SOAP vs REST vs JSON - a 2021 comparison | Raygun.coms

SOAP vs REST vs JSON are frequently mentioned acronyms when speaking about web services. While SOAP and REST are two leading approaches to transferring data over a network using API calls, JSON is a compact data format that RESTful web services can use. Deciding whether you should create a SOAP vs REST API is an essential question if you are planning to provide a web service. Each architectural style has its own use cases, benefits, and limitations. In this article, we'll look into both the SOAP protocol and the REST guidelines in detail and also see how JSON fits into the landscape.

Topics covered

What are web services?

Web services are responsible for online machine-to-machine communication. Computers use them to communicate with each other over the internet. In fact, 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, such as SOAP and REST, to transfer data from the server to the client.

SOAP is a standardized specification maintained by the World Wide Web Consortium that is used for exchanging structured information in the form of well-defined, secure messages. For a long time, SOAP was the go-to messaging protocol that almost every web service used.

As these days developers need to build lightweight web and mobile applications, the more flexible REST architecture quickly gained popularity. REST is not a standardized protocol but a set of loose guidelines that gives more freedom to developers and companies to decide how they want to transfer data over the network and structure their API calls.

These days, most public web services provide 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.

The main differences between SOAP and REST

SOAP and REST both allow you to create your own API. API stands for Application Programming Interface. It makes it possible to transfer data from an application to other applications. 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 are two API styles that approach the question of data transmission from a different point of view.

REST was created to address the problems of SOAP.

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 (Atomicity, Consistency, Isolation, Durability) compliance and authorization. Higher complexity 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 popular in the mobile era where even a few seconds matter a lot (both in page load time and revenue).

What does REST stand for?

REST stands for 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.

Instead, the REST guidelines allow developers to implement the details according to their own needs. Web services built following the REST architectural style are called RESTful web services.

To create a REST API, you need to follow six architectural constraints:

1. **Uniform interface** – Requests from different clients should look the same, for example, the same resource shouldn't have more than one URI.
2. **Client-server separation** – The client and the server should act independently. They should interact with each other only through requests and responses.
3. **Statelessness** – There shouldn't be any server-side sessions. Each request should contain all the information the server needs to know.
4. **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.
5. **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.
6. **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.

What's the main reason to use REST?

Nowadays, REST is the most popular choice of developers to build public APIs. You can find many 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.

For example, 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 of 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. According to Nordic APIs, 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. Besides, REST inherits HTTP operations, meaning you can make simple API calls using well-known HTTP verbs such as GET, POST, PUT, and DELETE.

Challenges/limitations in REST

As REST is a stateless architecture by definition (see the third architectural constraint), sessions can't be maintained in web services. So, each API call needs to be completely independent and include all the necessary data since it can't depend on previous calls. This can be an issue if your web service requires stateful operations consisting of a chain of messages that rely on each other for information.

Even though REST is a more flexible architecture, it's also less secure. There's no contract between the client and server like in the case of the SOAP protocol, so it's not recommended for web services where you need to transfer highly confidential data over the network. You'll also need to create and

maintain detailed documentation for your web service so that users of your REST API can understand the nature and implications of each API call.

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. Thanks to the increasing popularity of REST, the lightweight and human-readable JSON format has also quickly gained traction, as it's super suitable for quick data exchange.

JSON stands for JavaScript Object Notation. It's an easy-to-parse and lightweight data-interchange format. In spite of its name, JSON is completely language-agnostic, so 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. Therefore, JSON can be easily integrated with any language.

To see the difference between XML and JSON, here is an example code from the API docs of Atlassian's Crowd Server that allows you to request and accept data in both XML and JSON formats:

```
<?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>
```

This is how the above XML code looks like in JSON:

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

As you can see, JSON is a more lightweight and less verbose format, and it's easier to read and write as well. In most cases, it's ideal for data interchange over the internet. However, XML still has some advantages. For example, it allows you to place metadata within tags and also handles mixed content better—especially when mixed node arrays require detailed expressions.

What does SOAP stand for?

SOAP stands for 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 follows a formal and standardized approach that specifies how to encode XML files returned by the API. A SOAP message is, in fact, an ordinary XML file that consists of the following parts:

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

Here is how an ordinary SOAP message looks like. The example is from the W3C SOAP docs and it contains a SOAP envelope, a header block, and a body:

```
<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>
```

What's the main reason to use SOAP?

In the short- to medium-term future, SOAP will likely continue to be used for enterprise-level web services that require high security and complex transactions. APIs for financial services, payment gateways, CRM software, identity management, and telecommunication services are commonly used examples of SOAP.

One of the most well-known SOAP APIs is PayPal's public API that allows you to accept PayPal and credit card payments, add a PayPal button to your website, let users log in with PayPal, and perform other PayPal-related actions.

Legacy system support is another frequent argument for using SOAP. Popular web services that have been around for a while might have many users who still connect to their services through their SOAP API which was the market leader before REST gained popularity. Salesforce, for example, provides

both a SOAP and a REST API so that every developer can integrate Salesforce with their own platform in a way that suits them best.

SOAP can be an excellent solution in situations where you can't use REST.

Besides, SOAP can be an excellent solution in situations where you can't use REST. Although these days, most web service providers want to exchange stateless requests and responses, in some cases, you may need to process stateful operations. This happens in scenarios where you have to make a chain of operations act as one transaction—for instance, in the case of bank transfers.

Although SOAP APIs are stateless by default, SOAP does support stateful operations that can be implemented using the WS-* (Web Services) Specifications that are built on top of the core XML and SOAP standards.

Challenges/limitations in SOAP

As SOAP can only transfer messages as XML files, your SOAP API will be less performant, as XML is a verbose format compared to JSON. API calls made to your server will need more bandwidth and it will take more time to process the request and transfer the response back to the client.

In SOAP, the client-server communication depends on WSDL (Web Service Description Language) contracts, which implies tight coupling. Therefore, it's not recommended for loosely coupled applications as you can't opt out of using a contract between the client and the server.

SOAP also has a higher learning curve, it is harder to code, and can't be tested in the web browser (as opposed to REST). With SOAP, you need to generate contracts in WSDL, create client stubs, follow strict specifications, and more. This also means that as a programmer, you'll also have less freedom of choice.

SOAP vs. REST comparison table

Although REST is very popular these days, SOAP still has its place in the world of web services. To help you choose between them, here's a comparison table of SOAP and REST, that highlights the main differences between the two API styles:

	SOAP	**REST**
Meaning	Simple Object Access Protocol	Representational State Transfer
Design	**Standardized protocol with pre-defined rules to follow.**	**Architectural style with loose guidelines and recommendations.**
Approach	Function-driven (data available as services, e.g.: "getUser")	Data-driven (data available as resources, e.g. "user").
Statefulness	**Stateless by default, but it's possible to make a SOAP API stateful.**	**Stateless (no server-side sessions).**

	SOAP	REST
Caching	API calls cannot be cached.	API calls can be cached.
Security	**WS-Security with SSL support. Built-in ACID compliance.**	**Supports HTTPS and SSL.**
Performance	Requires more bandwidth and computing power.	Requires fewer resources.
Message format	**Only XML.**	**Plain text, HTML, XML, JSON, YAML, and others.**
Transfer protocol(s)	HTTP, SMTP, UDP, and others.	Only HTTP
Recommended for	**Enterprise apps, high-security apps, distributed environment, financial services, payment gateways, telecommunication services.**	**Public APIs for web services, mobile services, social networks.**
Advantages	High security, standardized, extensibility.	Scalability, better performance, browser-friendliness, flexibility.
Disadvantages	**Poorer performance, more complexity, less flexibility.**	**Less security, not suitable for distributed environments.**

How to decide on SOAP or REST

Both SOAP and REST web services are platform-independent, which means that the client and server machines can use different technologies and programming languages.

To decide whether to use SOAP vs REST for your web service, you'll need to take the following factors into consideration:

- **Coupling:** for loosely coupled applications, choose REST while for tightly coupled applications, go with SOAP. Most modern web and mobile applications provide loosely coupled web services.
- **State:** if you need to process stateful operations and have complex API calls in which subsequent messages rely on each other, opt for SOAP. However, if your operations are stateless, REST is almost always the better option, especially because it also allows you to store data in the cache of the user's browser (not suitable for high-security data).
- **Security:** if you need to comply with security law or you transfer highly sensitive data, go with SOAP.
- **Knowledge of your team:** as SOAP has a higher learning curve, you'll need developers who have the sufficient knowledge. In most places, it's easier to find developers who have experience with creating REST APIs.

Overall, if you don't have a reason to use SOAP, such as security or creating a tightly coupled enterprise application, REST will most likely be the better choice. It's not just easier to code, test, and maintain, but the data transfer also requires less bandwidth, so you can provide a faster web service.

There is a time for all approaches

As you can see, both SOAP and REST have their advantages and disadvantages. Even though these days REST is the most popular approach to developing web services, mainly because it’s ideal for loosely coupled applications and allows for fast development and data transfer via JSON messages, SOAP is still the better choice in some cases, especially when your API needs to ensure high security.