SOAP vs REST

Web services

- Web services are responsible for **online machine-to-machine communication**.
- Computers use them to **communicate** with each other **over the internet**.
- it’s only the front-end interfaces of websites and applications that reside on end users’ devices.
- The related **data is stored on a remote server** and **transmitted to the client machine through APIs** that provide web services for third-party users.
- APIs can **use different architectures to transfer data from the server to the client**.

Web services

- For a long time, **SOAP was the go-to messaging protocol** that almost every web service used.
- to build lightweight web and mobile applications, the more flexible REST architecture quickly gained popularity.
- In 2018, most public web services provided REST APIs and transfer data in the compact and easy-to-use JSON data-interchange format.
- However, enterprise users still frequently choose SOAP for their web services.

SOAP and REST

SOAP and REST both allow you to create your own API.

An API receives requests and sends back responses through internet protocols such as HTTP, SMTP, and others.

Many popular websites provide public APIs for their users, for example, Google Maps has a public REST API that lets you customize Google Maps with your own content.

There are also many APIs that have been created by companies for internal use.

SOAP and REST

SOAP and REST are two API styles that approach the question of data transmission from a different point of view.

SOAP is a standardized protocol that sends messages using other protocols such as HTTP and SMTP.

The SOAP specifications are official web standards, maintained and developed by the World Wide Web Consortium (W3C). As opposed to SOAP, **REST is not a protocol but an architectural style**. The REST architecture lays down a set of guidelines you need to follow if you want to provide a RESTful web service, for example, stateless existence and the use of HTTP status codes.

As **SOAP is an official protocol**, it comes with **strict rules and advanced security features** such as built-in ACID compliance and authorization.

Higher complexity, it **requires more bandwidth** and **resources** which can lead to slower page load times.

REST was created to address the problems of SOAP. Therefore it has a more **flexible architecture**. It consists of only loose guidelines and lets developers implement the recommendations in their own way.

It allows different messaging formats, such as HTML, JSON, XML, and plain text, while **SOAP only allows XML**. REST is also a more lightweight architecture, so RESTful web services have a better performance. Because of that, it has become incredibly popular in the mobile era where even a few seconds matter a lot (both in page load time and revenue).

REST - Representational State Transfer.

It’s an **architectural style** that defines a set of recommendations **for designing loosely coupled applications** that use the HTTP protocol for data transmission.

REST doesn’t prescribe how to implement the principles at a lower level.

REST guidelines allow developers to implement the details according to their own needs.

RESTful web services

Uniform interface – Requests from different clients should look the same, for example, the **same resource shouldn’t have more than one URI**.

Client-server separation – The client and the server should act independently. They should **interact** with each other **only through requests and responses**.

Statelessness – There **shouldn’t be any server-side sessions**. Each request should contain all the information the server needs to know.

Cacheable resources – Server responses should contain information about whether the data they send is cacheable or not. **Cacheable resources should arrive with a version number** so that the client can avoid requesting the same data more than once.

Layered system – There might be **several layers of servers** between the client and the server that returns the response. This **shouldn’t affect either the request or the response**.

Code on demand [optional] – When it’s necessary, the **response can contain executable code** (e.g., JavaScript within an HTML response) that the client can execute.

REST

Examples all over the internet: especially since **all big social media sites provide REST APIs** so that developers can seamlessly integrate their apps with the platform.

These **public APIs also come with detailed documentation** where you can get all the information you need to pull data through the API.

Twitter has a number of public REST APIs that all serve different purposes, such as a Search API with which you can find historical tweets, a Direct Message API with which you can send personalized messages, and an Ad API with which you can programmatically manage your ad campaigns.

The **WordPress REST API** is another popular example for REST APIs.

It **provides endpoints for WordPress data types** so that you can interact remotely with the content of a WordPress site and achieve great things such as building mobile apps with WordPress.

REST is almost always better for web-based APIs, as it **makes data available as resources** (e.g. user) as **opposed to services** (e.g. getUser) which is how SOAP operates.

REST inherits HTTP operations, meaning you can make simple **API calls using the well-known HTTP verbs like GET, POST, PUT, and DELETE**.

REST and JSON

JSON stands for JavaScript Object Notation.

The **REST architecture allows API providers to deliver data in multiple formats such as plain text, HTML, XML, YAML, and JSON**, which is one of its most loved features.

REST is lightweight and human-readable JSON format has also quickly gained attraction, as it’s super suitable for quick and painless data exchange.

JSON is **easy-to-parse** and lightweight data-interchange format.

It can be **used with any programming language**, not just JavaScript.

Its syntax is a subset of the Standard ECMA-262 3rd Edition.

JSON files consist of collections of name/value pairs and ordered lists of values that are universal data structures used by most programming languages.

JSON can be easily integrated with any language.

XML and JSON

Example code from the API docs of Atlassian's Crowd Server:

XML Format:

```
<?xml version="1.0" encoding="UTF-8"?>
<authentication-context>
   <username>my_username</username>
   <password>my_password</password>
   <validation-factors>
     <validation-factor>
       <name>remote_address</name>
       <value>127.0.0.1</value>
     </validation-factor>
   </validation-factors>
</authentication-context>
```

JSON Format:

```
{
   "username" : "my_username",
   "password" : "my_password",
   "validation-factors" : {
      "validationFactors" : [
         {
            "name" : "remote_address",
            "value" : "127.0.0.1"
         }
      ]
   }
}
```

SOAP - Simple Object Access Protocol

It's a **messaging protocol for interchanging data** in a **decentralized and distributed environment**.

SOAP can **work with any application layer protocol**, such as HTTP, SMTP, TCP, or UDP.

It **returns data** to the receiver in **XML format**.

Security, authorization, and error-handling are built into the protocol and, unlike REST, **it doesn't assume direct point-to-point communication**. Therefore it performs well in a distributed enterprise environment.

SOAP - Standards for XML

XML file that consists of the following parts:

- **Envelope (required)** – This is the starting and ending tags of the message.
- **Header (optional)** – It contains the optional attributes of the message. It allows you to extend a SOAP message in a modular and decentralized way.
- **Body (required)** – It contains the XML data that the server transmits to the receiver.
- **Fault (optional)** – It carries information about errors occurring during processing the message.

SOAP message example

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
      <n:priority>1</n:priority>
      <n:expires>2001-06-22T14:00:00-05:00</n:expires>
    </n:alertcontrol>
  </env:Header>
  <env:Body>
    <m:alert xmlns:m="http://example.org/alert">
      <m:msg>Pick up Mary at school at 2pm</m:msg>
    </m:alert>
  </env:Body>
</env:Envelope>
```