**Content**

[1. Introduction into the world of APIs 2](#_Toc30414986)

[1. Terms 3](#_Toc30414987)

[2. What is an API 4](#_Toc30414988)

[1. Rapid API 4](#_Toc30414989)

[2. Mash-up 5](#_Toc30414990)

[3. API vs Web Service 5](#_Toc30414991)

[3. Protocols 6](#_Toc30414992)

[1. HTTP 6](#_Toc30414993)

[2. XML 6](#_Toc30414994)

[4. SOAP and REST 8](#_Toc30414995)

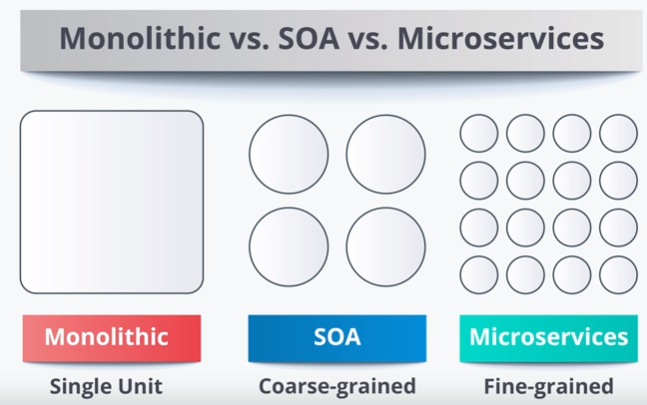
[5. REST 9](#_Toc30414996)

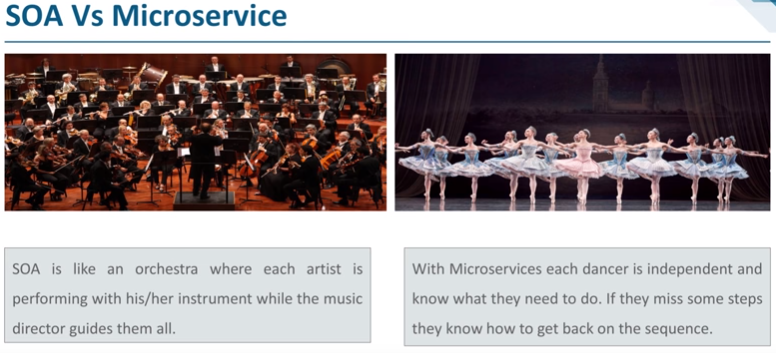
[6. OAuth 2.0 10](#_Toc30414997)

[1. Authorization Code Grant 12](#_Toc30414998)

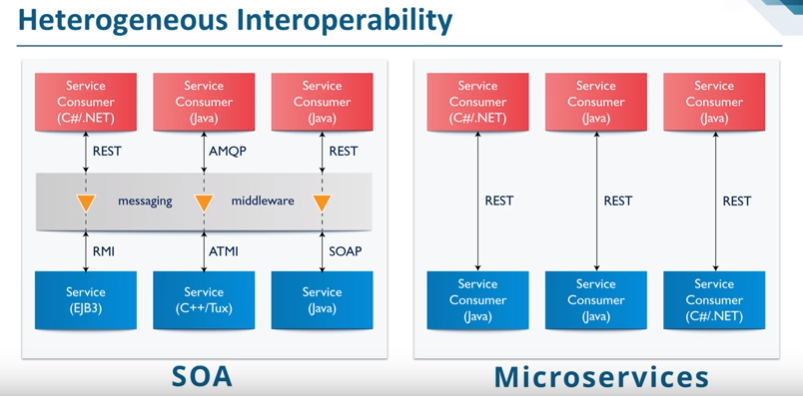
[2. OpenID Connect (OIDC) and PKCE (Proof Key for Code Exchange) 13](#_Toc30414999)

1. Introduction into the world of APIs





Microservices are decentralized.



In computing, the Java Remote Method Invocation (Java RMI) is a Java API that performs remote method invocation, the object-oriented equivalent of remote procedure calls (RPC), with support for direct transfer of serialized Java classes and distributed garbage-collection.

Asynchronous Task and Memory Interface, or ATMI, is a runtime framework and programming model for heterogeneous CPU-GPU systems.

* 1. Terms
* **Monolithic Applications**

A single executable single process in a single big …. There is 1 thing which has all the capabilities. Unreliable, 1 feature down, the whole system down. Inflexible. No place for updates. Complex.

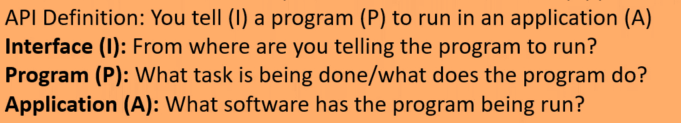
* **REST**
* **SOA**
* **Microservices**

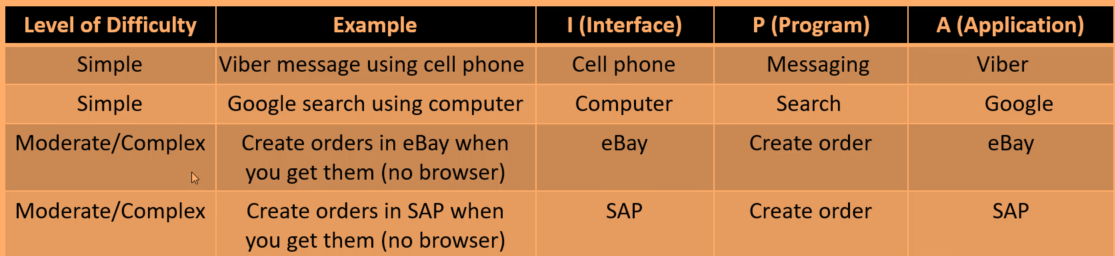
“maybe we can’t get a definition but we can identify common things” /Martin Fowler/

* **GraphQL**
* **API management tools**
* **OAuth**
* **PKCE**
* **OAS and Swagger**

1. What is an API



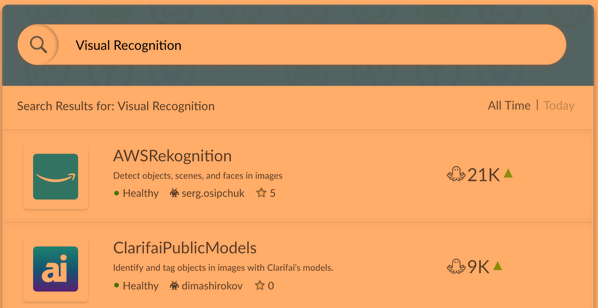




Tell a program to run that you don’t know.

* Just use the program, don’t write it
* Platform independent (you just tell it to run)
* Upgrade safe
  1. Rapid API

Rapid API is the largest marketplace for APIs, it is free to sign up. If you are a developer it’s a one-stop location of public APIs for you.



* 1. Mash-up

Combining different APIs. Google Flights / Travelocity – by searching for flight in the background it calls the APIs of different airlines and provides me the most beneficial. Not a commonly used word in the IT, but can be the future. APIs calls APIs and so on…

* 1. API vs Web Service

Web = internet

Service = API

Not all APIs use internet, but obviously there which are web services.

So, all web services are APIs, but not all APIs are web services. Not all APIs use the internet.

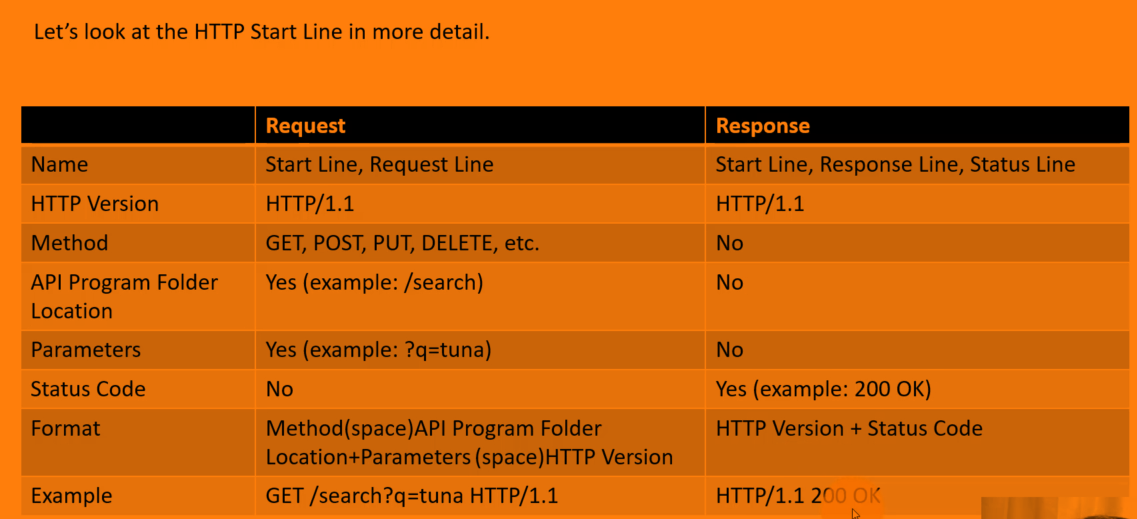
Web Services use XML or JSON format, you cannot use whatever format. REST, SOAP, or XML/RPC to transfer that data (protocols)

When people talk about APIs they usually talk about web services.

1. Protocols
   1. HTTP

Hypertext: What is regular text? [www.google.com](http://www.google.com) – what does regular text do? Nothing. HTTP makes it special, Hyper, goes somewhere else. HTTP allows him to go to a google computer, makes it hypertext.

1. Start Line
2. Headers – header fields (content-type, url, cache-control …. )
3. Blank Line
4. Body

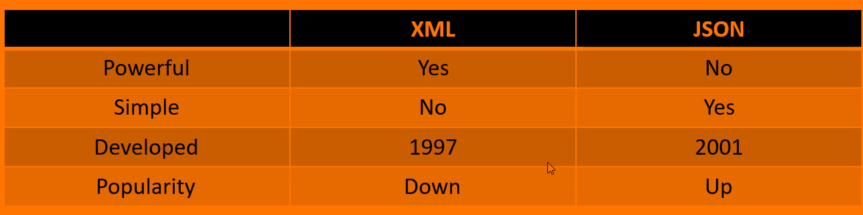


**Idempotence:** can do as many times as you want and result stays the same (safe repeat)



* 1. XML

eXtensible –you can customize them, sister of HTML, there are not customizable. Buth created by W3C organization.

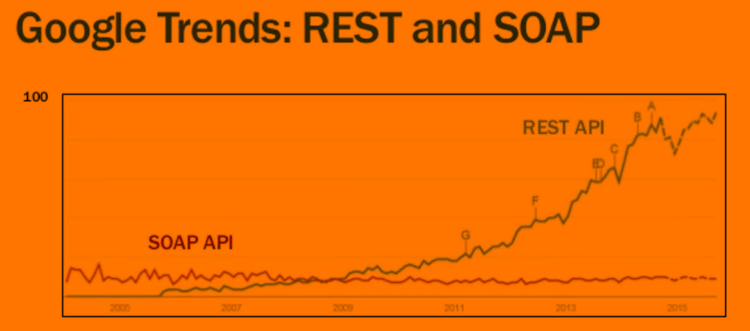
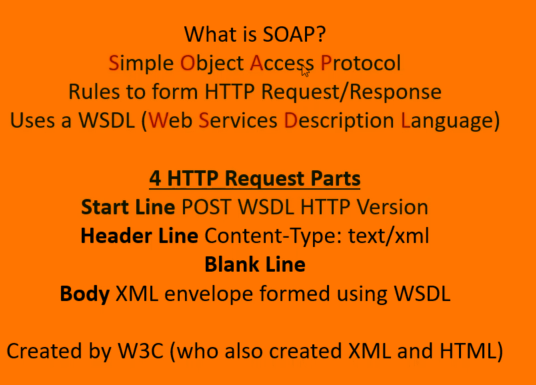


XML is more powerful, has the ability to transform XML , query, security, email schemas, capabilities.



1. SOAP and REST

* They are ways doing web services.
* Ways to form HTTP requests and responses
* REST is simpler



Simple - subjective

Object - API

Access

Protocol - rules, ways to do something, follow the rules, get it done.

WSDL – Web Services Description Language

* 1980 – SUN RPC (was not HTTP based)
* 1989 – HTTP
* 1998 – XML-RPC ( P, Procedure, API) – python mbe?
* 1999 – SOAP – better , always POST
* 2000 – REST (1 guy, not org) – in rest you have methods, stateless ?!

1. REST

* Cache
* Stateless

Stat is a point in time. REST is stateless it does not depend on state of the server which contains the information?!

Representational State Transfer – the response is a representation of state of record in google server, copy.

* Authentication – proving your identity (uname, pw)
* Authorization – limited access (others public and private photos



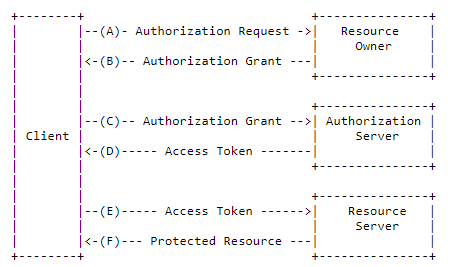
1. OAuth 2.0

RFC 6749 <https://tools.ietf.org/html/rfc6749>

Request for Comments (RFC), in information and communications technology, is a type of text document from the technology community. An RFC document may come from many bodies including from the Internet Engineering Task Force (IETF), the Internet Research Task Force (IRTF), the Internet Architecture Board (IAB), or from independent authors.The RFC system is supported by the Internet Society (ISOC).

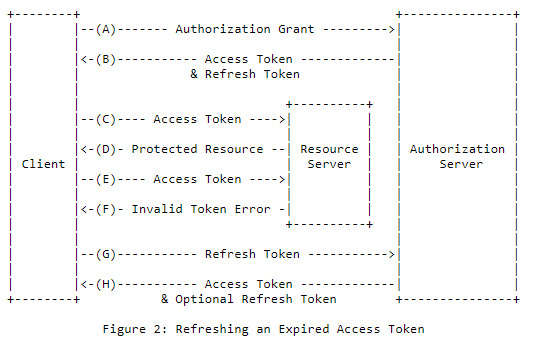
Allows 3rd party access, limited (authorized) to a web service (HTTP).

* Resource Owner – can be you
* Resource Server – which provides the photo
* Client / Application – 3rd party, that need access
* Authorization Server – Which provides the access token

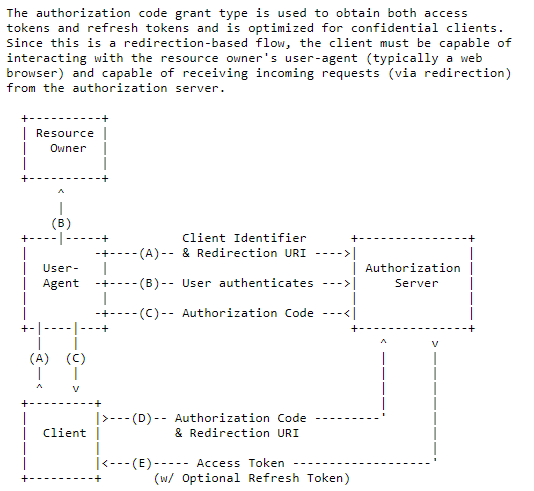


Authorization Grant Types:

* Authorization Code – resource owner needs to log in to the authorization server and say OK. Direct ask.
* Implicit – code
* Resource Owner – Full access ask for resources
* Client Credentials – access public
* Access token – to get a protected resource
* Refresh token – get a new access token, once expired.



* 1. Authorization Code Grant



Note: The lines illustrating steps (A), (B), and (C) are broken into

two parts as they pass through the user-agent. User-Agent = Browser.

The flow illustrated in Figure 3 includes the following steps:

(A) The client initiates the flow by directing the resource owner's

user-agent to the authorization endpoint. The client includes

its client identifier, requested scope, local state, and a

redirection URI to which the authorization server will send the

user-agent back once access is granted (or denied).

(B) The authorization server authenticates the resource owner (via

the user-agent) and establishes whether the resource owner

grants or denies the client's access request.

(C) Assuming the resource owner grants access, the authorization

server redirects the user-agent back to the client using the

redirection URI provided earlier (in the request or during

client registration). The redirection URI includes an

authorization code and any local state provided by the client

earlier.

(D) The client requests an access token from the authorization

server's token endpoint by including the authorization code

received in the previous step. When making the request, the

client authenticates with the authorization server. The client

includes the redirection URI used to obtain the authorization

code for verification.

(E) The authorization server authenticates the client, validates the

authorization code, and ensures that the redirection URI

received matches the URI used to redirect the client in

step (C). If valid, the authorization server responds back with

an access token and, optionally, a refresh token.

* 1. OpenID Connect (OIDC) and PKCE (Proof Key for Code Exchange)

So, how do you protect your SPA in such a hostile environment? When SPAs were new and browsers as well as providers were more limited in their capabilities, OAuth 2.0 and its sister standard, **OpenID Connect (OIDC)** offered an approach called the Implicit flow. This flow has always had problems inherent to it and these problems are exacerbated by the advanced capabilities focused on user experience in browsers. Today, **Proof Key for Code Exchange (PKCE)** provides a modern solution for protecting SPAs.

OIDC is a thin identity layer for authentication and Single Sign-On that rides on top of OAuth 2.0, an authorization framework. In this post, you’ll learn some foundational concepts of OIDC and OAuth2. You’ll be guided through a simple SPA example written in Vue.js that starts with the older (now deprecated) Implicit flow and then shows the more secure Authorization Code with PKCE flow.

1. Swagger and OAS (OpenAPI Specification)

OpenAPI Specification – the industry standard for RESTful API design.