

Fatec

Jales

Prof. José Camargo

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE
SISTEMAS
6º Semestre

Disciplina: TÓPICOS ESPECIAIS

ARQUITETURA ORIENTADA A SERVIÇOS - PARTE 01 –
FUNDAMENTOS TEÓRICOS

Prof. Me. Jorge Luís Gregório

www.jlgregorio.com.br



Agenda



- Contextualizando
- O que é *um web-service*?
- *Software as a Service* e sistemas orientados a serviço;
- Vantagens da abordagem orientada a serviços na Engenharia de Software;
- Open Banking e SOA
- Arquitetura Orientada a Serviços
- Manifesto SOA
- Princípios Orientadores
- Elementos SOA/SOAP
- Padrões SOA
- A abordagem REST
- REST x SOA
- Restrições REST
- Request & Response
- Parâmetros de Requisição
- HTTP Status Code
- Verbos HTTP
- Níveis de Maturidade REST
- HATEOAS
- Documentação
- Autenticação e JWT
- Versionamento

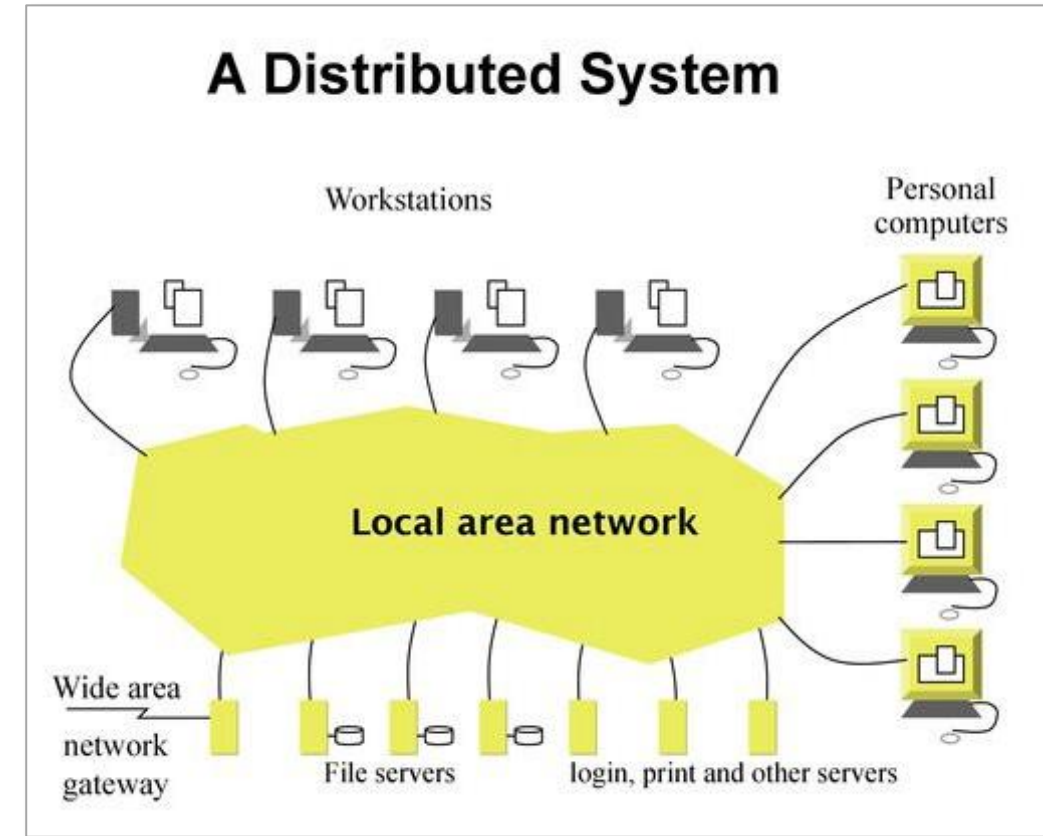


Contextualizando

- Até o final da década de 1990 a maioria dos sistemas eram monolíticos;
- Um Sistema monolítico é caracterizado pelo **forte acoplamento** entre seus componentes, o que dificulta sua **manutenabilidade**, **escalabilidade** e, principalmente, **integração** com outros sistemas concebidos, especialmente considerando diferentes tecnologias e linguagens de programação;
- Se apenas uma parte do sistema ficasse indisponível ou precisasse de manutenção, todo o sistema teria que ficar *offline* até que os problemas fossem resolvidos;
- A integração entre diferentes sistemas era demasiadamente complexa, pois envolvia ligações P2P, mapeamento de dados e outros conceitos caros e ineficientes em termos computacionais;

Contextualizando

- Conforme o uso das Tecnologias de Informação e Comunicação (TIC) foram avançando no ambiente de negócios, a integração entre diferentes sistemas e disponibilidade tornaram-se fatores cruciais considerando a qualidade dos serviços e a competitividade das organizações;
- Assim, foi necessário repensar as arquiteturas de sistemas utilizadas até então, iniciando com uma **abordagem distribuída**, em que um sistema é composto por partes geograficamente distribuídas;



Contextualizando...

- Mesmo a abordagem distribuída não resolveu completamente os problema, pois a integração entre diferentes sistemas de diferentes linguagens ou plataformas ainda era algo extremamente ineficiente;
- Surgiram muitos padrões diferentes (SOAP, Cordoba, RMI, etc) e implementados por diversos fornecedores, o que também trazia uma série de problemas “antigos”, como dificuldade de integração, manutenção, escalabilidades, etc.
- A ideia de um sistema usar a funcionalidade de outro era limitada a utilização de uma base de dados comum, RPC (*remote procedure call*) ou troca de informações via arquivos estruturados de maneira deficiente, burocrática e com pouca ou nenhuma semântica;
- Nos piores casos, sistemas inteiros eram reescritos na tecnologia A ou B objetivando integrar-se perfeitamente ao ecossistema da empresa que era provido por um único fornecedor.

Em meio ao caos, surge o *web-service!*

- *A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.*

W3C (2004)

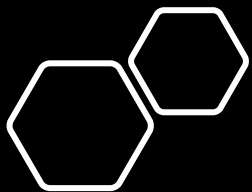
Usando serviços (*web-services*)

- Um serviço é uma unidade ou conjunto de funcionalidades de software independente, que foi desenvolvido para concluir uma tarefa específica, como recuperar determinadas informações ou executar uma operação. Ele contém as integrações de dados e o código necessários para executar uma função de negócios completa. Esses serviços podem ser acessados remotamente e é possível interagir com eles e atualizá-los de maneira independente
- Ao expor serviços usando protocolos de rede padrão, como SOAP, JSON, ActiveMQ ou Apache Thrift, para enviar solicitações ou acessar dados, a SOA facilita a vida dos desenvolvedores, pois eles **não precisam fazer a integração do zero**. Em vez disso, eles podem usar padrões chamados *enterprise service buses* (ESBs), que realizam a integração entre um componente central e sistemas de ***back-end*** e depois os tornam acessíveis como interfaces de serviços. Isso também **ajuda os desenvolvedores a reutilizar funções existentes, em vez de recriá-las.**

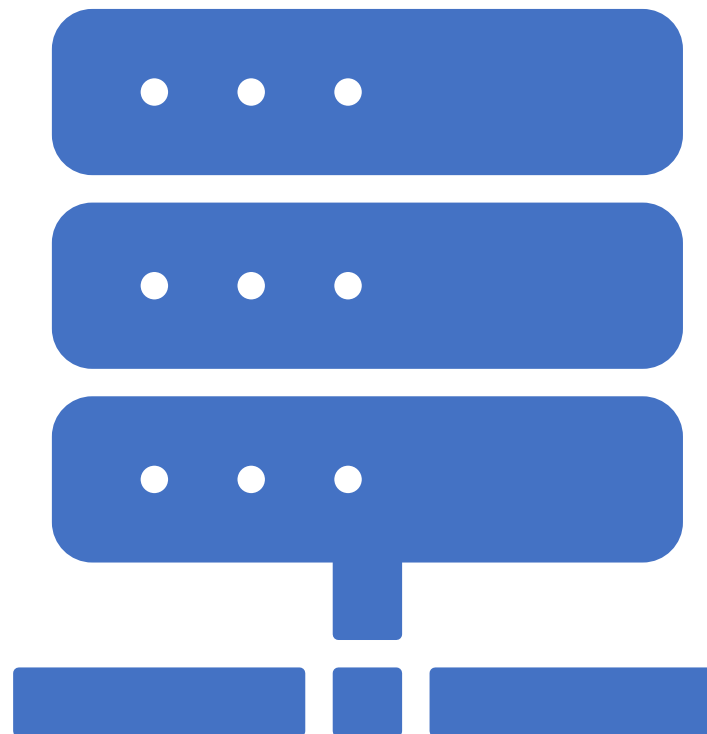
- No estilo de arquitetura orientada a serviços, a comunicação entre os serviços é realizada por um sistema de "**baixo acoplamento**". Essa é uma maneira de interconectar componentes, também chamados de "elementos", em um sistema ou rede para que transmitam informações ou coordenem processos empresariais e **reduzam as dependências entre eles**. Efetivamente, isso cria uma aplicação nova.

Vantagens em comparação com a abordagem monolítica

- **Time to market acelerado e maior flexibilidade:** a reutilização de serviços faz com que seja mais fácil e rápido montar aplicações. Os desenvolvedores não precisam sempre começar do zero, como no caso das aplicações monolíticas.
- **Uso de infraestrutura legada em novos mercados:** com a SOA, é mais fácil para os desenvolvedores escalar ou ampliar o uso de uma funcionalidade para plataformas ou ambientes novos.
- **Redução de custos resultante da maior agilidade e eficiência no desenvolvimento.**
- **Fácil manutenção:** como todos os serviços são autossuficientes e independentes, é possível modificá-los e atualizá-los conforme a necessidade, sem afetar os outros serviços.
- **Escalabilidade:** como a SOA permite executar serviços usando vários serviços, plataformas e linguagens de programação diferentes, há uma grande melhoria na escalabilidade. Além disso, a SOA usa um protocolo de comunicação padronizado, o que permite às empresas reduzir a interação entre clientes e serviços. Com a redução no nível de interação, é possível escalar aplicações com menos urgência e aborrecimentos.
- **Maior confiabilidade:** por ser mais fácil fazer o debug de serviços menores do que um grande código, a SOA permite criar aplicações mais confiáveis.
- **Maior disponibilidade:** os recursos da SOA estão disponíveis para todos.



O QUE É UM SERVIÇO?



O que é um serviço?

...um *webservice* é uma **representação padrão** de algum **recurso computacional e informacional** que pode ser **usado por outros programas**. Esses recursos podem ser de informação, como um catálogo de peças; recursos computacionais, como um processador especializado; ou recursos de armazenamento”

Sommerville (2018, p. 491-492)

Uma definição mais técnica

Um *webservice* é...

*Um componente de software **reusável**, **fracamente acoplado**, que encapsula **funcionalidade discreta**, e que pode ser **distribuído** e **acessado** por meio de programação. Um webservice é um serviço acessado usando protocolos padrões da internet e baseados em XML.*

Sommerville (2018, p. 492)

Software como um serviço e sistemas orientados a serviço

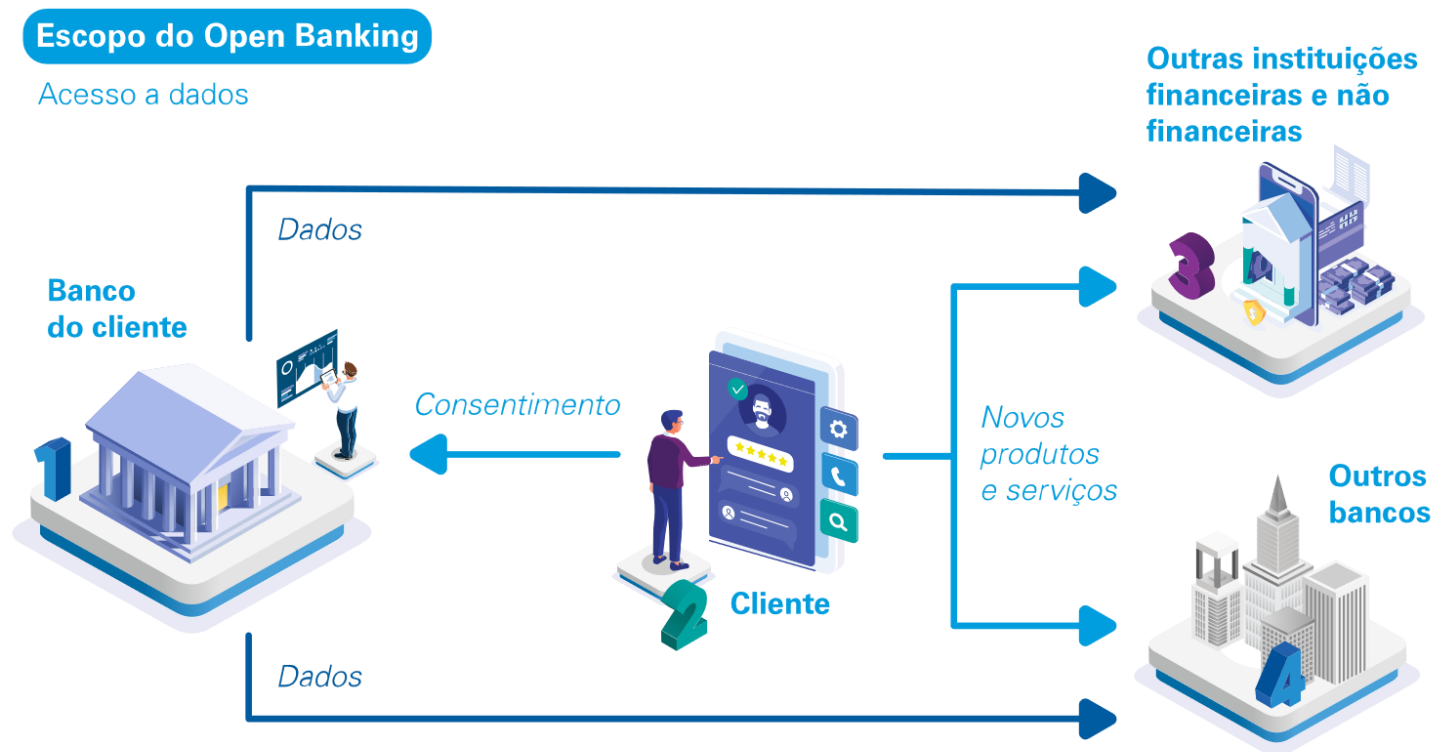
- Não são a mesma coisa!
- ***SaaS – Software as a Service:*** consiste em fornecer funcionalidades de software a usuários de maneira remota, por meio da web, e não por meio de aplicações.
- **Sistemas Orientados a Serviço:** são concebidos por meio de componentes de serviços reusáveis, que são acessados por outras aplicações e não diretamente pelos usuários.
- **Um SaaS pode ou não ser um Sistema orientado a serviços.**

Vantagens de uma abordagem orientada a serviços para a Engenharia de Software

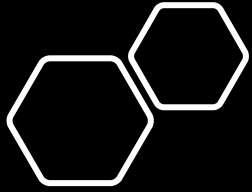
- **Possibilidade de criar aplicações mais simples:** considerando que grande parte das funcionalidades (simples ou complexas) já estão implementadas por serviços próprios ou de terceiros. Particularmente útil em sistemas mobile;
- **Maior capacidade evolutiva do sistema,** considerando mais funcionalidades, escalabilidade e mudanças nas regras de negócio ou legislação;
- **Maior diversidade de serviços:** mais funcionalidades ao cliente, agregando valor ao sistema;
- **Possibilidade de gerar renda:** é possível vender o serviço;
- **Capacidade de mudar de fornecedor rapidamente:** em casos em que um serviço fica indisponível; relação custo-benefício; etc.
- **Exposição de funcionalidades:** permitir acesso público ou autorizado a serviços de software – particularmente útil quando se deseja que parceiros e clients acessem informações e/ou serviços da organização.

Open Banking

Segundo o Banco Central, **Open Banking** é “...o compartilhamento padronizado de dados, produtos e serviços por meio de abertura e integração de sistemas, com o uso de interface dedicada para essa finalidade, por instituições financeiras, instituições de pagamento e demais instituições autorizadas a funcionar pelo BC, de forma segura, ágil e conveniente”.



Fonte: <https://home.kpmg/br/pt/home/industrias/servicos-financeiros/open-banking.html>



Arquitetura Orientada a Serviços



Arquitetura Orientada a Serviços

- **Arquitetura Orientada a Serviços** (SOA, do inglês *service-oriented architecture*) é um estilo de arquitetura que se baseia na ideia de que serviços executáveis podem ser incluídos em aplicações. Os serviços têm interfaces publicadas bem definidas, e as aplicações podem escolher se elas são adequadas ou não.

Sommerville (2018, p. 495)

Manifesto SOA

Orientação a Serviço é um paradigma que molda o que você faz.
Arquitetura Orientada a Serviço (SOA) é um tipo de arquitetura
que resulta da aplicação de orientação a serviço.

Nós temos aplicado orientação para ajudar organizações a,
de maneira consistente e sustentável,
agregar valor ao negócio, com maior agilidade
e efetividade de custos, em alinhamento com a dinâmica das necessidades de negócio.

Através de nosso trabalho, priorizamos:

Valor do negócio em relação a estratégia técnica;

Objetivos estratégicos em relação a benefícios específicos de projetos;

Interoperabilidade intrínseca em relação a integração personalizada;

Serviços compartilhados em relação a implementações de propósito específico;

Flexibilidade em relação a otimização; e

Refinamento evolutivo em relação a busca da perfeição inicial.

Isso é, mesmo valorizando os itens à direita, valorizamos mais os itens à esquerda.

Princípios orientadores

- Respeitar a estrutura social e de poder da organização.
- Reconhecer que SOA, em última instância, requer mudanças em múltiplos níveis.
- O escopo da adoção de SOA pode variar.
Manter os esforços gerenciáveis e dentro de limites significativos.
- Produtos e padrões, por si só, não proverão uma SOA nem aplicarão os paradigmas de orientação a serviço por você.
- SOA pode ser realizada através de uma variedade de tecnologias e padrões.
- Estabelecer um conjunto uniforme de padrões e políticas corporativas
embasado em padrões da indústria, de facto, e da comunidade.
- Buscar uniformidade no exterior e permitir diversidade no interior.
- Identificar serviços através da colaboração entre partes interessadas no negócio e na tecnologia.
- Maximizar o uso de serviços considerando o escopo de utilização atual e futuro.
- Verificar que os serviços satisfaçam os requisitos e objetivos de negócio.
- Evoluir os serviços e sua organização em resposta ao uso real.
- Separar os diferentes aspectos de um sistema que mudam com diferentes frequências.
- Reduzir dependências implícitas e publicar todas as dependências externas para aumentar a robustez e diminuir o impacto de mudanças.
- A cada nível de abstração, organizar cada serviço em torno de uma unidade de funcionalidade coesa e gerenciável.

Elementos SOA

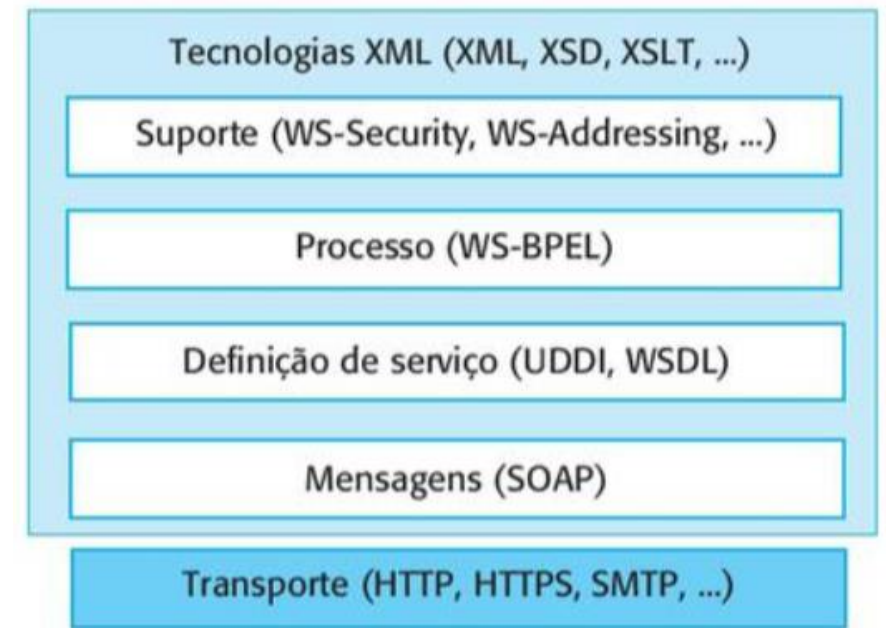
- Fornecedor do serviço;
- Solicitante do serviço;
- Registro ou descoberta de serviços;
- O serviço em si.



Fonte: Sommerville (2018, p. 495)

Padrões SOA

- O desenvolvimento de padrões de serviços é fundamental para permitir que os serviços estejam de acordo com o manifesto SOA;
- Os protocolos de *web-services* contemplam todos os aspectos computacionais de troca de informações, incluindo linguagens de programação;
- Os padrões baseiam-se em XML (*eXtensible Markup Language*), que é uma notação legível por homens e por máquinas, o que facilita e padroniza toda a gama de conceitos e recursos SOA.



Fonte: Sommerville (2018, p. 495)

Principais padrões para SOA

- **SOAP:** é um padrão de troca de mensagens que apoia a comunicação entre serviços. O protocolo simples de acesso a objetos (do inglês *Simple Object Access Protocol*) define os componentes essenciais e opcionais das mensagens passadas entre os serviços. Os serviços em uma arquitetura orientada a serviços às vezes são chamados de serviços baseados em SOAP.
- **WSDL:** linguagem de descrição de web services (WSDL, do inglês *web-service description language*) é um padrão para definição de interfaces de serviços. Ela estabelece como operações do serviço (nomes, parâmetros e tipos de operação) e como as ligações dos serviços devem ser definidas.
- **WS-BPEL:** padrão para uma linguagem de fluxo de trabalho que é usada para definir os programas de processo que envolvem vários serviços diferentes. Do inglês, *Web Services Business Process Execution Language*.

Descoberta de Serviços

- UDDI (*Universal Description, Discovery and Integration* - Descrição, Descoberta e Integração Universais): define uma maneira de publicar e descobrir informações sobre serviços da Web.
 - O UDDI tem duas funções:
 - É um protocolo baseado em SOAP que define como os clientes se comunicam com os registros UDDI.
 - É um conjunto específico de registros globais replicados.
- O UDDI inclui um esquema XML para mensagens SOAP que define um conjunto de documentos para descrever informações comerciais e de serviços, um conjunto comum de APIs para consultar e publicar informações nos diretórios e uma API para replicar entradas de diretório entre nós UDDI pares.

Fonte: <https://www.ibm.com/docs/pt-br/rsas/7.5.0?topic=standards-universal-description-discovery-integration-uddi>

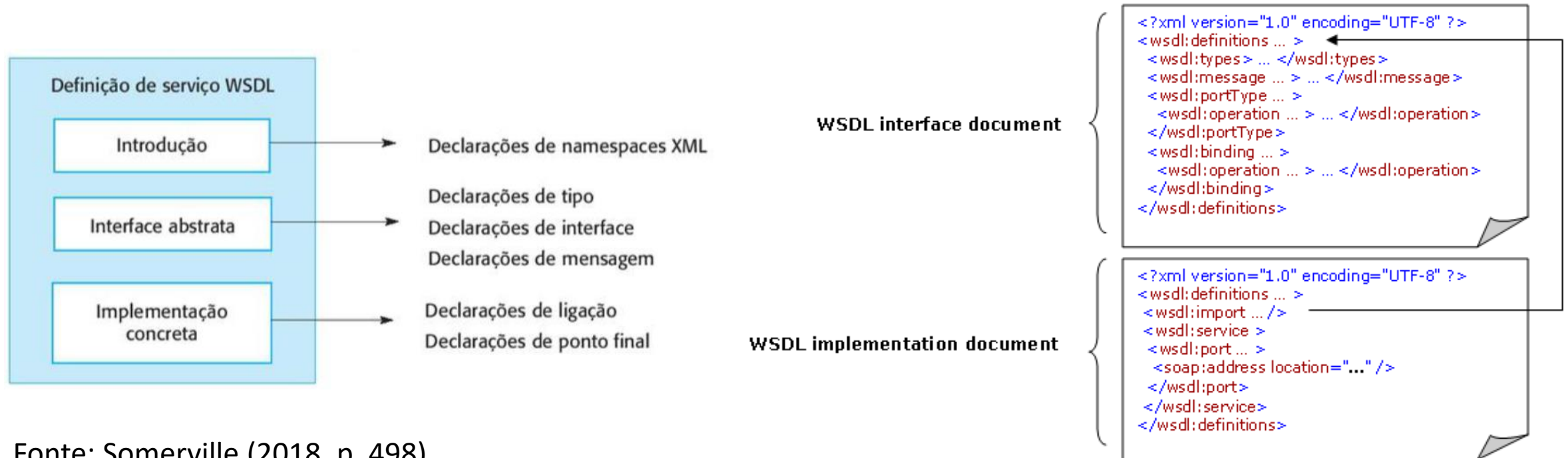
Atualmente todos os serviços UDDI estão inativos, pois os usuários preferem encontrar serviços via mecanismos de busca.

Padrões de suporte à aplicações corporativas

- ***WS-Reliable Messaging***: padrão de troca de mensagens que assegura que elas serão entregues apenas uma única vez;
- ***WS-Security***: conjunto de padrões de suporte à segurança da informação dos web-services, que inclui padrões que especificam a definição de políticas de segurança e padrões para cobrir o uso de assinaturas digitais;
- ***WS-Addressing***: define como as informações de endereço devem ser representadas em uma mensagem SOAP;
- ***WS-Transactions***: define como as transações entre serviços distribuídos devem ser coordenadas.

Fonte: Somerville (2018, p. 497)

Organização de uma especificação WSDL



Fonte: Somerville (2018, p. 498)

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:tns="http://ws.cdyne.com/WeatherWS/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/" xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  targetNamespace="http://ws.cdyne.com/WeatherWS/">
  <wsdl:types>...</wsdl:types>
  <wsdl:message name="GetWeatherInformationSoapIn">...</wsdl:message>
  <wsdl:message name="GetWeatherInformationSoapOut">...</wsdl:message>
  <wsdl:message name="GetCityForecastByZIPSoapIn">...</wsdl:message>
  <wsdl:message name="GetCityForecastByZIPSoapOut">...</wsdl:message>
  <wsdl:message name="GetCityWeatherByZIPSoapIn">
    <wsdl:part name="parameters" element="tns:GetCityWeatherByZIP"/>
  </wsdl:message>
  <wsdl:message name="GetCityWeatherByZIPSoapOut">...</wsdl:message>
  <wsdl:message name="GetWeatherInformationHttpGetIn"/>
  <wsdl:message name="GetWeatherInformationHttpGetOut">...</wsdl:message>
  <wsdl:message name="GetCityForecastByZIPHttpGetIn">...</wsdl:message>
  <wsdl:message name="GetCityForecastByZIPHttpGetOut">...</wsdl:message>
  <wsdl:message name="GetCityWeatherByZIPHttpGetIn">...</wsdl:message>
  <wsdl:message name="GetCityWeatherByZIPHttpGetOut">...</wsdl:message>
  <wsdl:message name="GetWeatherInformationHttpPostIn"/>
  <wsdl:message name="GetWeatherInformationHttpPostOut">...</wsdl:message>
  <wsdl:message name="GetCityForecastByZIPHttpPostIn">...</wsdl:message>
  <wsdl:message name="GetCityForecastByZIPHttpPostOut">...</wsdl:message>
  <wsdl:message name="GetCityWeatherByZIPHttpPostIn">...</wsdl:message>
  <wsdl:message name="GetCityWeatherByZIPHttpPostOut">...</wsdl:message>
  <wsdl:portType name="WeatherSoap">...</wsdl:portType>
  <wsdl:portType name="WeatherHttpGet">...</wsdl:portType>
  <wsdl:portType name="WeatherHttpPost">...</wsdl:portType>
  <wsdl:binding name="WeatherSoap" type="tns:WeatherSoap">...</wsdl:binding>
  <wsdl:binding name="WeatherSoap12" type="tns:WeatherSoap">...</wsdl:binding>
  <wsdl:binding name="WeatherHttpGet" type="tns:WeatherHttpGet">...</wsdl:binding>
  <wsdl:binding name="WeatherHttpPost" type="tns:WeatherHttpPost">...</wsdl:binding>
  <wsdl:service name="Weather">...</wsdl:service>
</wsdl:definitions>
```

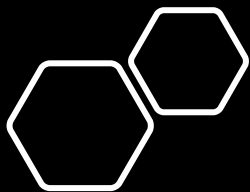
- types

- message

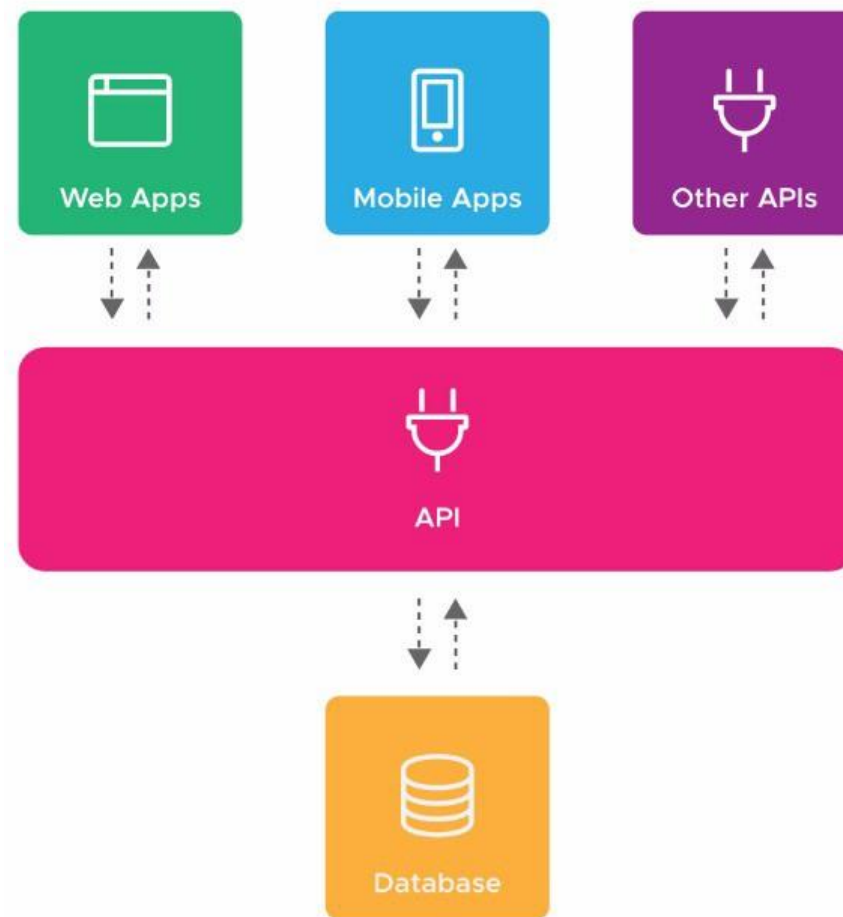
- portType

- binding

- service

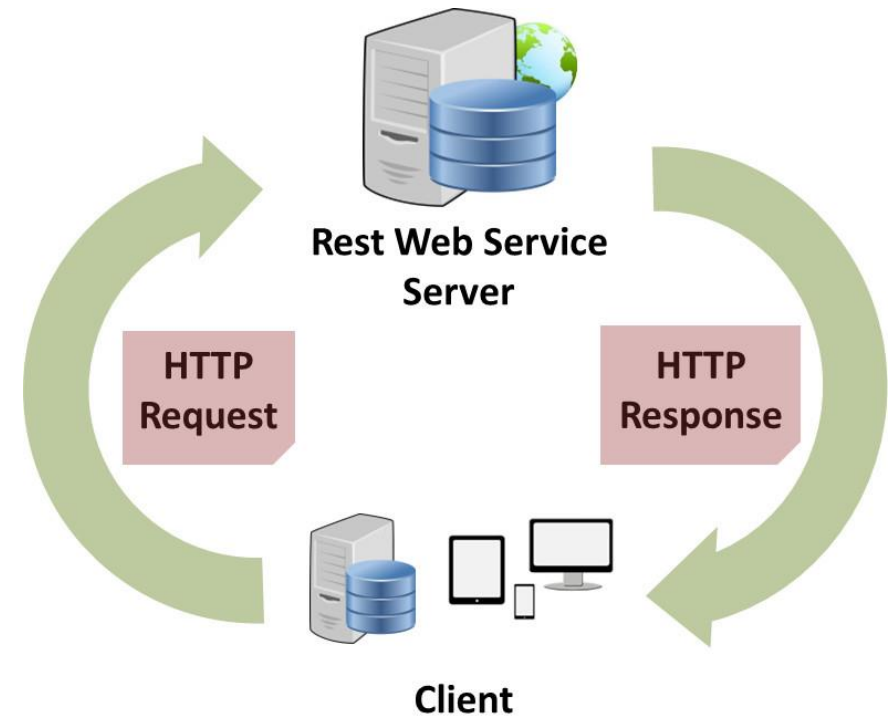


SERVIÇOS REST



O QUE É REST?

- Os serviços baseados no protocolo SOAP (XML e padrões) normalmente são mais complexos, pois envolvem uma série de aspectos relacionados à qualidade, tempo de resposta, segurança, entre outros.
- Isso significa que, por sua natureza, o protocolo SOAP é mais “pesado”, pois processar, criar e transmitir XML (envelopamento) torna a troca de mensagens mais lenta.
- Assim, a abordagem **REST** (*Representational State Transfer*) surge como uma alternativa mais “leve”, pois essencialmente envolve a troca de estados representacionais (somente dados) usando a notação JSON que, apesar fraca semanticamente, é extremamente simples sob o ponto de vista computacional, além de ser legível por máquinas e homens.



O início do REST

- **Roy Thomas Fielding**, em sua tese de doutorado em 2000, propôs a abordagem REST - *Representational State Transfer*;
- *Dissertação na íntegra:*
<https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
- Ele também definiu 6 (seis) restrições para que um *web-service* seja considerado REST. Veremos essas restrições nos próximos slides.
- **Atenção!**
 - REST não é uma nova linguagem ou uma nova tecnologia!
 - REST é um estilo arquitetural.

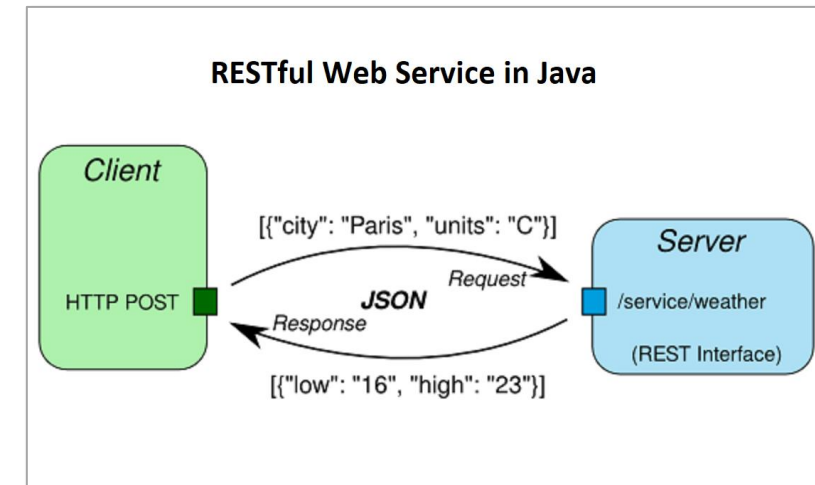
Portanto...

- REST é um estilo de arquitetura baseado em transferir representações de recursos de um servidor para um cliente. É o estilo que constitui a base da web como um todo e tem sido utilizado como um método mais simples do que SOAP/WSDL para implementar interfaces web.

Sommerville (2018, p. 501)

Segundo a Red Hat (2020)

- **REST** é um conjunto de princípios de arquitetura que atende às necessidades de aplicações mobile e serviços web leves. Como se trata de um grupo de diretrizes, são os desenvolvedores que precisam implementar essas recomendações.
- Quando uma solicitação de dados é enviada a uma **API REST**, ela normalmente é feita por meio do protocolo de transferência de hipertexto (*hypertext transfer protocol*, mais conhecido como HTTP). Depois que a solicitação é recebida, as APIs projetadas para REST (chamadas de serviços web ou APIs RESTful) retornam mensagens em diversos formatos: HTML, XML, texto simples e JSON.



API – *Application Programming Interface*

- An application programming interface, or API, enables companies to open up their applications' data and functionality to external third-party developers, business partners, and internal departments within their companies. This allows services and products to communicate with each other and leverage each other's data and functionality through a documented interface. Developers don't need to know how an API is implemented; they simply use the interface to communicate with other products and services.

IBM CLOUD EDUCATION (2020)

IMPORTANTE: considerando o conceito de API, os *web-services* REST são comumente chamados de APIs REST.

SOAP VS REST



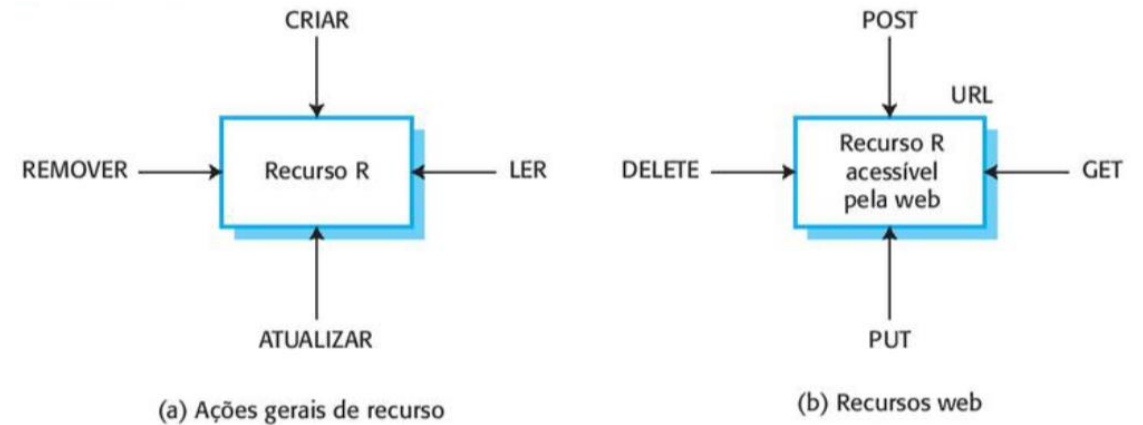
	SOAP	REST
1	Protocolo de troca de mensagens baseado em XML	Um estilo arquitetural – não uma nova tecnologia
2	Usa WSDL na comunicação entre clientes e servidores	Usa XML, JSON, CSV, texto e outros formatos
3	Invoca serviços por meio de chamadas de métodos RPC – Remote Procedure Call	Invoca serviços por meio de URLs
4	A resposta é um arquivo XML de difícil leitura para humanos	A resposta é um JSON ou XML simples, legível por humanos
5	A comunicação é feita por HTTP, mas suporta outros protocolos	A comunicação é realizada unicamente por meio de HTTP
6	Invocação de serviços via Javascript é complexa	Invocação de serviços via Javascript é simples
7	Comparado ao REST, fica devendo na performance (envelopamento SOAP)	Comparado ao SOAP, consome menos recursos, menos banda e retorna um conteúdo mais limpo.

As 6 restrições REST (by Roy Thomas Fielding)

1. **Client/Server:** clientes e servidores separados;
2. **Stateless:** sem estado – o servidor não deve guardar o estado do cliente. Cada requisição é considerada uma operação única, independente e deve ter todas as informações necessárias para ser atendida pelo server;
3. **Cacheable:** cacheável – o cliente pode fazer o cache de requisições, ou seja, decidir quando uma informação deve ser buscada no cache local ou no servidor;
4. **Uniform Interface:** interface uniforme – uma interface única entre cliente e servidor (API)
 - a. Identificação de recursos por um URI
 - b. Manipulação de recursos somente por meio de suas representações
 - c. Mensagens autodescritivas
 - d. HATEOS – *Hypermedia as the engine of application state*
5. **Sistemas em camadas:** deve ser possível colocar outros elementos entre o cliente e o servidor, tais como firewall, balanceamento de cargas, etc.
6. **Código sob demanda (opcional):** sugere um certo nível de adaptabilidade ao serviço, provendo que ele execute códigos externos.

Recurso: principal elemento REST

- Um recurso é um elemento de dados que será transferido entre cliente e servidor;
- Os recursos possuem diferentes tipos de representação, por exemplo: JSON, XML, CSV, texto puro, entre outros;
- O recurso lógico é igual em todas as representações, ou seja, se houver mapeamento entre os diferentes formatos ou notações, a perda de informação deve ser nula;
- Em uma arquitetura REST um recurso possui um URL único, e envolvem 4 operações básicas: create, read, update e delete (CRUD)



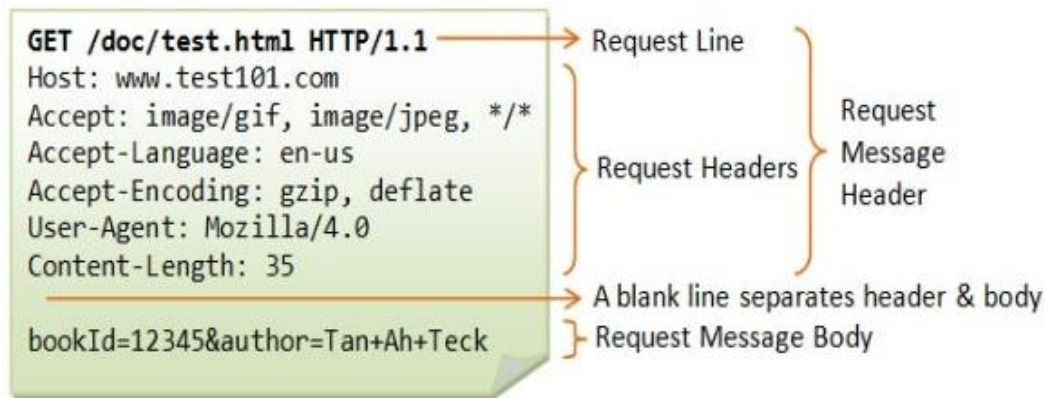
Fonte: Somerville (2019, p. 501)

A web é REST!

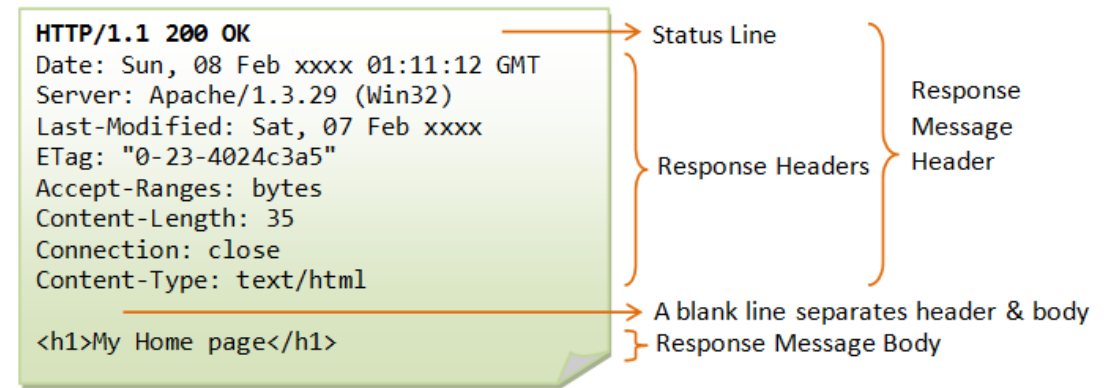
- As páginas web são recursos e possuem seu URL;
- Os protocolos HTTP e HTTPS baseiam-se em 4 verbos (ações ou operações): POST, GET, PUT e DELETE;
- POST: cria um recurso;
- GET: recupera um recurso;
- PUT: atualiza um recurso;
- DELETE: remove o recurso;
- **Atenção:** os navegadores, por padrão, aceitam somente os métodos GET e POST, entretanto, os *frameworks server-side* normalmente oferecem meios para que os verbos sejam usados corretamente via rotas e passage de parâmetros.

Request & Response

Request



Response



Exemplo de API REST

- VIA CEP é um *web-service* gratuito que fornece informações de endereço nos formatos JSON, XML, entre outros;
- A documentação é simples e informal;
- URL base: <https://viacep.com.br/>

Exemplo de pesquisa por CEP:

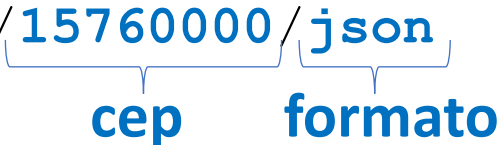

viacep.com.br/ws/01001000/json/

Cep a ser consultado

Formato



Parâmetros de requisição

- São parâmetros enviados diretamente via URL
- ***Path Params***: usam a barra para separar os parâmetros das requisição. Normalmente são obrigatórios.
 - No exemplo do serviço Via CEP, usamos dois *path params*: o cep a ser consultado e o formato da resposta:
 - `viacep.com.br/ws/15760000/json`

- ***Query Params***: normalmente não são obrigatórios. São separados pelo símbolo de interrogação (?) e usam o formato chave/valor.
 - `mywebservice.com/products/find?name=smartphone?maxvalue=2000`


Parâmetros via *header* e *body*

- ***Header Params***

- São enviados no cabeçalho da requisição;
- Não podem ser enviados via *browser request*;
- Para serem enviados, é necessário um script em JS ou um *client* específico, como o Postman (<https://www.postman.com/>) ou Insomnia (<https://insomnia.rest/>).
- Normalmente são usados para enviar dados de autenticação do serviço, controle e outras informações necessárias para que a requisição tenha sucesso;

- ***Body Params***

- São enviados no corpo da requisição
- Usados para enviar dados complexos formatos em JSON, XML e outros formatos suportados pelo serviço;
- Os dados enviados via *body* normalmente são usados para persistência, ou seja, inserção e atualização da base de dados;

Status Code do protocolo HTTP

- Em REST, cada requisição retorna um ***Status Code*** que permite ao cliente identificar se o processo foi realizado como o esperado;
- Podem ser classificados em 5 grupos:
 - ***1xx Information:*** retornam informações sobre a requisição;
 - ***2xx Success:*** a requisição foi processada com sucesso;
 - ***3xx Redirect:*** a requisição foi redirecionada a outro serviço;
 - ***4xx Client Error:*** houve erro no cliente;
 - ***5xx Server Error:*** houve algum erro no servidor.
- A lista completa com todos os *status codes* pode ser acessada em:
<https://httpstatuses.com/>

Principais *status codes* das requisições HTTP

- **200: OK** → a request foi processada com sucesso;
- **201: *Created*** → algo foi criado mas ainda será processado;
- **204: *No Content*** → algo foi excluído com sucesso, mas não foi retornado nada;
- **400: *Bad Request*** → o cliente faz uma requisição passando dados inválidos;
- **401: *Unauthorized*** → o cliente não está autorizado a realizar a operação. Normalmente o cliente não está autenticado no serviço;
- **403: *Forbidden*** → o cliente não possui permissão na operação em questão, mesmo estando autenticado.
- **404: *Not Found*** → o recurso requisitado não existe;
- **405: *Method Not Allowed*** – O usuário não possui permissão ao path de requisição;
- **500: *Internal Server Error*** – Ocorreu alguma falha no servidor;

1XX Informational	4XX Client Error Continued
100 Continue	409 Conflict
101 Switching Protocols	410 Gone
102 Processing	411 Length Required
	412 Precondition Failed
	413 Payload Too Large
	414 Request-URI Too Long
	415 Unsupported Media Type
	416 Requested Range Not Satisfiable
	417 Expectation Failed
	418 I'm a teapot
	421 Misdirected Request
	422 Unprocessable Entity
	423 Locked
	424 Failed Dependency
	426 Upgrade Required
	428 Precondition Required
	429 Too Many Requests
	431 Request Header Fields Too Large
	444 Connection Closed Without Response
	451 Unavailable For Legal Reasons
	499 Client Closed Request
2XX Success	5XX Server Error
200 OK	500 Internal Server Error
201 Created	501 Not Implemented
202 Accepted	502 Bad Gateway
203 Non-authoritative Information	503 Service Unavailable
204 No Content	504 Gateway Timeout
205 Reset Content	505 HTTP Version Not Supported
206 Partial Content	506 Variant Also Negotiates
207 Multi-Status	507 Insufficient Storage
208 Already Reported	508 Loop Detected
226 IM Used	510 Not Extended
	511 Network Authentication Required
	599 Network Connect Timeout Error
3XX Redirectional	
300 Multiple Choices	
301 Moved Permanently	
302 Found	
303 See Other	
304 Not Modified	
305 Use Proxy	
307 Temporary Redirect	
308 Permanent Redirect	
4XX Client Error	
400 Bad Request	
401 Unauthorized	
402 Payment Required	
403 Forbidden	
404 Not Found	
405 Method Not Allowed	
406 Not Acceptable	
407 Proxy Authentication Required	
408 Request Timeout	

Verbos HTTP

- **GET:**

- recupera um recurso;
- Aceita parâmetros via URL e via Header (query e path params)
- É o único verbo que não suporta ***body params***;
- Quando realizada com sucesso retorna 200 (OK);
- Quando retorna algum erro, normalmente retorna 404 (not found) ou 400 (bad request)

- **POST**

- Usado na criação de novos recursos;
- Retorna 200 (*OK*) ou 201 (*created*) em uma requisição bem sucedida;
- Suporta parâmetros via URL, header e body;

Verbos HTTP

- **PUT**

- Atualiza um recurso – é necessário ter o recurso original passado via body;
- Quando bem sucedida, retorna 200 (OK) ou 204 (*no content*);
- Suporta parâmetros via URL, header e body;

- **DELETE**

- Usado para excluir recursos;
- Retorna 200 (*OK*) ou 204 (*no content*) em uma requisição bem sucedida;
- Suporta parâmetros via URL, header e body;

Outros verbos HTTP

- **PATCH**

- Atualizações parciais. Exemplo: atualizar apenas 3 campos de um registro que possui 30.

- **HEAD**

- Parecido com o GET, mas sua resposta são apenas uma linha de dados e headers. Não possui corpo de resposta;

- **TRACE**

- Permite recuperar o conteúdo de uma requisição de volta, com propósitos de depuração (debug) da aplicação;

- **OPTIONS**

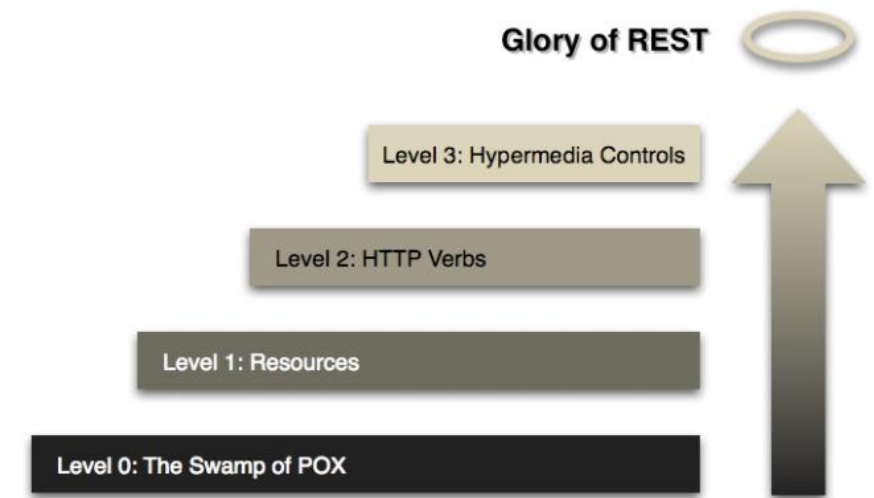
- Usado pelo client para encontrar operações HTTP e outras opções suportadas pelo serviço

- **CONNECT**

- Estabelece uma conexão entre cliente e servidor

NÍVEIS DE MATURIDADE REST

- Segundo o modelo de maturidade de Richardson, Restful é o nível mais elevado do REST.
- **Nível 0:** Pântano de XML → apesar de usar serviços REST, tudo está amontoado em apenas um *endpoint*;
- **Nível 1:** Recursos → cada recurso possui seu *endpoint* de entrada;
- **Nível 2:** Verbos HTTP → há o uso correto e semântico dos verbos HTTP (*post, get, put, delete, patch*)
- **Nível 3:** controles hipermídia → a resposta deve trazer links para operações sobre o recurso permitindo a navegação hipermídia sobre os dados retornados.



Fonte: <https://martinfowler.com/articles/richardsonMaturityModel.html>

HATEOAS

- **HATEOAS** – *Hypermedia as the engine of application state*
- É uma das restrições arquiteturais de Roy Thomas Fielding;
- As respostas incluem *links* que permitem navegar entre seus *endpoints*, provendo uma navegação dinâmica entre os dados da aplicação, assim como em uma página HTML;

```
HTTP/1.1 200 OK
Content-Type: application/vnd.acme.account+json
Content-Length: ...

{
  "account": {
    "account_number": 12345,
    "balance": {
      "currency": "usd",
      "value": 100.00
    },
    "links": {
      "deposit": "/accounts/12345/deposit",
      "withdraw": "/accounts/12345/withdraw",
      "transfer": "/accounts/12345/transfer",
      "close": "/accounts/12345/close"
    }
  }
}
```

Documentação

- A documentação é parte essencial do serviço;
- Os desenvolvedores de aplicações *clients* usam a documentação como um meio de comunicação eficiente com os desenvolvedores do serviço;
- A documentação deve definir a API de serviços considerando:
 - Os *endpoints* (URL) da aplicação REST;
 - As operações suportadas;
 - As restrições de uso do serviço;
 - Os tipos de retorno
 - Entre outros.
- Iremos aprender como usar o **Swagger** (<https://swagger.io/>) para documentar a API de serviços.

The screenshot displays the Swagger Petstore API documentation. At the top, there's a green header with the Swagger logo, the API URL 'http://petstore.swagger.io/v2/swagger.json', an 'api_key' input field, and an 'Explore' button. Below the header, the title 'Swagger Petstore' is followed by a brief description: 'This is a sample server Petstore server. You can find out more about Swagger at <http://swagger.io> or on [#swagger](irc.freenode.net). For this sample, you can use the api key 'special-key' to test the authorization filters.'

Links for 'Find out more about Swagger' include 'http://swagger.io', 'Contact the developer', and 'Apache 2.0'.

The main section is titled 'pet : Everything about your Pets' and includes sub-headers 'Show/Hide', 'List Operations', and 'Expand Operations'. It features a 'POST /pet' endpoint with a description 'Add a new pet to the store'. The 'Parameters' section shows a 'body' parameter with a required value and a description 'Pet object that needs to be added to the store'. The 'Response Messages' section shows a '405 Invalid input' response. A 'Try it out!' button is present.

Below the main section, there's a list of other endpoints: 'PUT /pet' (Update an existing pet), 'GET /pet/findByStatus' (Finds Pets by status), 'GET /pet/findByTags' (Finds Pets by tags), 'DELETE /pet/{petId}' (Deletes a pet), 'GET /pet/{petId}' (Find pet by ID), 'POST /pet/{petId}' (Updates a pet in the store with form data), and 'POST /pet/{petId}/uploadImage' (uploads an image).

The bottom section is titled 'store : Access to Petstore orders' and 'user : Operations about user', both with 'Show/Hide', 'List Operations', and 'Expand Operations' sub-headers. At the very bottom, there's a footer with '[BASE URL: /v2 , API VERSION: 1.0.0]' and a 'VALID' button.

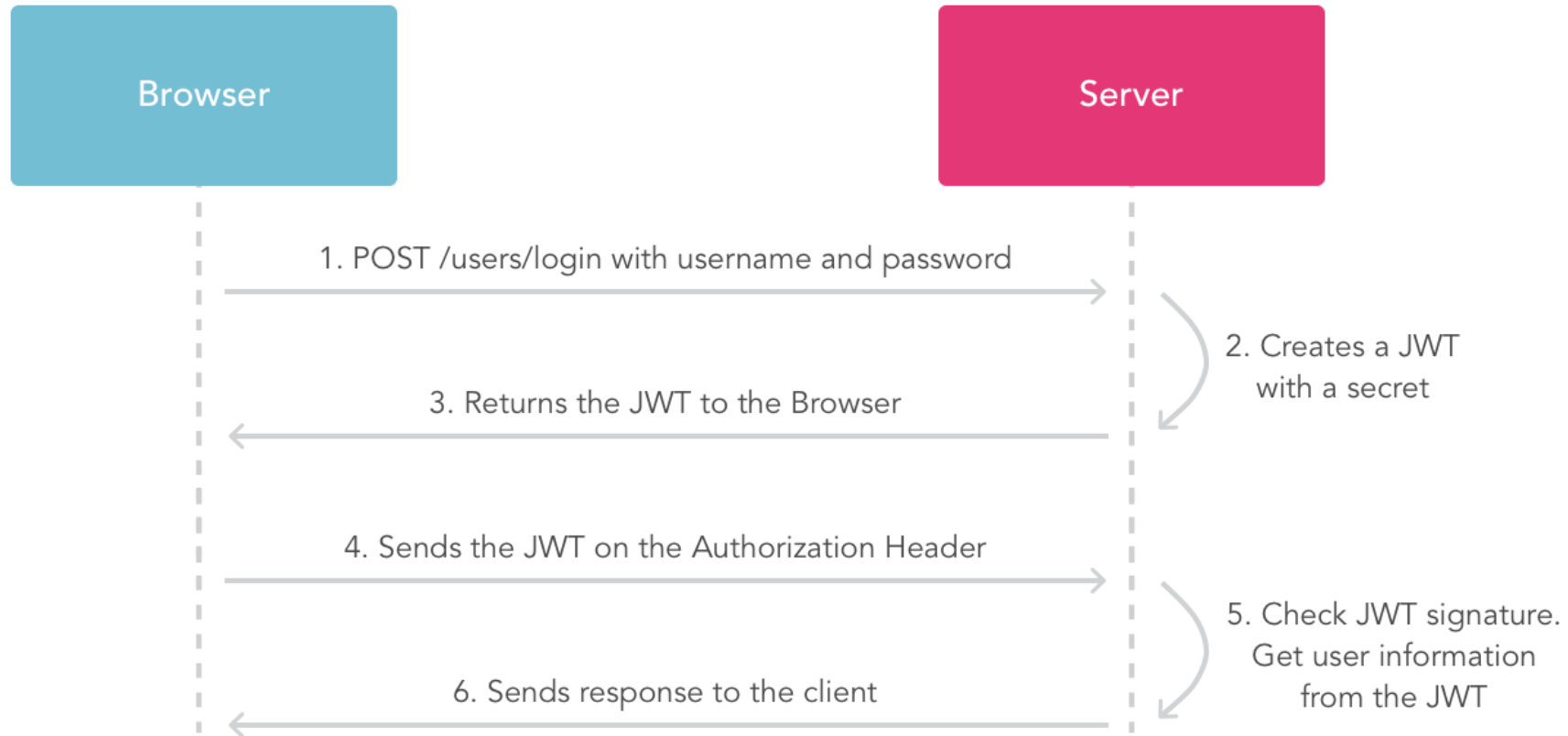
Autenticação

- Para que as aplicações *client* possam usar a API, normalmente é necessário fazer autenticação;
- Isso é necessário para evitar que qualquer aplicação, principalmente as não autorizadas, usem sua API;
- A autenticação é realizada via JWT – JSON Web Token;

What is JSON Web Token?

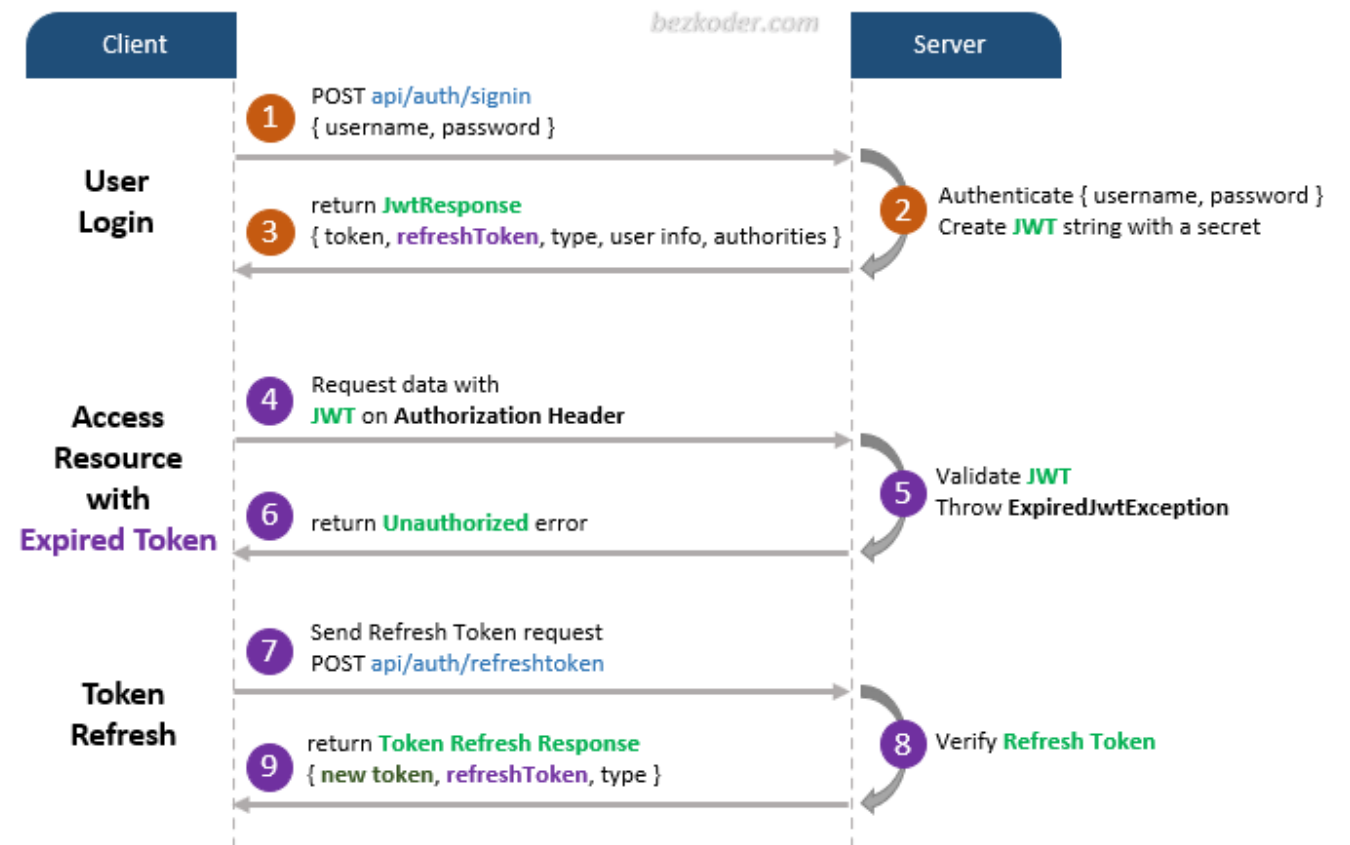
*JSON Web Token (JWT) is an open standard ([RFC 7519](#)) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the **HMAC** algorithm) or a public/private key pair using **RSA** or **ECDSA**.*

Ciclo de vida JWT



Refresh Token

- Os *tokens* possuem tempo de expiração;
- Quando uma requisição é realizada com um *token* expirado, será retornado um erro, normalmente *status code* 401 (*unauthorized*).
- Assim, é necessário usar um *refresh token* que possibilita a obtenção de um novo *token* de acesso e um novo *refresh token* sem enviar novamente dados de usuários.



Access Token vs Refresh Token

- ***Access token***

Permitem acesso a recursos protegidos do serviço, além de identificar o usuário no sistema. É capaz de suportar dados extras, que identificam o usuário e sua sessão, como o IP por exemplo. Normalmente possui um tempo de vida curto;

- ***Refresh token***

Permite obter um novo *access token* sem a necessidade de uma nova autenticação envolvendo as credenciais do usuário (nome de usuário e senha). Possui um tempo de vida mais longo

Anatomia de um JWT

- **Header:**

- O tipo de algoritmo usado na geração do token;
- O tipo de Token

- **Payload:**

- Dados que são trafegados, normalmente informações do Token: data de expiração, criador do Token, entre outros;

- **Assinatura:**

- Hash gerador a partir do *header* e do *payload* que verifica se o *token* é válido.

The image shows a screenshot of the <https://jwt.io/> website. It is divided into two main sections: 'Encoded' and 'Decoded'.

Encoded: This section has a sub-header 'PASTE A TOKEN HERE'. Below it, a text area contains a JWT token: `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c`. The token is color-coded: 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.' is red, 'eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.' is purple, and '.SflKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c' is blue.

Decoded: This section has a sub-header 'EDIT THE PAYLOAD AND SECRET'. It displays the decoded components of the token in three panels:

- HEADER: ALGORITHM & TOKEN TYPE:** Shows a JSON object: `{ "alg": "HS256", "typ": "JWT" }`.
- PAYLOAD: DATA:** Shows a JSON object: `{ "sub": "1234567890", "name": "John Doe", "iat": 1516239022 }`.
- VERIFY SIGNATURE:** Shows the signature generation process: `HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), your-256-bit-secret)`. There is a checkbox for 'secret base64 encoded' which is currently unchecked.

Fonte: <https://jwt.io/>

Versionamento

- Preparar sua API para novas *features* e correções é essencial para manter o serviço sempre disponível aos seus clients;
- Uma mudança na regra de negócio, na legislação, entre outras, são mudanças que causam impactos no serviço. Logo é necessário estar preparado;
- Existem diversos meios para versionar sua API, mas a mais comum é via URL usando *path params* após a **URL base**. Veja:
 - `http://meuwebservice/client/v1/find`
 - `http://meuwebservice/client/v2/find`
- Dessa forma é possível publicar diversas versões de sua API, retirando do versões mais antigas após informar todos os seus clientes.
- Veremos como trabalhar versionamento de APIs.

Resumo

- A abordagem orientada a serviços (*web-service*) é melhor maneira de desenvolver aplicações fracamente desacopladas, dinâmicas, com grande capacidade de evolução e integração, além de serem escaláveis e facilmente documentadas;
- Com o crescimento dos dispositivos *mobiles*, *weareables* e de outros conceitos como IoT (*Internet of Things*), AI (*Artificial Intelligence*), as aplicações baseadas em serviços possuem papel fundamental na evolução tecnológica em curso.
- REST é uma abordagem mais leve se comparada ao protocolo SOAP, pois trata-se de um estilo arquitetural que trafega geralmente informações em formato JSON, por isso é a abordagem mais utilizada na maioria das aplicações de serviços atualmente;
- Há também o conceito de *microserviço*, em que cada parte da aplicação pode ser desenvolvida de maneira totalmente independente, inclusive usando linguagens de programação e plataformas diferentes.

Referências

IBM CLOUD EDUCATION. **Application Programming Interface (API)**. 2020. Disponível em: <<https://www.ibm.com/cloud/learn/api>>. Acesso em: 25 set. 2021.

IBM. **UDDI (Descrição, Descoberta e Integração Universais)**. 20??. Disponível em: <<https://www.ibm.com/docs/pt-br/rsas/7.5.0?topic=standards-universal-description-discovery-integration-uddi>>. Acesso em: 12 out 2021.

JWT.IO. **Introduction to JSON Web Tokens**. 20??. Disponível em: <<https://jwt.io/introduction>>. Acesso em: 22 out. 2021.

MDN WEB DOCS. **HTTP request methods**. 2021. Disponível em <<https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>>. Acesso em 10 out. 2021.

PEYROTT, S. **JWT HANDBOOK**. 2018. Disponível em: <<https://auth0.com/resources/ebooks/jwt-handbook>>. Acesso em: 10 nov. 2021.

RED HAT. **O que é arquitetura orientada a serviços (SOA)?** 2020. Disponível em: <<https://www.redhat.com/pt-br/topics/cloud-native-apps/what-is-service-oriented-architecture>> Acesso em: 13 out. 2021.

SOMMERVILLE, I. **Engenharia de Software**. 10 ed. São Paulo: Pearson Education do Brasil, 2018.

W3C. **Web Services Architecture**. 2004. Disponível em: <<https://www.w3.org/TR/ws-arch/>> Acesso em: 20 out. 2021.