API Architecture Types

* REST
* SOAP
* RPC
* Other

REST - (Representational State Transfer)

1. Client-Server Architecture Consisting of clients, servers and resources
2. Stateless Client-Server communication(I.E. No client side data is saved between each different request)
3. Cacheable data to eliminate the need for some client-server interactions.
4. A uniform interface between components so that information is transferred in a standardised form.
5. Layered System constraint
6. Code on demand(Optional)

Why REST?

1. Limited Resources and bandwidth - Use when you have limited resources in your domain
2. Statelessness - If you don’t need to know the state of information between multiple requests.
3. Caching - If your client need a resource multiple times it is good to cache that instead of using a lot of requests
4. Ease of coding - It is easier to start to implement

SOAP - (Simple Object Access Protocol)

1. Can take in any type of application layer protocol(I.E. HTTP, SMTP, TCP)
2. Returning messages must be returned as XML docs.
3. Typically Used with WSDL(Web Service Design Language)

Why SOAP?

1. Asynchronous processing and subsequent invocation - if there is a need for guaranteed levels of reliability and security then SOAP provides a lot of additional features, especially when concerned to security
2. A Formal means of communication - When you need specific and rigid communication between client and server
3. Stateful operations - If your architecture requires the retention of the state of the application between requests

RPC - (Remote Procedure Call)

1. Takes call from the client and sends them to the server executing procedures on the server side
2. Takes a call from the client, serializes it, sends it to the server, server deserializes it. The server then executes the requested procedure, serializes the response, sends it to the client, then the client deserializes that response and uses it.

Why RPC?

1. Command API - If you are working in a domain that uses a lot of different commands(Join, leave, send, etc.) you can use this type of architecture.
2. Customer-specific APIs for internal microservices - If you want high message rate and message performance in your internal service then something like RPC would be a good thing to have. RPC is able to optimize very well and is able to make it very efficient to send messages.

SOAP Advantages

* Language, platform, and transport independent( REST requires HTTP)
* Works well in distributed environments
* Standardized
* Provides significant pre-built extensibility with the use of WS(Web Service Design Language) standards
* Built in error handling
* Automation when used with certain language products

SOAP Challenges

* WSDL File - The WSDL file on the client side and the functions on the server side are tightly coupled. If you change a part of the Server side that is part of the client WSDL then you need to go into all of them to fix it.
* Document Size - The sizes of the requests for SOAP are usually larger and need more bandwidth in the domain.

REST Advantages

* Nothing expensive needed to interact with the web service
* Smaller learning curve
* Efficient(SOAP must use XML for messages, REST can use smaller formats)
* Fast
* More closely related to other web technologies in design philosophy

REST Challenges

* Lack of Security - REST lacks any security in its URLs. It is appropriate for public URL’s where security is not needed much. However, if you plan on sending confidential data between client and server then you should try to find a better architecture type
* Lack of State - If you need state in your application(Like in a commercial site) then you would have to have the state be maintained client side. This can lead to more encumbered client applications. So if you need to know the state between different API calls then you shouldn’t use REST

RPC Advantages

* Straightforward Interactions - It uses only GET to fetch information and POST for everything else
* Easy to add functions - Easy to create new endpoints into our API
* High performance - Lightweight payloads

RPC Challenges

* Tight Coupling to the underlying system - It is not easy to implement an abstraction level in an RPC API. Given that the system is very tightly coupled you are not able to add an abstraction layer to your API and that can lead to security concerns
* Low discoverability - It is difficult to find out what your API is doing based on your requests.
* Function explosion - It is very easy to create new functions and inject them into your API. This can lead to a lot of very similar functions in your API.

To be clear this is what I have found out after researching the different types of API’s. This does not include all API types and might not be completely correct as I am just going off of things I have found on the internet.

Links to Sites I looked at:

<https://www.altexsoft.com/blog/soap-vs-rest-vs-graphql-vs-rpc/>

<https://www.guru99.com/comparison-between-web-services.html>

<https://www.redhat.com/architect/apis-soap-rest-graphql-grpc>

<https://www.smashingmagazine.com/2016/09/understanding-rest-and-rpc-for-http-apis/>