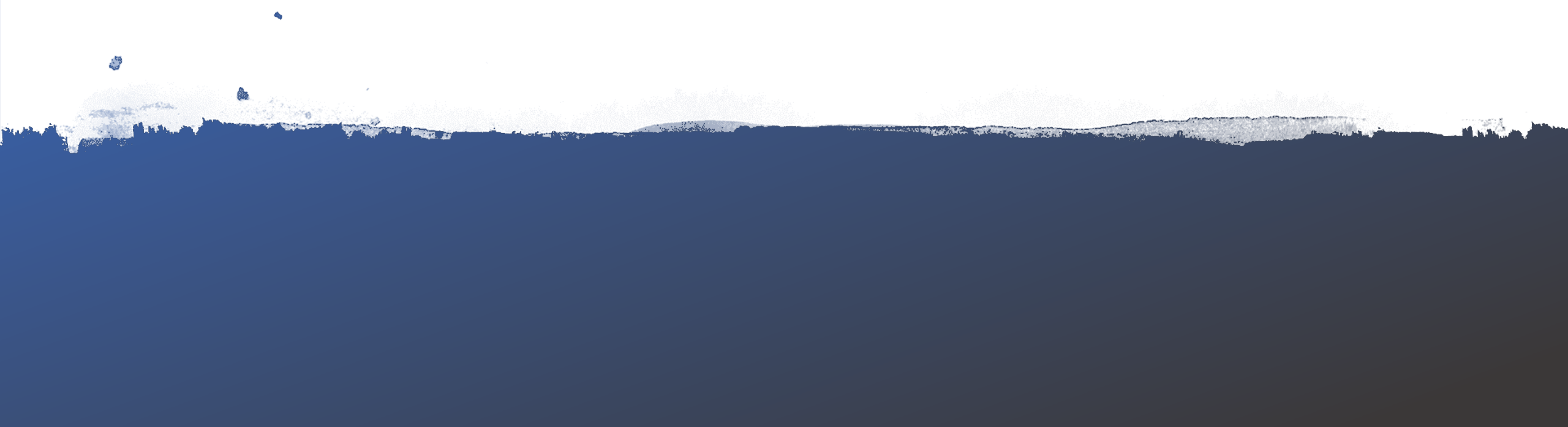
SoapUI and REST



USEFUL REFERENCES

SoapUI is an open-source web service testing application for service-oriented architectures and representational state transfers. Its functionality covers web service inspection, invoking, development, simulation and mocking, functional testing, load and compliance testing.

General Info
https://en.wikipedia.org/wiki/SoapUI

Introduction to SOAPUI
https://www.soapui.org/getting-started/introduction.html

Tool download
https://www.soapui.org/

What is SOAP and REST API?
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. **REST** allows different messaging formats, such as HTML, JSON, XML, and plain text, while **SOAP** only allows XML. **REST** is not a protocol but an architectural style. **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)

USEFUL REFERENCES

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:

- **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.

USEFUL REFERENCES

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 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. 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 the well-known HTTP verbs like GET, POST, PUT, and DELETE.

USEFUL REFERENCES

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:

- **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.

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 vs. REST

	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).**
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.**

MORE REFERENCES

API design guidelines – best practices for building a user-friendly API
https://raygun.com/blog/api-design-guidelines/

What is REST?
https://www.codecademy.com/articles/what-is-rest