Web Services

Jatin Rohilla & Ruman Saleem

What is an API ?

(Application Programming Interface)

What is a Web Service?

- A web service is a unit of managed code that can be remotely invoked using HTTP. That is, it can be activated using HTTP requests.
- Web services allow you to expose the functionality of your existing code over the network.
- Once it is exposed on the network, other applications can use the functionality of your program.

API vs Web Services

- A Web service is merely an API wrapped in HTTP.
- An API doesn’t always need to be web based. An API consists of a complete set of rules and specifications for a software program to follow in order to facilitate interaction.
- All Web services are APIs but all APIs are not Web services.

Web Service Implementations

There are several protocols / architectures for implementation of Web Services. We will be covering the following:

- RPC & gRPC
- SOAP
- REST
- GraphQL

SOAP

(Simpe Object Access Protocol)

Structure of a SOAP Message

```
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
    soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
    <soap:Header>
        ...
    </soap:Header>
    <soap:Body>
        ...
        <soap:Fault>
        ...
        </soap:Fault>
    </soap:Body>
</soap:Envelope>
```

SOAP Headers

```
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
    soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
    <soap:Header>
        <m:Trans xmlns:m="https://www.example.org/transaction/"
            soap:mustUnderstand="1">234
        </m:Trans>
    </soap:Header>
    ...
</soap:Envelope>
```

SOAP HTTP Request

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn
```

```
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
    soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
    <soap:Body xmlns:m="http://www.example.org/stock">
        <m:GetStockPrice>
            <m:StockName>IBM</m:StockName>
        </m:GetStockPrice>
    </soap:Body>
</soap:Envelope>
```

SOAP HTTP Response

HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

```
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
    soap:encodingStyle="http://www.w3.org/2003/05/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>
```

Cons of SOAP

- Exclusively based on XML, complex syntax.
- SOAP is based on the **contract**, so there is a tight coupling between client and server applications.
- SOAP is slow because payload is large for a simple string message, since it uses XML format.
- Anytime there is change in the server side contract, client stub classes need to be generated again.
- Can’t be tested easily in browser

REST

REpresentational **S**tate **T**ransfer

Features

- Introduced by *Roy Fielding* in 2000
- It is not a protocol. It is resource based architectural style.
- Very popular in modern applications.
- It has no “official” standards, and hence flexible.
- It is stateless client server model.

RESTful APIs

- A base URL (e.g. https://api.example.com/v2).
- URI - resource identifier (e.g. `/users/12`).
- HTTP Methods/Verbs (e.g. GET, POST, PUT, PATCH, DELETE).
- Data representation - Current representation, given through `Content-Type` Header (e.g. application/json, application/xml)
- A restful API follows the REST constraints.

Methods	**Operation performed on Server**
GET	Retrieve resource(s).
POST	Insert a new resource. Also used to update.
PUT / PATCH	Insert or update the resource, if exists.
DELETE	Delete the resource.
OPTIONS	List allowed operations on resource.
HEAD	Retrieve only response headers.

REST Examples

GET /users
Response: **200 OK**
List of all users

```
[
  {
    "id": 1,
    "name": "Leanne Graham",
    "email": "Sincere@april.biz"
  },
  ...
  {
    "id": 10,
    "name": "Clementine Bauch",
    "email": "Nathan@yesenia.net"
  }
]
```

GET /users/1
Response: **200 OK**
Details of specific user with id=1.

```
{
  "id": 1,
  "name": "Leanne Graham",
  "email": "Sincere@april.biz"
}
```

DELETE /users/1
Response: **200 OK**
Destroyed user with id=1.

```
{}
```

REST Examples

POST /users

```
{
  "name": "Chelsey Dietrich",
  "username": "Kamren",
  "email":
"Lucio_Hettinger@annie.ca"
}
```

Response: 201 Created

```
{
  "id": 11,
  "name": "Chelsey Dietrich",
  "username": "Kamren",
  "email":
"Lucio_Hettinger@annie.ca"
}
```

PUT /users

```
{
  "name": "Chelsey Dietrich",
  "username": "Kamren",
  "email": "chelsey@kamren.ca"
}
```

Response: 200 OK

```
{
  "id": 11,
  "name": "Chelsey Dietrich",
  "username": "Kamren",
  "email": "chelsey@kamren.ca"
}
```

REST Examples

GET /users/1/posts
Response: **200 OK**
List of all posts by user whose id=1

```
[
  {
    "userId": 1,
    "id": 1,
    "title": "sunt aut facere"
  },
  ...
  {
    "userId": 1,
    "id": 2,
    "title": "qui est esse",
  }
]
```

REST
DEMO

Cons of REST

1. Too many endpoints. The client has to remember all of them, or refer documentation again and again.
2. Multiple API calls to retrieve relational data, resulting in too many round trips.
3. The shape of the response is determined by the server.
4. Over-fetching and Under-fetching

GraphQL

Graph Query Language

What is GraphQL ?

- NOT a Database
- NOT a Library
- NOT Language Specific

- A Query Language for the API
- Spec for server to execute graphql queries
- Allows to fetch deeply nested associations in one trip

```
{
    QUERY{
        USER{
            NAME
            AGE
        }
    }
}
```

Operation Type.

Operation Endpoint

Required Fields

Types of Operations

1. Query - READ DATA
2. Mutation - CREATE, UPDATE, DELETE
3. Subscription - Better than polling (REST)

GraphQL
DEMO