248 ms Size: 249.12 KB

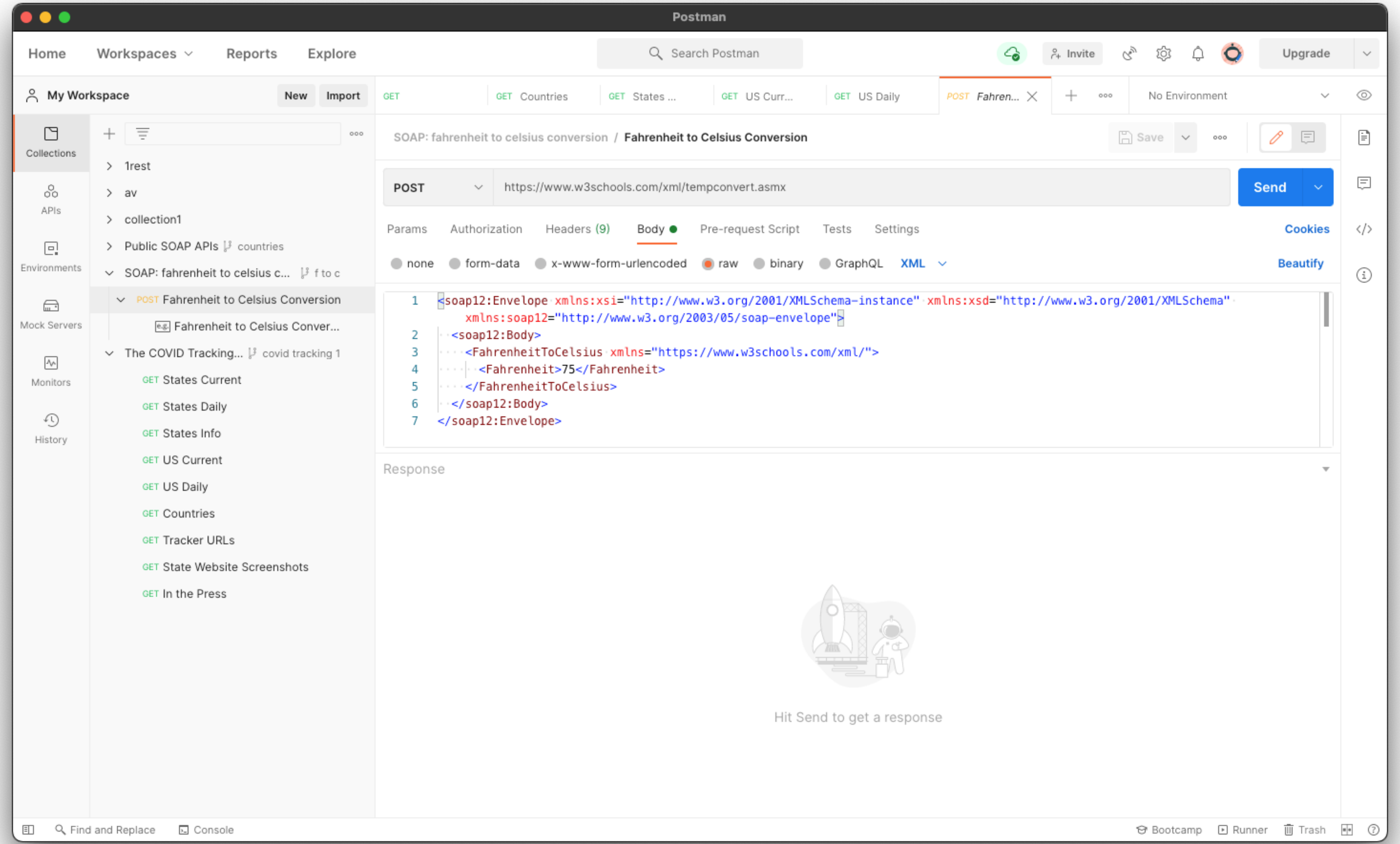
```
[
    {
        "date": 20210307,
        "states": 56,
        "positive": 28756489,
        "negative": 74582825,
        "pending": 11808,
        "hospitalizedCurrently": 40199,
        "hospitalizedCumulative": 878613,
        "inIcuCurrently": 8134,
        "inIcuCumulative": 45475,
        "onVentilatorCurrently": 2802,
        "onVentilatorCumulative": 4281,
        "dateChecked": "2021-03-07T24:00:00Z",
        "death": 515151,
        "hospitalized": 878613,
        "totalTestResults": 363825123,
        "lastModified": "2021-03-07T24:00:00Z",
        "recovered": null,
        "total": 0,
        "posNeg": 0,
        "deathIncrease": 842,
        "hospitalizedIncrease": 726,
        "negativeIncrease": 131835,
        "positiveIncrease": 41835,
```



GET https://covidtracking.com/api/us/daily

Status: 200 OK Time: 248 ms Size: 249.12 KB

```
        "hash": "5468834088e2fb08a860dda6178a1faabe61edce"
    },
    {
        "date": 20210306,
        "states": 56,
        "positive": 28714654,
        "negative": 74450990,
        "pending": 11783,
        "hospitalizedCurrently": 41401,
        "hospitalizedCumulative": 877887,
        "inIcuCurrently": 8409,
        "inIcuCumulative": 45453,
        "onVentilatorCurrently": 2811,
        "onVentilatorCumulative": 4280,
        "dateChecked": "2021-03-06T24:00:00Z",
        "death": 514309,
        "hospitalized": 877887,
        "totalTestResults": 362655064,
        "lastModified": "2021-03-06T24:00:00Z",
        "recovered": null,
        "total": 0,
        "posNeg": 0,
        "deathIncrease": 1680,
        "hospitalizedIncrease": 503,
        "negativeIncrease": 143835,
```

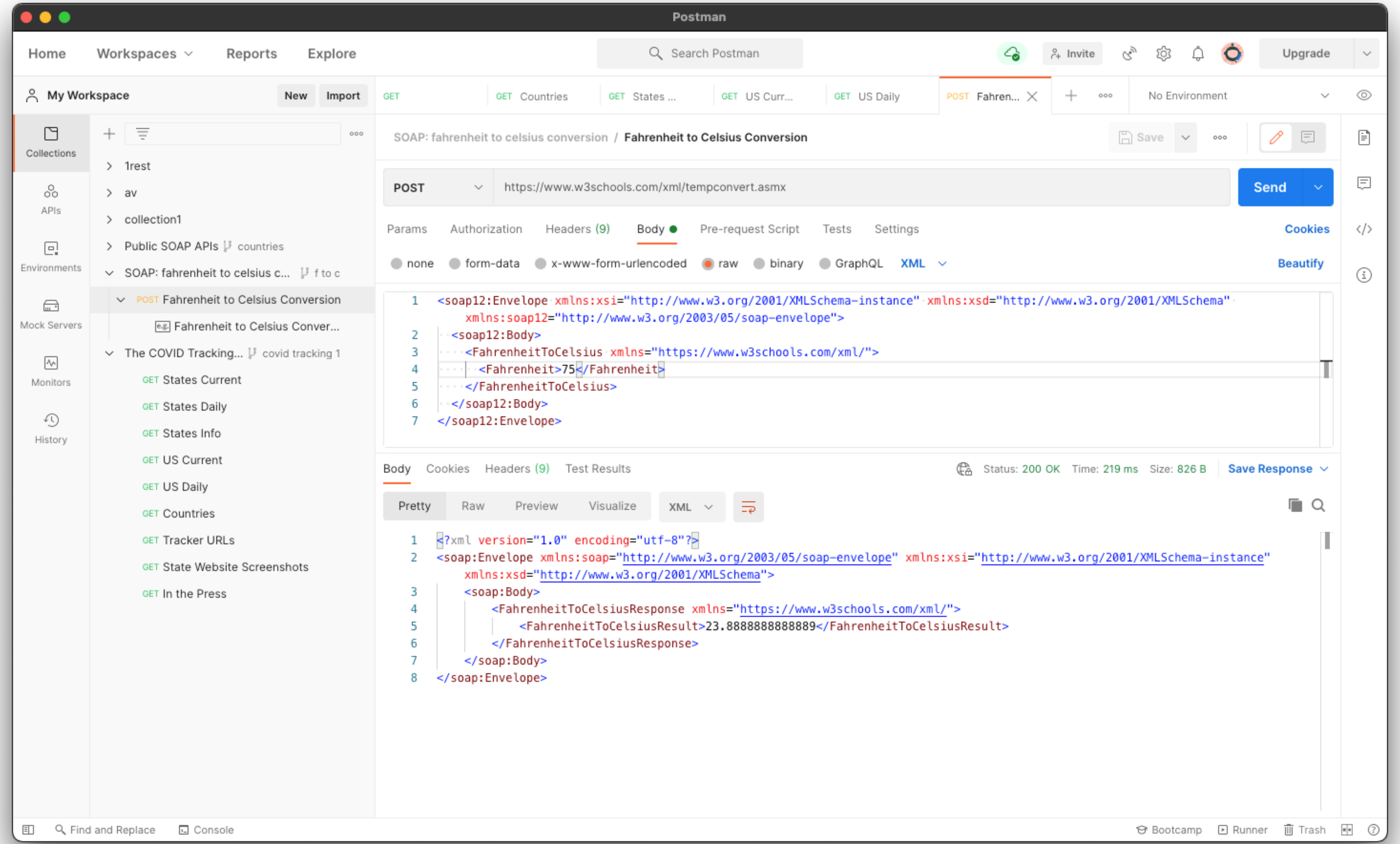


SOAP: fahrenheit to celsius conversion / Fahrenheit to Celsius Conversion

POST https://www.w3schools.com/xml/tempconvert.asmx

Body: none, form-data, x-www-form-urlencoded, [✓ raw], binary, GraphQL — XML

```
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <FahrenheitToCelsius xmlns="https://www.w3schools.com/xml/">
      <Fahrenheit>75</Fahrenheit>
    </FahrenheitToCelsius>
  </soap12:Body>
</soap12:Envelope>
```



POST https://www.w3schools.com/xml/tempconvert.asmx

```
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <FahrenheitToCelsius xmlns="https://www.w3schools.com/xml/">
      <Fahrenheit>75</Fahrenheit>
    </FahrenheitToCelsius>
  </soap12:Body>
</soap12:Envelope>
```

Status: 200 OK Time: 219 ms Size: 826 B

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <soap:Body>
        <FahrenheitToCelsiusResponse xmlns="https://www.w3schools.com/xml/">
            <FahrenheitToCelsiusResult>23.8888888888889</FahrenheitToCelsiusResult>
        </FahrenheitToCelsiusResponse>
    </soap:Body>
</soap:Envelope>
```

XML

- XML: eXtensible Markup Language
- Universal format for structured documents and data on the Web
- Common data format of Web services
- Supports semi-structured data model

Example

```
<book price = “95” currency = “USD”>
    <title> Programming Language Pragmatics</title>
    <author> Michael Scott </author>
    <publisher> Morgan Kaufmann </publisher>
    <edition> 3rd </edition>
              …
    <year> 2009 </year>
</book>
```

XML: Key Concepts

- Document
- Elements
- Attributes, e.g. Text
- Others
  - Namespace declarations, comments, processing instructions, …

Elements

- Enclosed in tags:
  - Book, title, author, …
  - Start tag: <book>   End tag: </book>
- Empty element   <red></red> OR <red/>
- Elements are ordered, may be repeated or nested

Basic XML Tag Syntax

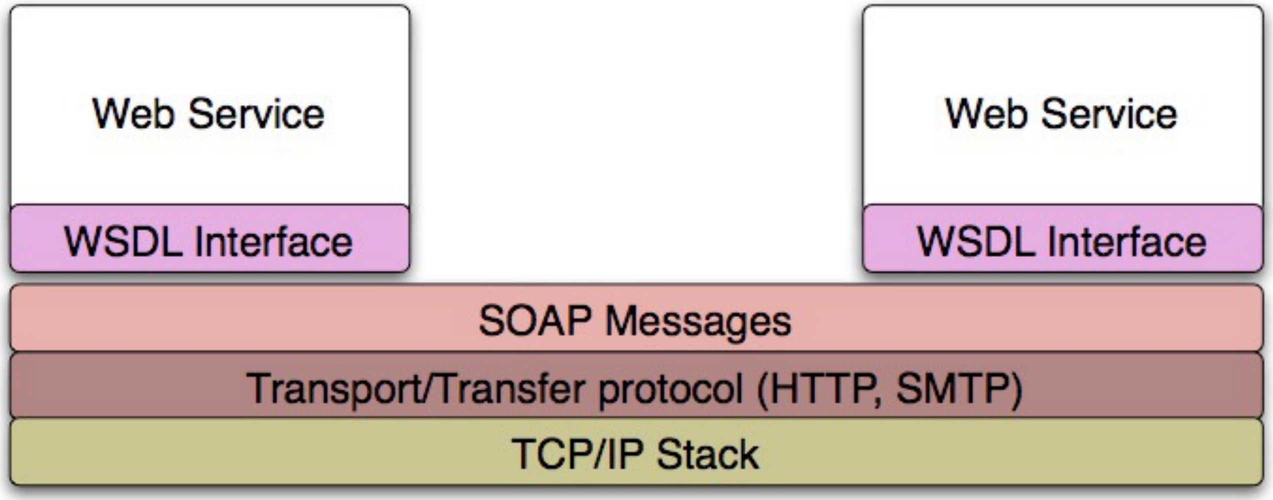
- Tags written as with HTML, but …
  - Case-sensitive names
  - Always need end tags
  - Special empty-element
  - Always quote attribute values
- Some other constraints for tags
  - Start with a letter or underscore
  - After first character, numbers, -, and . are allowed
  - Cannot contain white-spaces

Attributes

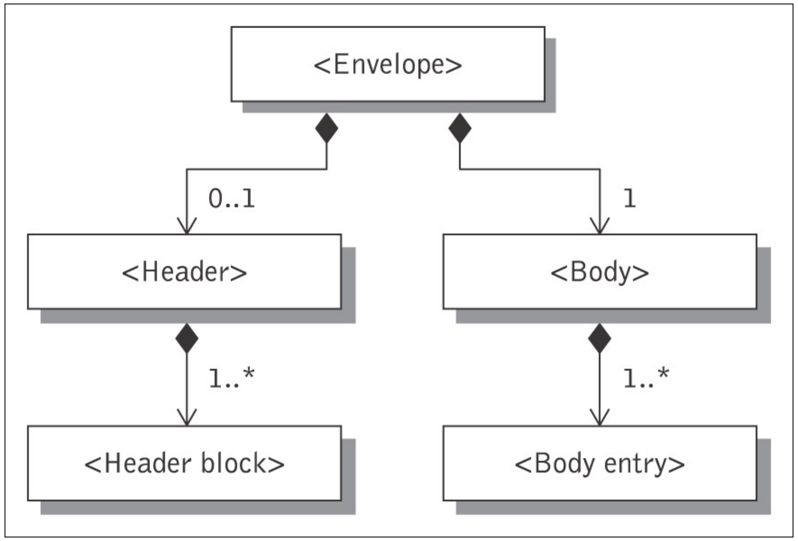
- Associated to Elements, …
  - <book price="20">
- Attributes
  - Unordered
  - Names must be unique
  - Cannot be nested
  - Provide metadata for element
  - Value enclosed in “ ”
- Multiple attributes separated by spaces
- Same naming conventions as elements

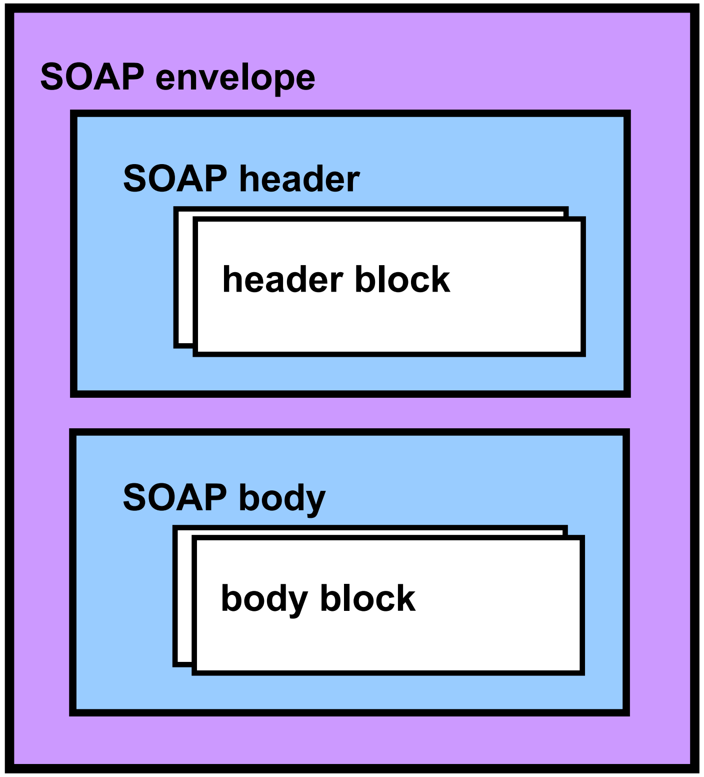
Simple Object Access Protocol

- Standard messaging protocol used by web services
- Supports inter-application communication



SOAP Message





- SOAP messages are seen as enveloper where the application encloses the data to be sent
- Consists of an <Envelope> element containing an optional <Header> and a mandatory <Body> element

SOAP Request (Example)

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV=“http://schemas.xmlsoap.org/soap/envelope/”
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/”>
  <SOAP-ENV:Header>
    <t:transId xmlns:t=“http://a.com/trans”>345</t:transId>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <m:Add xmlns:m=“http://a.com/Calculator”>
      <n1>3</n1>
      <n2>4</n2>
    </m:Add>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Response (Example)

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV=“http://schemas.xmlsoap.org/soap/envelope/”
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/”>
  <SOAP-ENV:Header>
    <t:transId xmlns:t=“http://a.com/trans”>345</t:transId>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <m:AddResponse xmlns:m=“http://a.com/Calculator”>
      <result>7</result>
    </m:AddResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

A More Involved Example

- Sample SOAP Request and Response Message for Google's Web Service Interface
  - http://www.w3.org/2004/06/03-google-soap-wsdl.html

For illustration only

WSDL

- WSDL: Web Service Description Language
- Pronounced “Whiz Dull”
- XML-based
- Why we need WSDL for web services?
  - Web services are designed to support machine-to-machine interaction
  - No human in the loop
  - Needs a specified and self-explanatory programming interface

Contents of a WSDL File (1)

- WSDL describes a service’s functionality
  - A service interface
    - Operations that can be invoked by service users
  - For each operation
    - Input parameters whose values are provided by service users, such as zipcode, address, …
    - Output parameters whose value will be returned to service users, such as directions, map image, …
- By parsing a WSDL file, a program can …
  - Determine if service is suitable, how to format the request, and how to handle the response

Contents of a WSDL File (2)

- Describes how to bind a service
  - Messaging style
  - Formatting (encoding) style
  - Transport protocol such as http, smtp, soap
- Describes where to locate a web service
  - A set of ports
    - A port defines the location of a web service, e.g., network address location or URL
- By parsing a WSDL file, a program can:
  - Locate and bind a web service

WSDL Document Content

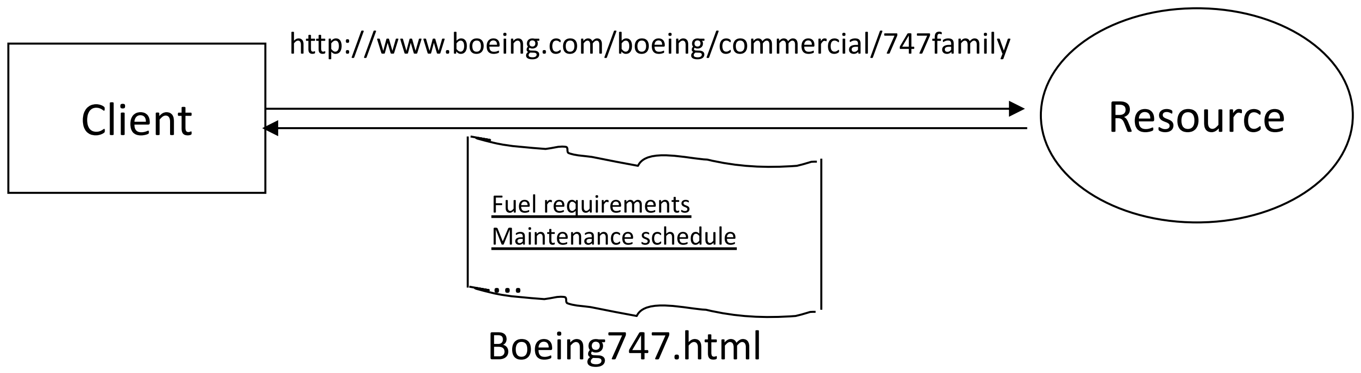
- Abstract (interface) definitions
  - <types> data type definitions
  - <message> operation parameters
  - <operation> abstract description of service actions
  - <portType> set of operation definitions
- Concrete (implementation) definitions
  - <binding> operation bindings
  - <port> association of endpoint with a binding
  - <service>location/address for each binding
- Example
  - http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl

DEMO

- Consuming a SOAP service
  - Python Zeep library
  - Netbeans 8 with Java EJB
- Publishing a SOAP service
  - Python Spyne library
  - Netbeans 8 with Java EJB

REST (Representational State Transfer)

- The Client references a Web resource using a URL
- A resource representation returned (an HTML document)
- Representation (e.g., Boeing747.html) puts client in new state
- When client selects hyperlink in Boeing747.html, it accesses another resource
- New representation places client into yet another state
- Client transfers state with each resource representation



Web Resources

- Information from database
  - invoice, resume, price, phone number,…
- Image
  - map, photo, ...
- Audio
  - song, speech, …
- Video
  - movie clip, …
- Others

Resource Representation

- Each resource is represented as a distinct Uniform Resource Identifier (URI)
  - Uniform Resource Name (URN)
    - e.g., isbn-10: 3642078885
  - Uniform Resource Locator (URL)
    - e.g., http://www.imdb.com/title/tt0068646/?ref_=fn_al_tt_1

REST Design Pattern

- Create a resource for every service
- Uniquely identify each resource with a logical URL
- Design your information to link to other information
  - That is, the information that a resource returns to a client should link to other information in a network of related information

REST Design Pattern (2)

- All interactions between a client and a web service are done with simple operations
- Most web interactions are done using HTTP and just four operations:
  - Retrieve information (HTTP GET)
  - Create information (HTTP PUT)
  - Update information (HTTP POST)
  - Delete information (HTTP DELETE)

An Example of RESTful Web Service

- Service: Get a list of parts
  - Web service makes an available URL to a parts list resource
  - A client uses this URL to get the parts list
    - http://www.parts-depot.com/parts
    - Note
      - How web service generates the parts list is completely transparent to the client
      - This is loose coupling

Data Returned: Parts List

- Each resource is identified as a URL
- Parts list has links to get each part’s detailed info
- Key feature of REST design pattern
  - Client transfers from one state to next by examining and choosing from alternative URLs in the response document

```
<?xml version="1.0"?>
<Parts>
    <Part id="00345" href="http://www.parts-depot.com/parts/00345"/>
    <Part id="00346" href="http://www.parts-depot.com/parts/00346"/>
    <Part id="00347" href="http://www.parts-depot.com/parts/00347"/>
    <Part id="00348" href="http://www.parts-depot.com/parts/00348"/>
</Parts>
```

Second Web Service

- Get detailed information about a particular part
  - Web service makes available a URL to each part resource
    - For example, here's how a client requests a specific part:
      - http://www.parts-depot.com/parts/00345
    - Data returned

```
<?xml version="1.0"?>
<Part>
    <Part-ID>00345</Part-ID>
    <Name>Widget-A</Name>
    <Description>This part is used within the frap assembly</Description>
    <Specification href="http://www.parts-depot.com/parts/00345/specification"/>
    <UnitCost currency="USD">0.10</UnitCost>
    <Quantity>10</Quantity>
</Part>
```

Web Service Examples

- Weather service
  - http://vhost3.cs.rit.edu/weather/Service.svc
- IMDB service
  - http://vhost3.cs.rit.edu/IMDB/Service.svc
- Calculator service
  - http://vhost3.cs.rit.edu/Calculator/Service.svc
- Test the services via the following link
  - http://vhost3.cs.rit.edu/Application/
- Some source code and sample services
  - http://vhost3.cs.rit.edu/CentralRepository/index.aspx

Response Formats of RESTful Web Services

- XML: eExtensible Markup Language
  - Universal format for structured documents and data on the Web
  - Common data format of Web services
- JSON: Javascript Object Notation
  - Derived from the JavaScript scripting language
  - Used for serializing and transmitting structured data

XML-Formatted Response Example

```xml
<root response="True">
<Movie Title="Titanic" Year="1997" imdbID="tt0120338" Type="movie"/>
<Movie Title="Titanic II" Year="2010" imdbID="tt1640571" Type="movie"/>
<Movie Title="Titanic: The Legend Goes On..." Year="2000" imdbID="tt0330994"
Type="movie"/>
<Movie Title="Titanic" Year="1953" imdbID="tt0046435" Type="movie"/>
<Movie Title="Titanic" Year="1996" imdbID="tt0115392" Type="movie"/>
<Movie Title="Raise the Titanic" Year="1980" imdbID="tt0081400" Type="movie"/>
<Movie Title="Titanic" Year="2012" imdbID="tt1869152" Type="series"/>
<Movie Title="The Chambermaid on the Titanic" Year="1997" imdbID="tt0129923"
Type="movie"/>
<Movie Title="Titanic: Blood and Steel" Year="2012" imdbID="tt1695366"
Type="series"/>
<Movie Title="Titanic" Year="1943" imdbID="tt0036443" Type="movie"/>
</root>
```

(http://www.omdbapi.com/?s=titanic&r=xml)

Json-Formatted Response Example

- {"Search":[{"Title":"Titanic","Year":"1997","imdbID":"tt0120338","Type":"movie"},{"Title":"Titanic II","Year":"2010","imdbID":"tt1640571","Type":"movie"},{"Title":"Titanic: The Legend Goes On...","Year":"2000","imdbID":"tt0330994","Type":"movie"},{"Title":"Titanic","Year":"1953","imdbID":"tt0046435","Type":"movie"},{"Title":"Titanic","Year":"1996","imdbID":"tt0115392","Type":"movie"},{"Title":"Raise the Titanic","Year":"1980","imdbID":"tt0081400","Type":"movie"},{"Title":"Titanic","Year":"2012","imdbID":"tt1869152","Type":"series"},{"Title":"The Chambermaid on the Titanic","Year":"1997","imdbID":"tt0129923","Type":"movie"},{"Title":"Titanic: Blood and Steel","Year":"2012","imdbID":"tt1695366","Type":"series"},{"Title":"Titanic","Year":"1943","imdbID":"tt0036443","Type":"movie"}]}

- (http://www.omdbapi.com/?s=titanic)

DEMO

- Consuming a REST service
  - CURL Command line
  - Postman
- Publishing a REST service
  - Python Spyne library