Choosing between SOAP and REST web services

When designing a web service, spend time considering both the end user and the service itself. For example, how many requests do you expect the service to handle on an average day? A RESTful web service–that is, a service that conforms to the REST architectural style–tends to use less bandwidth and memory; however, a service that uses the SOAP protocol may be better suited for an enterprise application.

For more information on the benefits of SOAP and REST, including why you may want to choose one architectural style over the other, continue reading below.

On this page:

SOAP (Simple Object Access Protocol)

SOAP, created in 1998, is an XML-based protocol that was built with the intention of supporting enterprise software systems. In the SOAP style, data is typically available via create, read, update and delete services (for example, `createUser`, `readUserData`, `deleteUser`).

Consider using a SOAP service when you care about any of the following benefits:

- **Independence**: Whereas RESTful services require the HTTP transport protocol, SOAP services allow for a range of protocols, such as HTTP, HTTPS, FTP, and even SMTP.
- **Reliability:** Because XML is required for sending and receiving messages, and because a Web Service Description Language (WSDL) document can be used to precisely define a SOAP service, you can expect a greater degree of reliability in how the service operates and how you send and receive data with the service.
- **Security**: Per its specification, SOAP services can implement security enhancements, such as WS-Security, which, if used, enhance security through message integrity, message confidentiality, and single message authentication.

REST (Representative State Transfer)

The REST architectural style, which emerged in the early 2000s, was largely built in response to one central idea: that any one server should be able to communicate with any other server. According to the principles of REST, data must be accessed and manipulated through resource identifiers (for example, `/users`, `/users/login`), and these resources must allow for a range of operations through HTTP verbs. The most common implementations of RESTful services make use of HTTP verbs like `GET`, `POST`, `PUT`, `PATCH`, and `DELETE` to perform basic create, read, update, and delete operations.

Consider using a RESTful service when you care about any of the following benefits:

- Versatility: RESTful services allow a number of formats for messaging, including HTML, JSON, and XML.
- Speed to market: When compared to a SOAP service, which tends to require a strict schema for messaging and may break if a client has not sufficiently prepared for an update, a RESTful service does not require any rigid definitions to operate and is therefore better suited for continuous development.
- Performance: RESTful services rely on HTTP, a more lightweight network protocol, and perform better over a network when compared to SOAP services. This is because SOAP services tend to contain significant overhead in network packets and, as such, must parse more data.
- Scalability: Per REST guidelines, state information should be transferred rather than stored. Because servers do not need to store any state information, RESTful services are easier to cache and also easier to deploy to new servers.

Case study: A web service for a town library

If you were to implement a web service for a town library, creating a service that allows librarians and other staff members to update the rental status of books that have been rented out by library members, you might consider exposing the following SOAP endpoints that connect to the library's management system.

- `createMember`: Creates new library members, allowing those members to rent books and other multimedia from the library.
- `updateMember`: Updates the account details for library members, including details like the member's home address or account standing.
- `removeMember`: Removes a library member's full account information.
- `createBook`: Creates new book records in the library's management system.
- `updateBook`: Updates the details for a book in the library's management system, including details like the amount of copies available.

- removeBook: Removes books that are no longer a part of the library's collection from the library's management system.

If you were to implement that same web service in the REST style, you might consider exposing the following resources. Take note of /member/{memberID} and /book/{bookID}: these resources support multiple operations on the library's data.

- /member: Creates new library members, allowing those members to rent books and other multimedia from the library.
- /member/{memberID}: Updates the account details for library members, including details like the member's home address or account standing *and* removes a library member's full account information *and* retrieves information on a specific member.
- /book: Creates new book records in the library's management system.
- /book/{bookID}: Updates the details for a book in the library's management system, including details like the amount of copies available *and* removes books that are no longer a part of the library's collection from the library's management system *and* retrieves information on a specific book.