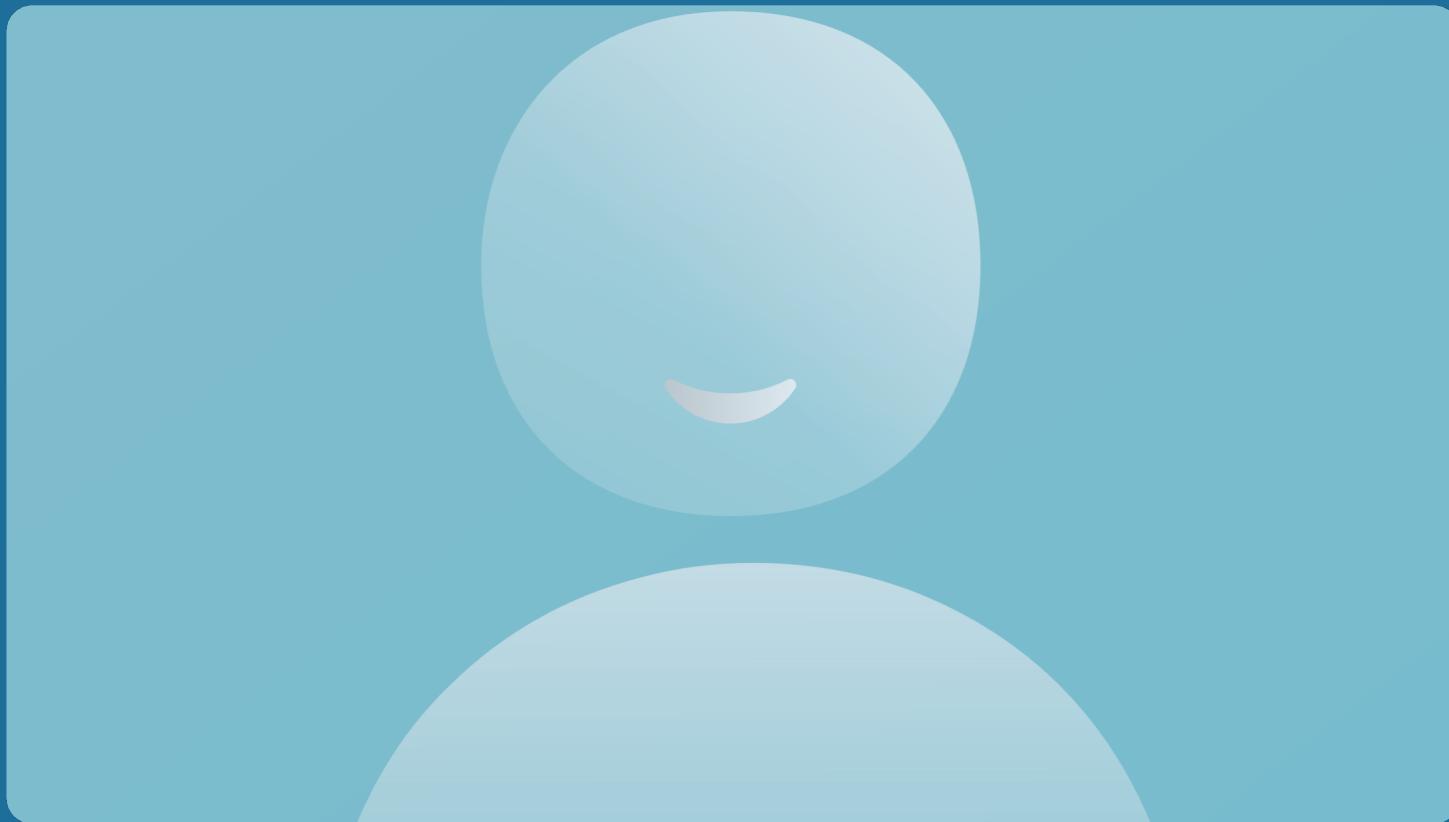


APIs e Web Services



Tópicos

- Unidade 1 - Introdução
 - Introdução ao Mundo das APIs
 - Protocolo HTTP
- Unidade 2 - Web APIs
 - Aspectos arquiteturais de APIs
 - Estilos: SOAP, REST, GraphQL, Webhooks e Websockets
- Unidade 3 - Segurança em APIs
 - Fundamentos de Segurança
 - Autenticação no Protocolo HTTP
 - Tecnologias e Frameworks: JWT, Oauth e CORS
- Unidade 4 - Gerenciamento de APIs
 - API Lifecycle Management (ALM)
 - Arquitetura de Microserviços e API Gateway
 - Boas práticas no desenvolvimento de APIs

APIs & Web Services

Informações Gerais

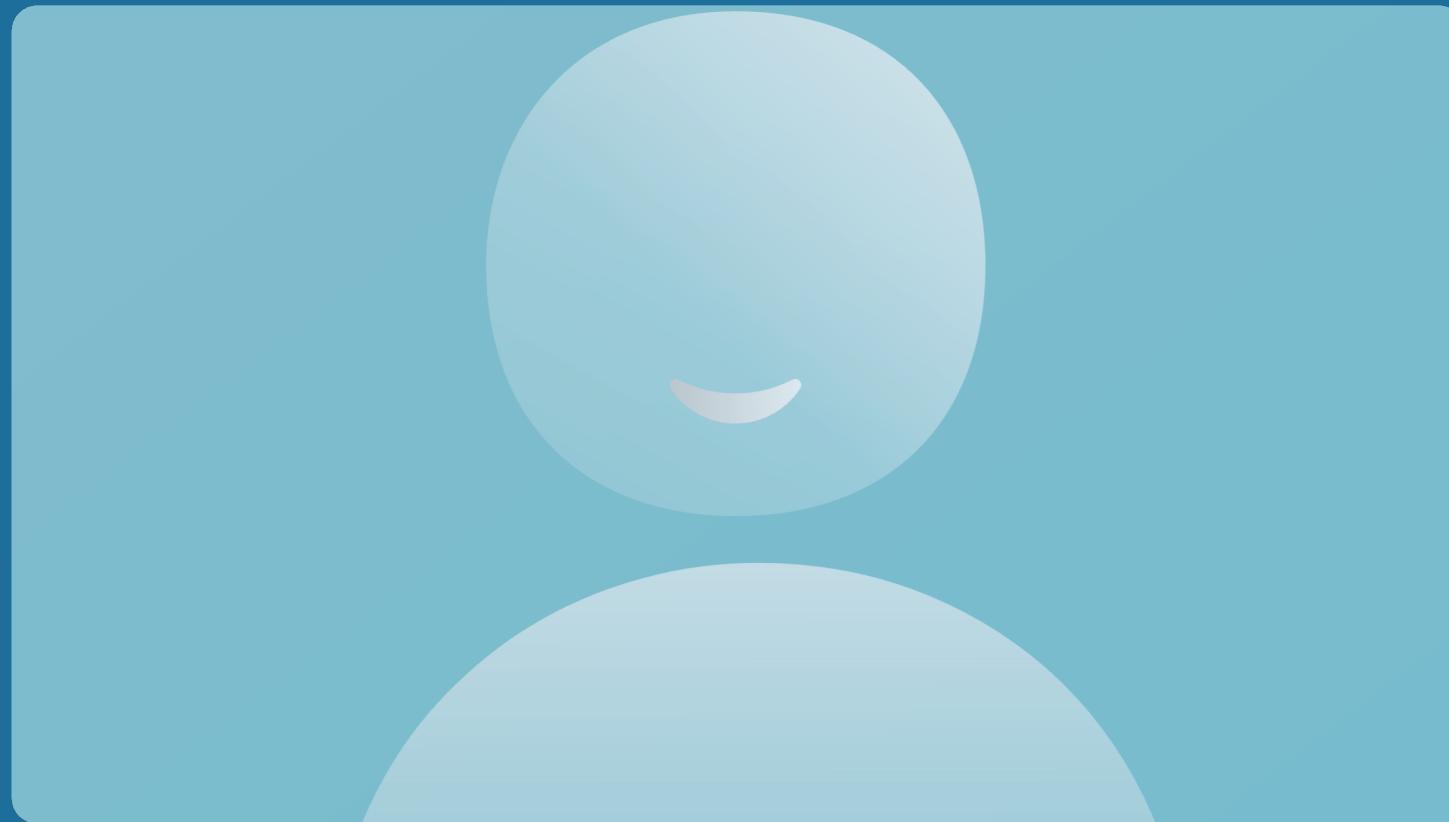
Ementa

Evolução das APIs. Gestão do ciclo de vida das APIs. Melhores práticas no projeto de API ferramentas para documentação de APIs. Mecanismos de segurança: autenticação, a vulnerabilidades. Abordagens arquiteturais de APIs: RESTful, GraphQL, WebSockets, WebF Streaming.

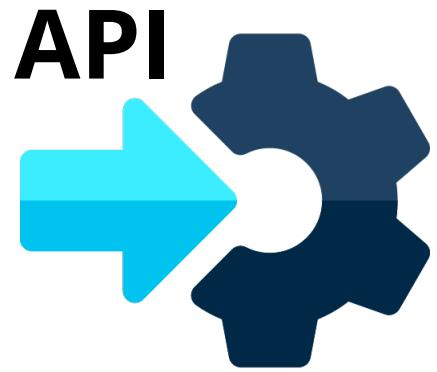
Bibliografia

- [Designing Web APIs](#). Brenda Jin, Saurabh Sahni, Amir Shevat. O'Reilly Media, Inc. (2018)
- [Mastering API Architecture](#). James Gough, Daniel Bryant, Matthew Auburn. O'Reilly Media, Inc. (2022)
- [API Design Patterns](#). John J. (JJ) Geewax. Manning Publications (2021)
- [Designing APIs with Swagger and OpenAPI](#). Josh Ponelat, Lukas Rosenstock. Manning Publications (2022)
- [Microservice APIs](#). Jose Haro. Manning Publications (2023)

Aspectos Arquiteturais de APIs



Aspectos Arquiteturais de APIs



Arquiteturas

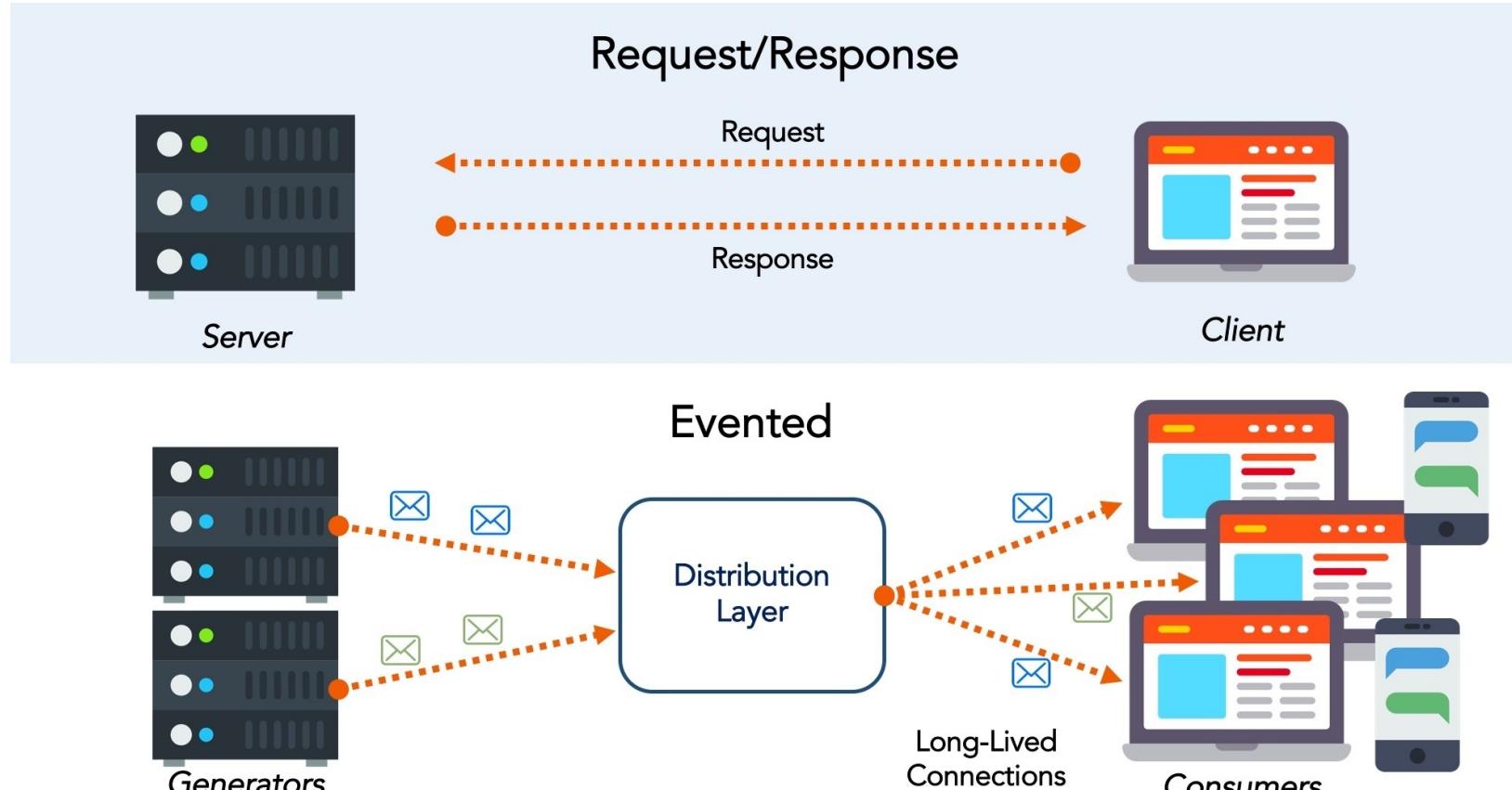
Representação
de Dados

Documentação



Abordagens de Comunicação em APIs

Request/Response vs Event Driven

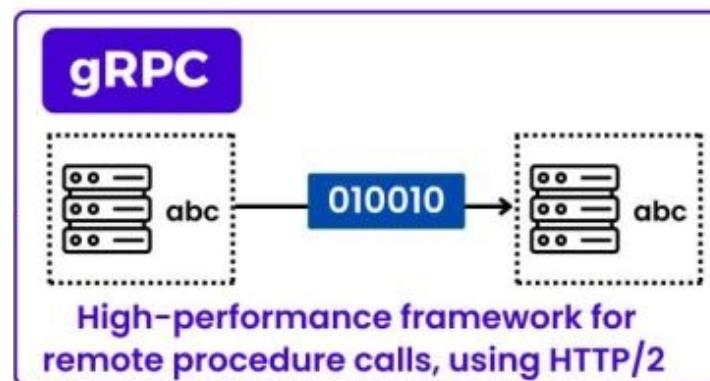
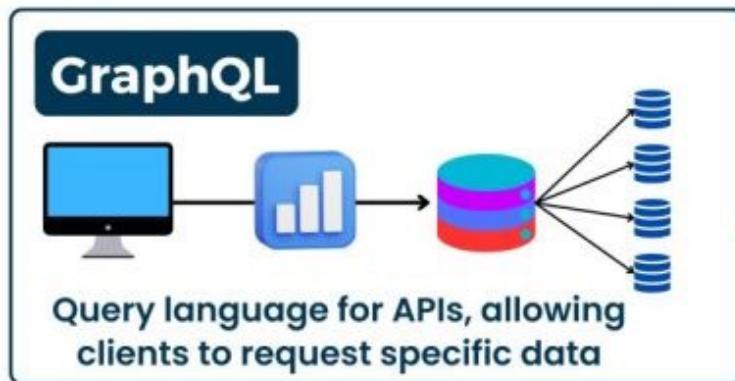
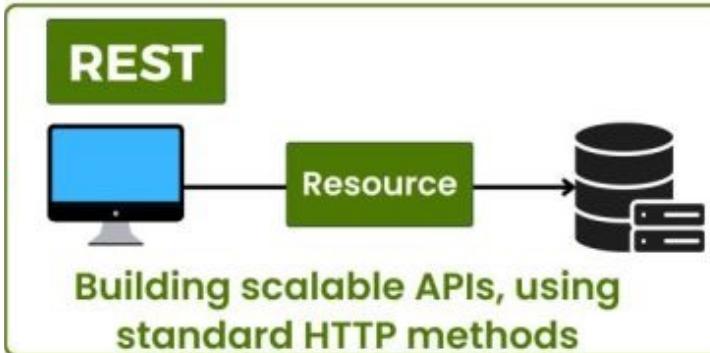
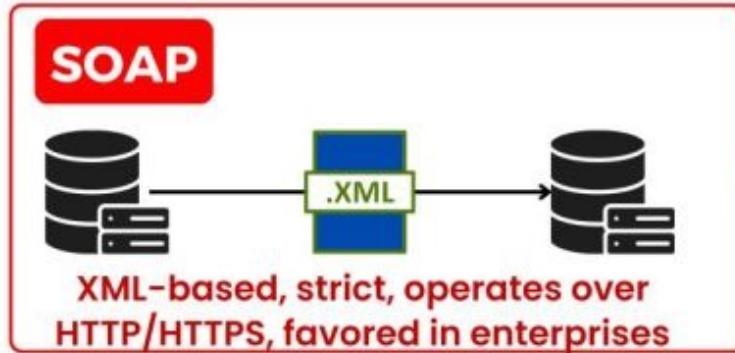


Fonte: [Getting Started with Building Realtime API Infrastructure](#) (Justin Baker)

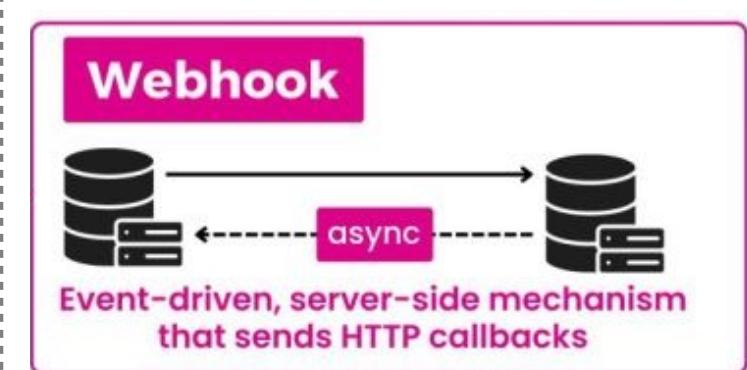
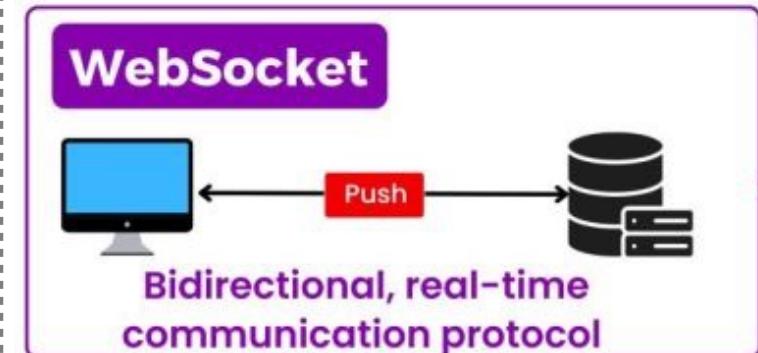


Principais estilos arquiteturais de APIs

Request/Response



Event Driven



Fontes:

- [Top Architectural Styles for APIs in 2023](#)
- [2023 State of the API Report](#) (Postman)



58

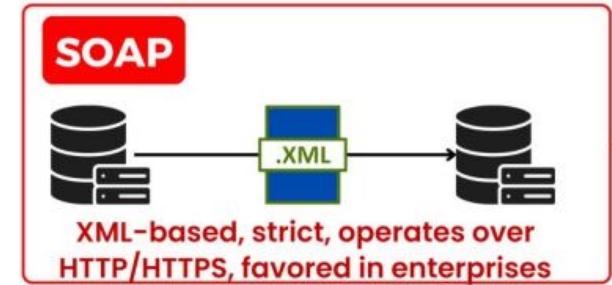


Prof. Rommel Vieira Carneiro

Estilos arquiteturais – SOAP

Simple Object Access Protocol (SOAP)

O SOAP é uma proposta robusta para a construção de APIs, que mescla complexidade e potência. Utiliza XML para definir comunicações estruturadas e se baseia em um par de cliente e servidor SOAP.



Casos de uso | Exemplos

- Integração de sistemas empresariais, como sistemas de gerenciamento de pedidos e contabilidade.
- Serviços de pagamento online.
- Ex: [Nota Fiscal Eletrônica \(NFe\) do Governo Brasileiro](#)

Referências sugeridas

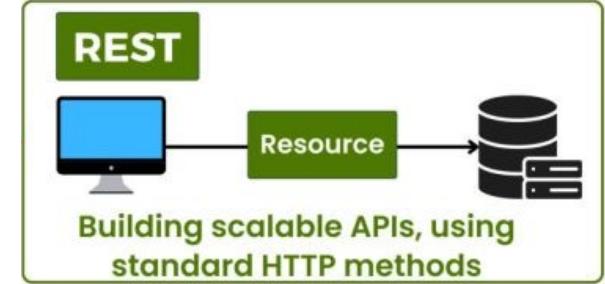
- [SOAP Specifications](#) (W3C)



Estilos arquiteturais – REST

REpresentational State Transfer (REST)

REST é um estilo arquitetural que aproveita principalmente os métodos HTTP. Permite uma interação fácil com recursos, tornando-se um padrão de referência para uma infinidade de aplicações e APIs modernas.



Casos de uso | Exemplos

- APIs de mídia social, onde os clientes acessam dados de usuário e publicações.
- APIs de serviços da Web
- Exs: [GitHub Rest API](#)

Referências sugeridas

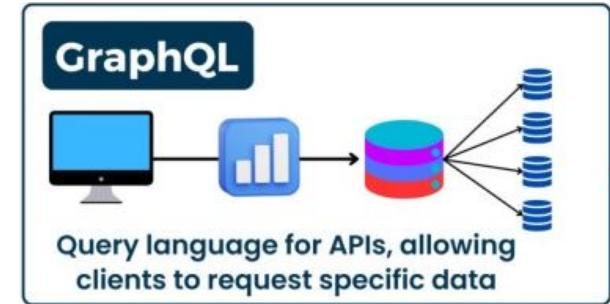
- [Architectural Styles and the Design of Network-based Software Architectures](#) (Fielding, 2000)
- [RFC 7231 - Hypertext Transfer Protocol \(HTTP/1.1\): Semantics and Content](#)



Estilos arquiteturais - GraphQL

Graph Query Language (GraphQL)

GraphQL oferece flexibilidade e precisão. Permite que os clientes solicitem exatamente o que precisam, reduzindo a redundância e melhorando o desempenho. Alguns percebem o GraphQL como um substituto do REST, porém os padrões podem coexistir se complementando.



Casos de uso | Exemplos

- Aplicações c/ diferentes clientes requerendo de diferentes conjuntos de dados, como aplicativos de análise de dados
- Exs: [GitHub GraphQL](#)

Referências sugeridas

- [GraphQL | A query language for your API](#)

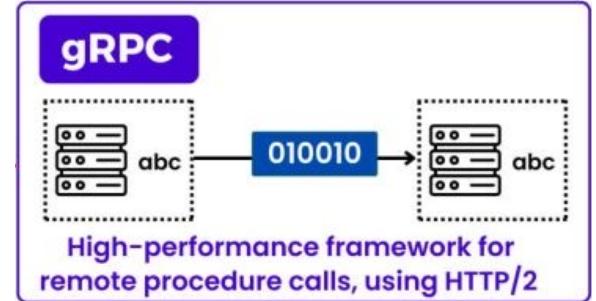


Estilos arquiteturais – RPC | gRPC

Google Remote Procedure Call (gRPC)

gRPC utiliza o protocolo HTTP/2 com transmissão de dados binários.

Prioriza desempenho e velocidade, especialmente para arquiteturas de microsserviços.



Casos de uso | Exemplos

- Arquiteturas de microsserviços onde a eficiência de comunicação é crucial, como em sistemas de transporte público em tempo real.
- Exs: [Slack Web API](#) (RPC)

Referências sugeridas

- [gRPC.io | Documentation](#)

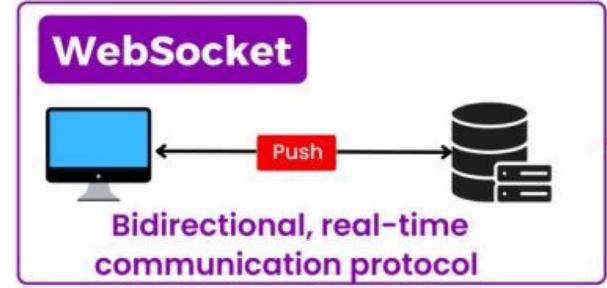


Estilos arquiteturais – WebSocket

Web Sockets

Oferece uma alternativa de comunicação em tempo real e bidirecional.

São ideais para aplicativos de chat, transmissão ao vivo e troca de dados em tempo real, evitando o polling (short polling ou long polling).



Casos de uso | Exemplos

- Aplicativos de chat em tempo real.
- Plataformas de jogos online com funcionalidade multiplayer.
- Placar de jogos

Referências sugeridas

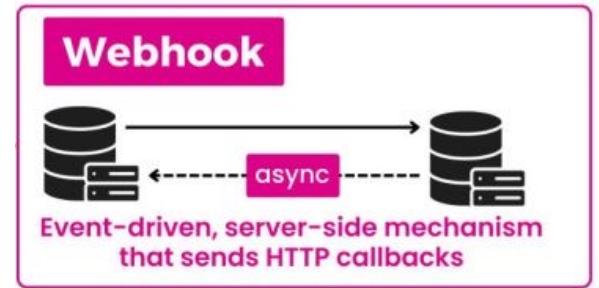
- [RFC 6455 - The WebSocket Protocol](#)
- [Socket.IO](#) (Implementação similar)



Estilos arquiteturais - WebHook

WebHooks

Voltados para as arquiteturas orientadas a eventos, fornecem uma alternativa para que servidores notifiquem clientes quando eventos específicos ocorrem.



Casos de uso | Exemplos

- Notificações de transações financeiras.
- Atualizações em tempo real em aplicativos de colaboração.

Referências sugeridas

- [Web hooks to revolutionize the web](#) (Jeff Lindsay, 2007)



Comparativos entre Estilos

ABORDAGEM	VANTAGENS	DESVANTAGENS
REST	<ul style="list-style-type: none">• Fácil interação com recursos• Padrão amplamente adotado	<ul style="list-style-type: none">• Pode levar a múltiplas requisições para ações complexas• Falta de flexibilidade em algumas situações
SOAP	<ul style="list-style-type: none">• Comunicação estruturada e rica em recursos• Integração com sistemas corporativos	<ul style="list-style-type: none">• Complexidade de implementação• Overhead devido ao uso de XML• Requer cliente e servidor SOAP
GraphQL	<ul style="list-style-type: none">• Flexibilidade para clientes obterem o que precisam• Melhor eficiência de rede	<ul style="list-style-type: none">• Requer mecanismo de consulta mais complexo no servidor• Gera consultas complexas e ineficientes se mal projetado
gRPC	<ul style="list-style-type: none">• Performance pela comunicação binária e multiplexação• Geração automática de código	<ul style="list-style-type: none">• Requer suporte a HTTP/2• Pode ser mais difícil de depurar devido à natureza binária• Maior curva de aprendizado em comparação com APIs REST
WebSockets	<ul style="list-style-type: none">• Atualizações em tempo real para clientes.• Redução da latência em comparação com polling	<ul style="list-style-type: none">• Requer suporte em ambos os lados (cliente e servidor)• Maior complexidade em comparação com APIs tradicionais
Webhooks	<ul style="list-style-type: none">• Resposta em tempo real a eventos específicos• Redução de overhead de consulta constante	<ul style="list-style-type: none">• Pode exigir lógica adicional para confirmações e repetições• Falha de entrega pode ocorrer se não implementado corretamente

Comparativos entre Estilos

SOAP vs REST

ASPECTO	SOAP	REST
Estrutura	Protocolo baseado em XML	Protocolo baseado no estilo arquitetural
Comunicação de Dados	Utiliza um padrão de mensagens baseado em XML	Utiliza XML ou JSON para envio e recebimento dos dados
Comunicação	Invoca serviços por meio de HTTP e RPC (remote procedure call)	Baseado 100% em HTTP e nas URLs
Formato da Mensagem	Resultado codificado no padrão SOAP	Resultado facilmente interpretado por um humano
Compatibilidade com Javascript	Complexa	Simplificada
Desempenho	Médio	Alto



Comparativos entre Estilos

SOAP vs REST

Requisição SOAP

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
    <soap:body pb="http://www.acme.com/phonebook">
        <pb:GetUserDetails>
            <pb:UserID>12345</pb:UserID>
        </pb:GetUserDetails>
    </soap:Body> </soap:Envelope>
```

Requisição REST

GET <http://www.acme.com/phonebook/UserDetails/12345>



Comparativos entre Estilos

GraphQL vs REST

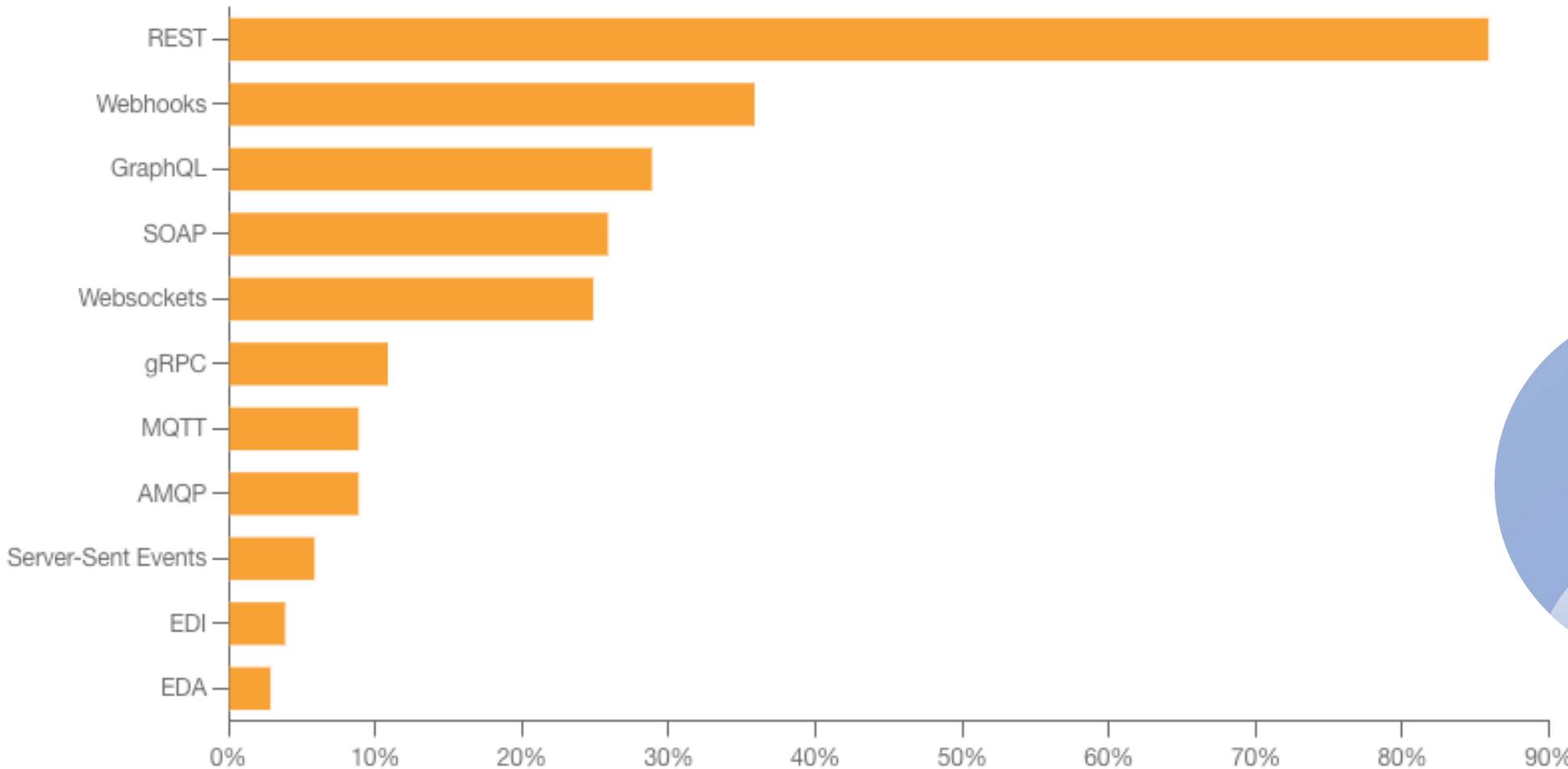
GraphQL	REST
Linguagem de consulta oferece eficiência e flexibilidade para resolver problemas comuns ao integrar APIs	Um estilo arquitetural amplamente visto como padrão para o desenho de APIs
Disponível via um único endpoint HTTP que provê todas as funcionalidades do serviço	Disponível via um conjunto de URLs, cada uma expondo um único recurso.
Utiliza uma arquitetura baseada no cliente	Utiliza uma arquitetura baseada no servidor
Dificulta o uso do mecanismo de cache	Utiliza cache de maneira simplificada e automática
Não trabalha com versionamento de API	Suporta múltiplas versões da API
Trabalha apenas com JSON	Suporta múltiplos formatos de dados (JSON, XML, etc)
Oferece uma única forma de documentação: GraphiQL	Oferece uma faixa de opções para documentação automatizada (OpenAPI e API Blueprint)
Dificulta o uso de códigos de status do protocolo HTTP para identificação de erros.	Utiliza os códigos de status do protocolo HTTP para identificar facilmente os erros

Fonte: [GraphQL vs. REST: Which Is the Best for API Development?](#)



Arquiteturas de APIs mais utilizadas

Postman 2023 State of the API Report



Fonte: [2023 State of the API Report | API Technologies](#)



69



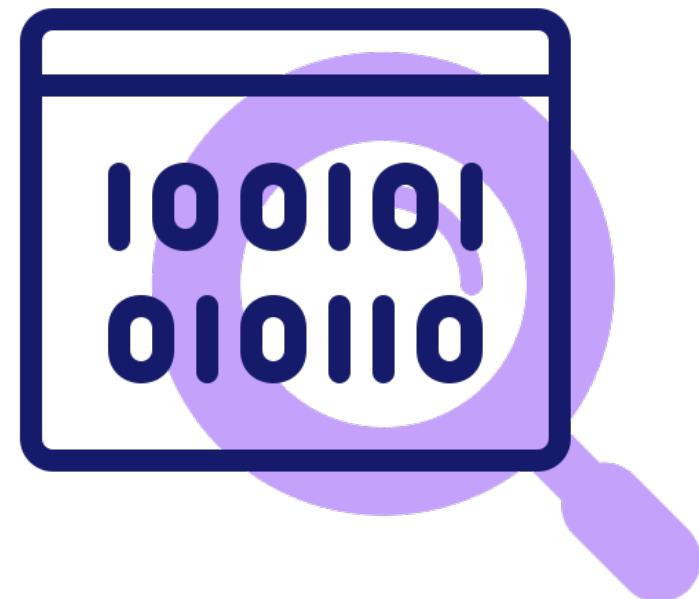
Prof. Rommel Vieira Carneiro

Representação de Dados em APIs

A **representação de Dados** é a maneira como os dados trocados entre aplicações são organizados e/ou descritos.

Alguns formatos utilizados em APIs

- [Extensible Markup Language \(XML\)](#)
- [JavaScript Object Notation \(JSON\)](#)
- [YAML Ain't Markup Language \(YAML\)](#)
- Comma Separated Values (CSV)
- Protocol Buffers (ProtoBuf)
- Binary JSON (BSON)
- [MessagePack](#)



Representação de Dados em APIs

XML vs JSON vs YAML



XML	JSON	YAML
<pre><Servers> <Server> <name>Server1</name> <owner>John</owner> <created>12232012</created> <status>active</status> </Server> </Servers></pre>	<pre>{ Servers: [{ name: "Server1", owner: "John", created: "12232012", status: "active" }] }</pre>	<pre>Servers: - name: Server1 owner: John created: 12232012 status: active</pre>

Fonte: [Comparison between XML Vs JSON Vs YAML](#)

Representação de Dados em APIs

MIME (Multipurpose Internet Mail Extensions)

Padrão utilizado para indicar o formato do conteúdo em uma mensagem de e-mail e muito utilizado em outras aplicações da Internet.

Características

- Tipos possíveis definidos pela [Internet Assigned Numbers Authority \(IANA\)](#)
- Um tipo é dividido em duas partes tipo/subtipo e pode ter parâmetros opcionais (charset, boundary)
- Os valores expressos no cabeçalho **Content-Type** seguem o padrão denominado MIME

Exemplos

- **Image/jpg**: transmissão de imagens (jpe, jpg, jpeg, ...)
- **text/html**: transmissão de textos em HTML
- **x-application/java**: transmissão de classes java (.class)
- **application/json; charset=UTF-8**: dados em formato JSON com codificação UTF-8



Padrões de Documentação de APIs

SOAP

- [WSDL – Web Services Description Language](#)



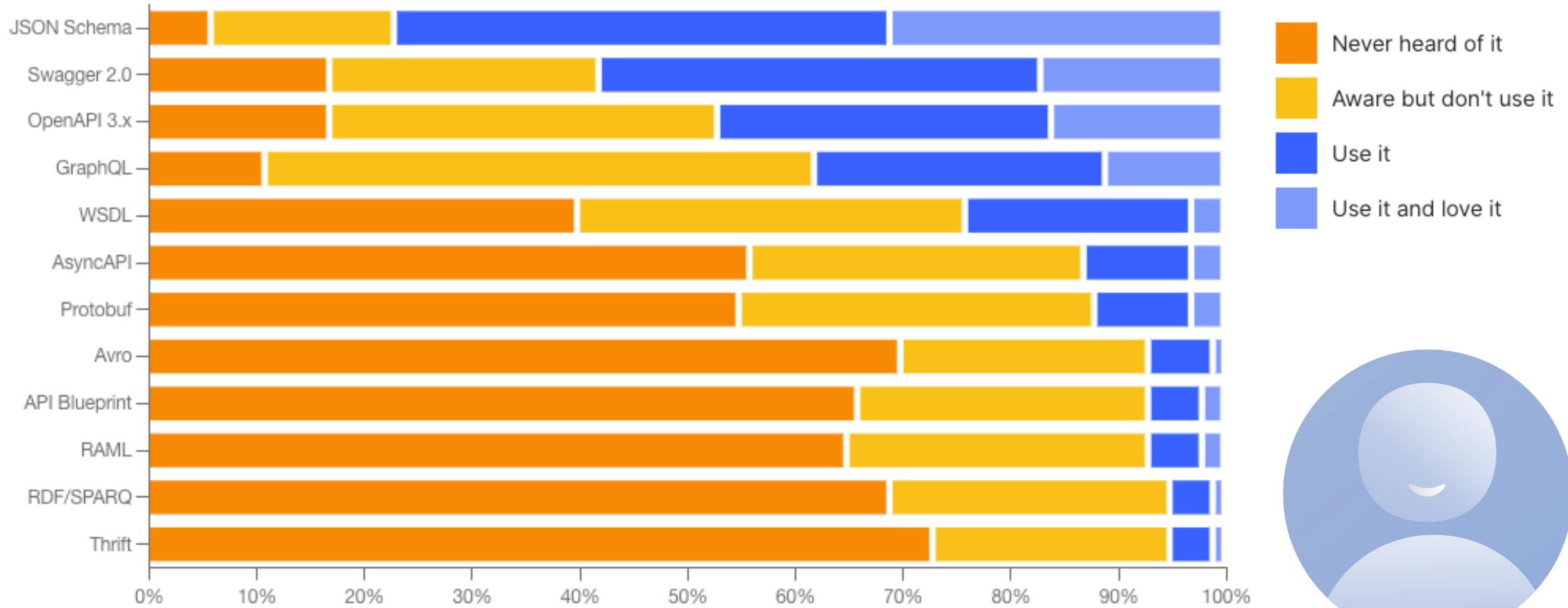
REST

- [OpenAPI Initiative](#)
- [Swagger](#)
- [API Blueprint](#)
- [RAML - RESTful API Modeling Language](#) – SalesForce
- [WADL – Web Application Description Language](#)



Especificações de APIs mais utilizadas

Postman 2023 State of the API Report



Fonte: [2023 State of the API Report | API Technologies](#)

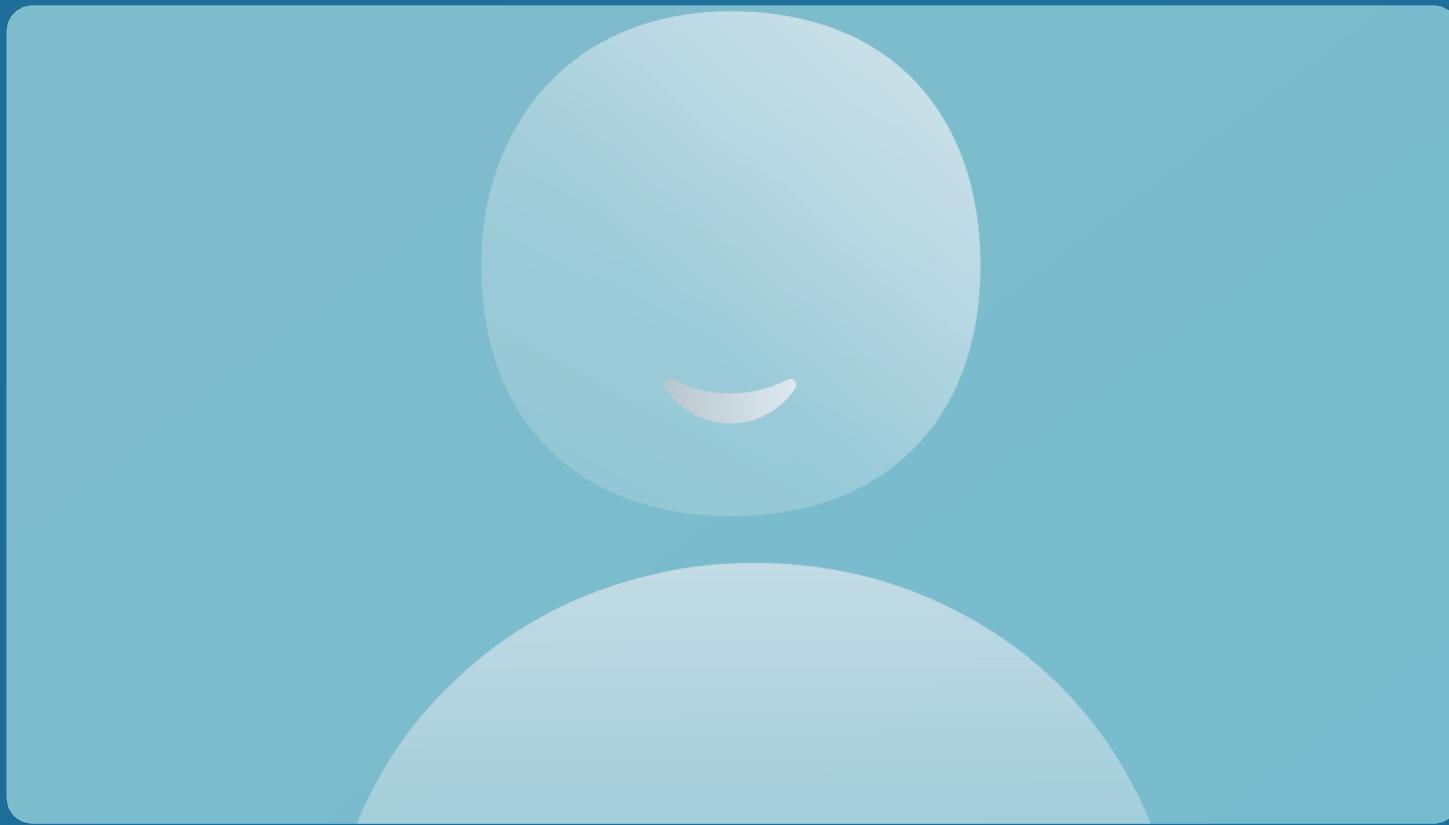


Resumo

- Visão geral de aspectos arquiteturais envolvidos na criação de uma API.
- Comparativo entre os principais estilos arquiteturais de APIs (SOAP, REST e GraphQL).
- Diferentes alternativas para a representação de dados trocados via APIs.
- Possibilidades para a documentação de APIs.



JavaScript Object Notation (JSON)



Estruturas de Dados com JSON

FOLHAJUS

Órgão do Ministério Público demite procurador da Lava Jato por outdoor em Curitiba

→ Procuradores fazem investida na reta final contra PEC do CNMP

MÔNICA B

Ex-ministros querem d
Bolsonaro atacou políti
que incentivavam pede

Presidente atacou políticas de e
Direitos Humanos, que agora v

Fonte: [Folha de S.Paulo: Notícias, Imagens, Vídeos e Entrevistas \(uol.com.br\)](https://www.folha.uol.com.br)



ELEIÇÕES 2022

Doria busca brechas nas regras de prévias do PSDB ante pressão de Eduardo Leite

→ PAINEL Doria e Leite disputam apoio de Mara Gabrilli nas prévias do PSDB

ÁSIA

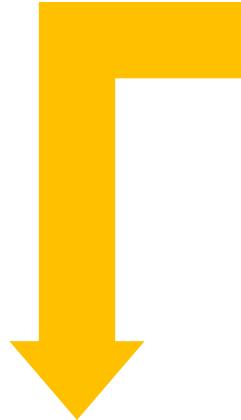
Brasil deve ser um dos principais prejudicados por crescimento mais lento da China

→ VAIVÉM Sem China, queda de preço da carne chega ao consumidor

titulo	resumo	data	autor	fonte	texto
Órgão do Ministério Público demite procurador da Lava Jato por outdoor em Curitiba	Procuradores fazem investida na reta final contra PEC do CNMP	18/10/2021	Italo Nogueira	FOLHAJUS	O CNMP (Conselho Nacional do Ministério Público) decidiu nesta segunda-feira (18) aplicar pena de demissão
Doria busca brechas nas regras de prévias do PSDB ante pressão de Eduardo Leite	Doria e Leite disputam apoio de Mara Gabrilli nas prévias do PSDB	18/10/2021	Carolina Linhares	ELEIÇÕES 2022	A praticamente um mês da votação das prévias presidenciais do PSDB e na véspera do primeiro debate



Estruturas de Dados com JSON



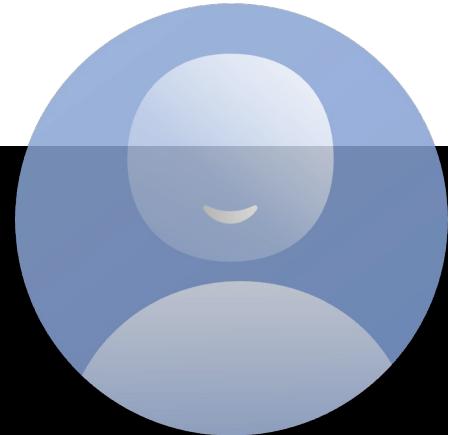
titulo	resumo	data	autor	fonte	texto
Órgão do Ministério Público demite procurador da LavaJato por outdoor em Curitiba	Procuradores fazem investida na reta final contra PEC do CNMP	18/10/2021	Italo Nogueira	FOLHAJUS	O CNMP (Conselho Nacional do Ministério Público) decidiu nesta segunda-feira (18) aplicar pena de demissão
Doria busca brechas nas regras de prévias do PSDB ante pressão de Eduardo Leite	Doria e Leite disputam apoio de Mara Gabrilli nas prévias do PSDB	18/10/2021	Carolina Linhares	ELEIÇÕES 2022	A praticamente um mês da votação das prévias presidenciais do PSDB e na véspera do primeiro debate

```
{  
  "noticias": [  
    {  
      "titulo": "Órgão do Ministério Público ... ",  
      "resumo": "Procuradores fazem investida ...",  
      "data": "18.out.2021 às 20h20",  
      "autor": "Italo Nogueira",  
      "fonte": "FOLHAJUS",  
      "texto": "O CNMP (Conselho Nacional ...."  
    }  
  ]  
}
```

DADOS em JSON

- Permite estruturar dados diversos
- Simples de ser criado e lido
- Compatível com diversas plataformas
- Oferece bom desempenho na leitura

Estruturas de Dados com JSON



```
{  
  "noticias": [  
    {  
      "titulo": "Órgão do Ministério Público demite procurador da Lava Jato por outdoor em Curitiba",  
      "resumo": "Procuradores fazem investida na reta final contra PEC do CNMP",  
      "data": "18.out.2021 às 20h20",  
      "autor": "Italo Nogueira",  
      "fonte": "FOLHAJUS",  
      "texto": "O CNMP (Conselho Nacional do Ministério Público) decidiu nesta segunda-feira (18) aplicar pena de demissão ao  
      procurador Diogo Castor de Mattos, membro da antiga força-tarefa da Lava Jato em Curitiba, pela contratação  
      de um outdoor em homenagem à operação."  
    },  
    {  
      "titulo": "Doria busca brechas nas regras de prévias do PSDB ante pressão de Eduardo Leite",  
      "resumo": "Doria e Leite disputam apoio de Mara Gabrilli nas prévias do PSDB",  
      "data": "18.out.2021 às 21h00",  
      "autor": "Carolina Linhares",  
      "fonte": "ELEIÇÕES 2022",  
      "texto": "A praticamente um mês da votação das prévias presidenciais do PSDB e na véspera do primeiro debate entre  
      tucanos, aliados de João Doria questionam regras do partido e buscam brechas para reverter o cenário  
      desfavorável ao governador paulista, que vê seu rival Eduardo Leite ganhar corpo na disputa."  
    }  
  ]  
}
```

Fonte: [Folha de S.Paulo: Notícias, Imagens, Vídeos e Entrevistas \(uol.com.br\)](https://www.uol.com.br/folha/noticias/)



Estruturas de Dados com JSON

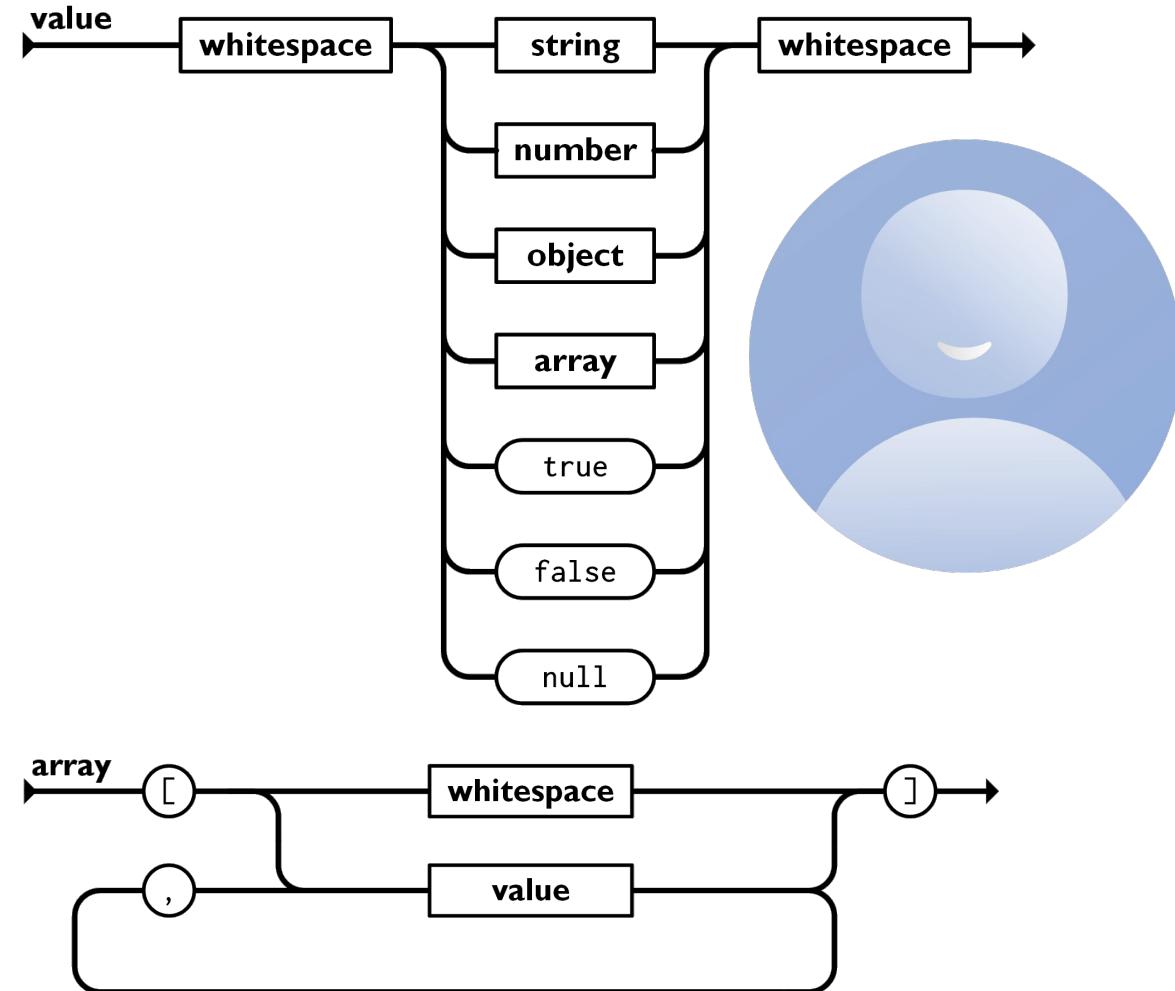
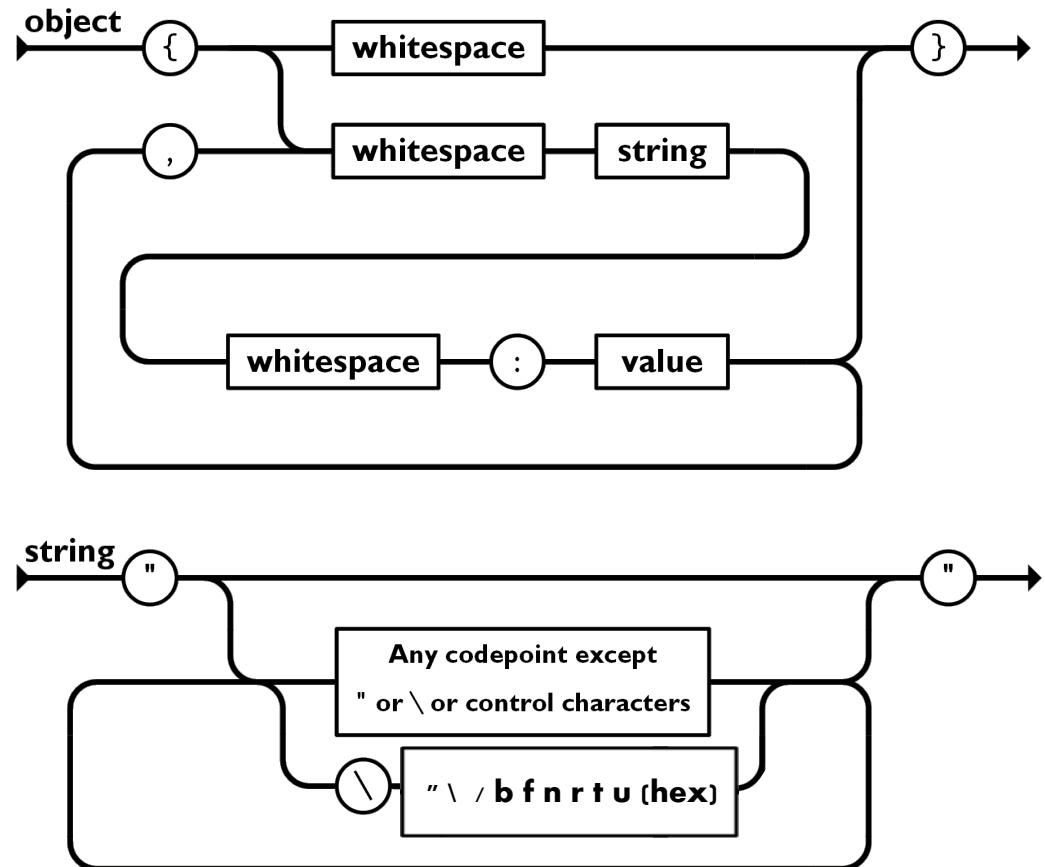
Tipos de Dados

- Dados numéricos: inteiro, real
- Dados textuais: caracteres, strings
- Dados booleanos: true e false
- Dados Compostos
 - Datas
 - Arrays
 - Objetos,
 -



```
{  
    "id": 45,  
    "titulo": "Órgão do Ministério Público ... ",  
    "resumo": "Procuradores fazem investiga ... ",  
    "data": "18.out.2021 às 20h20",  
    "autor": "Italo Nogueira",  
    "fonte": "FOLHAJUS",  
    "texto": "O CNMP (Conselho Nacional ....",  
    "curtidas": 258,  
    "publicado": true  
}
```

Estrutura do Formato JSON



Fonte: [Introducing JSON](https://json.org) (json.org)



81



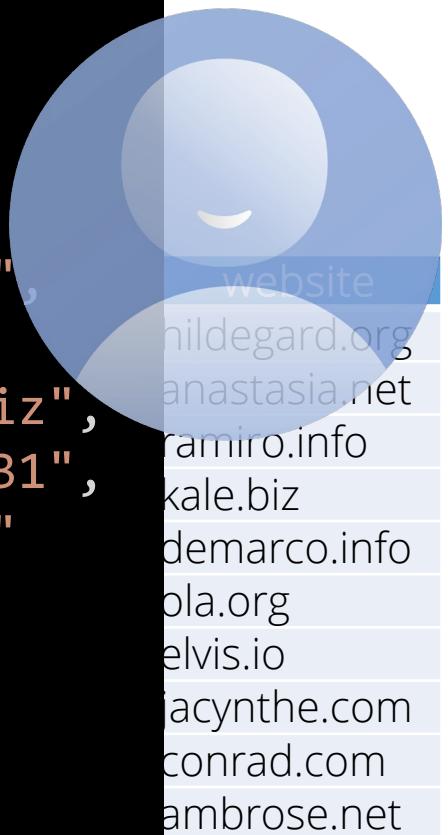
Prof. Rommel Vieira Carneiro

JSON vs Tabelas

id	nome	cidade
1	Leanne Graham	Belo Horizonte
2	Ervin Howell	Betim
3	Clementine Bauch	Rio de Janeiro
4	Patricia Lebsack	Betim
5	Chelsey Dietrich	São Paulo
6	Mrs. Dennis Schulist	Rio de Janeiro
7	Kurtis Weissnat	Belo Horizonte
8	Nicholas Runolfsdottir V	Belo Horizonte
9	Glenna Reichert	Betim
10	Clementina DuBuque	São Paulo



```
{  
  "contatos": [  
    {  
      "id": 1,  
      "nome": "Leanne Graham",  
      "cidade": "Belo Horizonte",  
      "categoria": "amigos",  
      "email": "Sincere@april.biz",  
      "telefone": "1-770-736-8031",  
      "website": "hildegard.org"  
    },  
    {  
      "id": 2,  
      "nome": "Ervin Howell",  
      "cidade": "Betim",  
      "categoria": "familia",  
      "email": "Shanna@melissa.tv",  
      "telefone": "010-692-6593",  
      "website": "anastasia.net"  
    },  
    ...  
  ]  
}
```



JSON vs Tabelas

```
{  
  "id": 901,  
  "name": {  
    "first": "John", "middle": "K", "last": "Doe"  
  },  
  "phones": [  
    { "type": "home", "number": "555-3762" },  
    { "type": "work", "number": "555-7242" }  
  ],  
  "lazy": false,  
  "married": null  
}
```

Exemplo de JSON

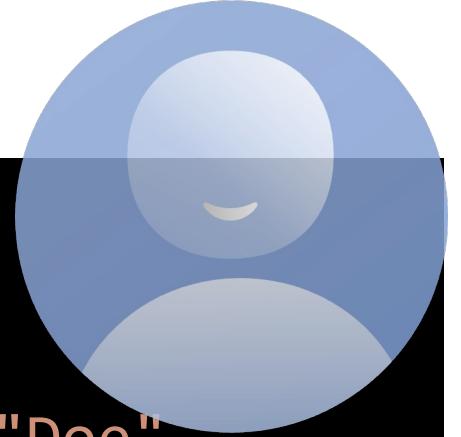
Fonte: adaptado de [JSON Format Example: JSON_TABLE function – IBM Developer](#)



83



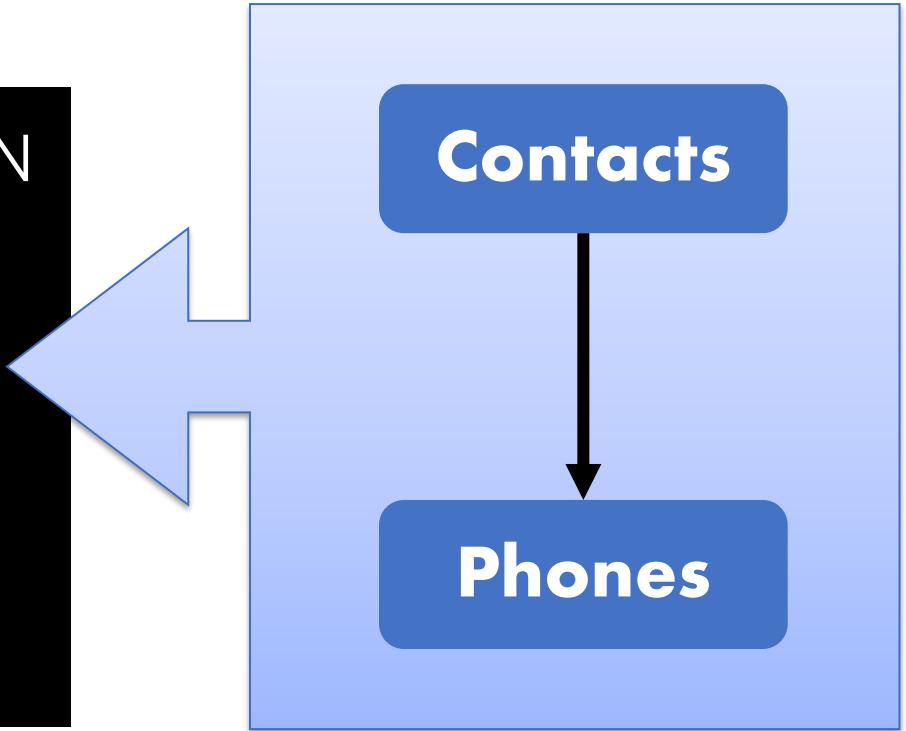
Prof. Rommel Vieira Carneiro



JSON vs Tabelas

```
{  
    "id": 901,  
    "name": {  
        "first": "John", "middle": "K", "last": "Doe"  
    },  
    "phones": [  
        { "type": "home", "number": "555-3762" },  
        { "type": "work", "number": "555-7242" }  
    ],  
    "lazy": false,  
    "married": null  
}
```

Exemplo de JSON



Fonte: adaptado de [JSON Format Example: JSON_TABLE function – IBM Developer](#)

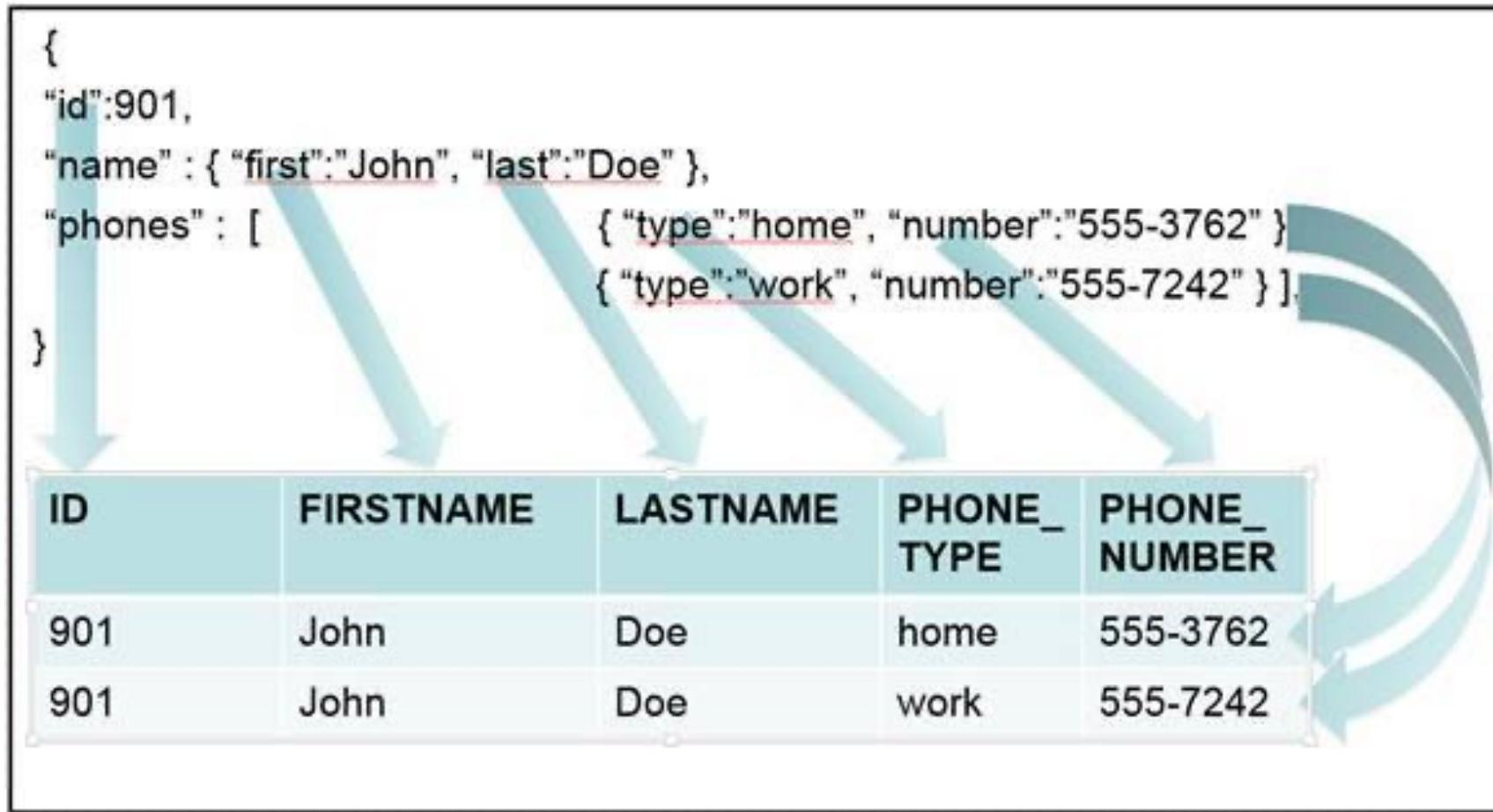


84



Prof. Rommel Vieira Carneiro

JSON vs Tabelas



Fonte: adaptado de [JSON Format Example: JSON_TABLE function – IBM Developer](#)

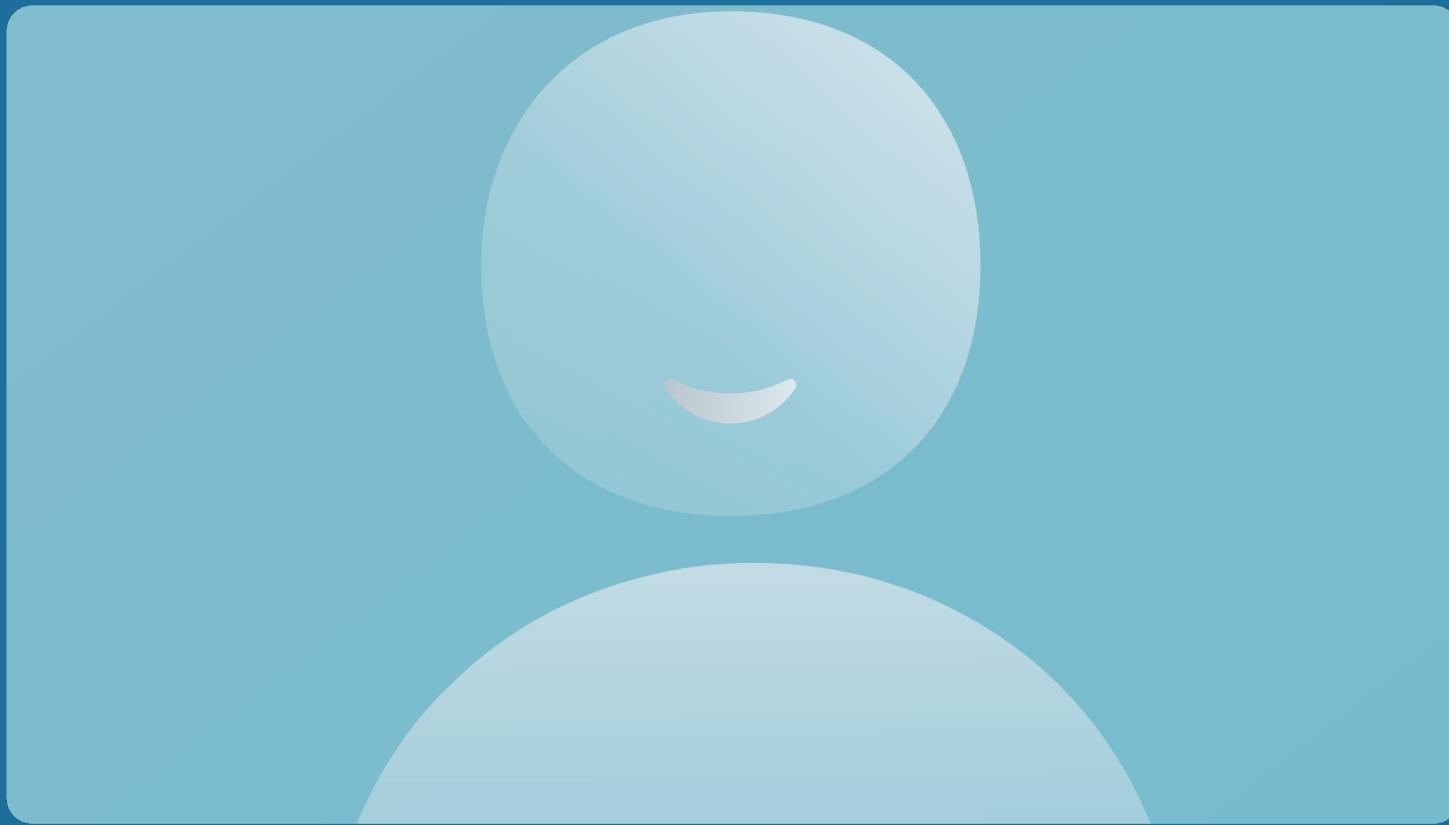


Resumo

- Como pensar as estruturas de dados em aplicações no formato JSON.
- Tipos de dados e a estrutura de um documento JSON.
- Um comparativo do JSON e Tabelas na descrição de dados.



JSON Schema



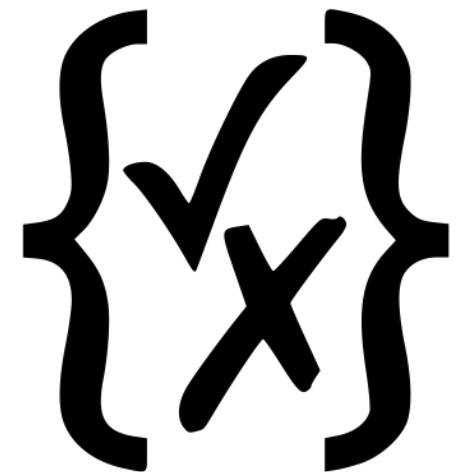
JSON Schema

JSON Schema é um vocabulário que permite descrever, referenciar e validar documentos JSON.



Benefícios

- Descreve formatos de dados
- Provê uma documentação legível para seres humanos
- Permite validação de dados



Aplicações

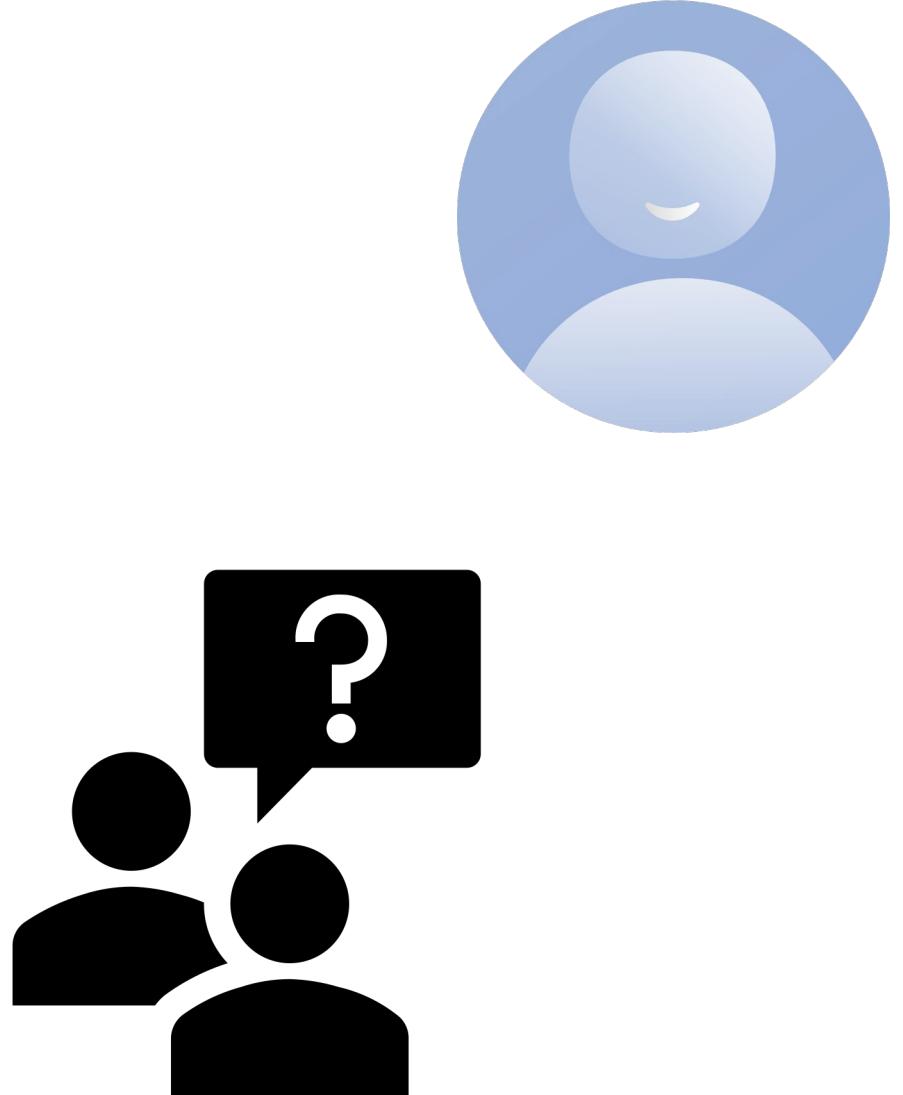
- Automação de testes
- Garantir qualidade de dados submetidas por clientes

JSON Schema

```
{  
  "productId": 1,  
  "productName": "Porta verde",  
  "price": 12.50,  
  "tags": [ "casa", "verde" ]  
}
```

DÚVIDAS COMUNS

- Esses dados referem-se a que?
- O que é productId?
- O preço pode ser zero (0)?
- Todas as tags são string?
- A informação productName é requerida?





JSON Schema

Dados em JSON

```
{  
  "productId": 1,  
  "productName": "Porta verde",  
  "price": 12.50,  
  "tags": [ "casa", "verde" ]  
}
```

Esquema dos Dados

```
{  
  "$schema": "https://json-schema.org/draft/2020-12/schema",  
  "$id": "https://example.com/product.schema.json",  
  "title": "Produto",  
  "description": "Um produto do catálogo",  
  "type": "object",  
}
```

DÚVIDAS COMUNS

- Esses dados referem-se a que?

Versão do esquema, URI do esquema, anotações e tipificação



JSON Schema

Dados em JSON

```
{  
  "productId": 1,  
  "productName": "Porta verde",  
  "price": 12.50,  
  "tags": [ "casa", "verde" ]  
}
```

DÚVIDAS COMUNS

- Esses dados referem-se a que?
- O que é productId?
- O preço pode ser zero (0)?
- Todas as tags são string?

Esquema dos Dados

```
{ ... // Definição de esquema e tipo do dado  
  "properties": {  
    "productId": {  
      "description": "Identificador único",  
      "type": "integer" },  
    "productName": {  
      "description": "Nome do produto",  
      "type": "string" },  
    "price": {  
      "description": "Preço do produto",  
      "type": "number", "exclusiveMinimum": 0 },  
    "tags": {  
      "description": "Rótulos para o produto",  
      "type": "array", "items": { "type": "string" },  
      "minItems": 1, "uniqueItems": true },  
  },  
}
```

Propriedades do objeto definido



JSON Schema

Dados em JSON

```
{  
  "productId": 1,  
  "productName": "Porta verde",  
  "price": 12.50,  
  "tags": [ "casa", "verde" ]  
}
```

DÚVIDAS COMUNS

- Esses dados referem-se a que?
- O que é productId?
- O preço pode ser zero (0)?
- Todas as tags são string?
- A informação productName é requerida?

Esquema dos Dados

```
{  
  ... // Definição de esquema e tipo do dado  
  ... // Definição das propriedades  
  "required": [ "productId", "productName" ]  
}
```

Definição das propriedades requeridas



JSON Schema

```
{  
  "$schema": "https://json-schema.org/draft/2020-12/schema",  
  "$id": "https://example.com/product.schema.json",  
  "title": "Produto",  
  "description": "Um produto do catálogo",  
  "type": "object",  
  
  "properties": {  
    "productId": { "description": "Identificador único", "type": "integer" },  
    "productName": { "description": "Nome do produto", "type": "string" },  
    "price": { "description": "Preço do produto", "type": "number", "exclusiveMinimum": 0 },  
    "tags": { "description": "Rótulos para o produto", "type": "array",  
              "items": { "type": "string" }, "minItems": 1, "uniqueItems": true },  
  },  
  
  "required": [ "productId", "productName" ]  
}
```

```
{  
  "productId": 1,  
  "productName": "A green door",  
  "price": 12.50,  
  "tags": [ "home", "green" ]  
}
```

Código Completo

JSON Schema - Palavras reservadas



Palavra Reservada	Significado
\$schema	Indica o dialeto (Versão) utilizado para descrever o esquema. Possui uma URI com o endereço do dialeto.
\$id	Indica o base URI para esta definição de esquema na Internet. Possui uma URI sem fragmentos com o endereço da definição do esquema.
\$ref	Indica um esquema definido em outro local.
\$defs	Indica um local para definição de esquemas e sub-esquemas JSON reutilizáveis em um esquema mais geral

IMPORTANTE: o caractere # pode ser utilizado para fazer referência à raiz do documento e com isso referenciar outras partes do documento. Ex: #/\$defs/subesquema



JSON Schema - Palavra reservada \$ref

```
{  
  "$id": "https://example.com/schemas/address",  
  
  "type": "object",  
  "properties": {  
    "street_address": { "type": "string" },  
    "city": { "type": "string" },  
    "state": { "type": "string" }  
  },  
  "required": ["street_address", "city",  
              "state"]  
}
```



```
{  
  "$id": "https://example.com/schemas/customer",  
  
  "type": "object",  
  "properties": {  
    "first_name": { "type": "string" },  
    "last_name": { "type": "string" },  
    "shipping_address": { "$ref": "/schemas/address" },  
    "billing_address": { "$ref": "/schemas/address" }  
  },  
  "required": ["first_name", "last_name"]  
}
```

Fonte: [Structuring a complex schema — Understanding JSON Schema 2020-12 documentation](#)



95



Prof. Rommel Vieira Carneiro

JSON Schema - Palavra reservada \$defs



```
{  
  "$id": "https://example.com/schemas/customer",  
  
  "type": "object",  
  "properties": {  
    "first_name": { "$ref": "#/$defs/name" },  
    "last_name": { "$ref": "#/$defs/name" },  
    "shipping_address": { "$ref": "/schemas/address" },  
    "billing_address": { "$ref": "/schemas/address" }  
  },  
  "required": ["first_name", "last_name", "shipping_address", "billing_address"],  
  
  "$defs": {  
    "name": { "type": "string" }  

```

Fonte: [Structuring a complex schema — Understanding JSON Schema 2020-12 documentation](#)



96



Prof. Rommel Vieira Carneiro

JSON Schema - Ferramentas - Validação

JSON Schema Validator

An online, interactive JSON Schema validator. Supports JSON Schema Draft 3, Draft 4, Draft 6, Draft 7 and Draft 2019-09.

Select schema: Custom

```
1 {  
2   "$schema": "https://json-schema.org/draft/2020-  
3   12/schema",  
4   "$id":  
5   "https://example.com/product.schema.json",  
6   "title": "Product",  
7   "description": "A product from Acme's catalog",  
8   "type": "object",  
9   "properties": {  
10    "productId": {  
11      "description": "The unique identifier for a  
12      product",  
13      "type": "integer"  
14    },  
15    "productName": {  
16      "description": "Name of the product",  
17      "type": "string"  
18    }  
19  },  
20  "required": [ "productId", "productName" ]  
}
```

Input JSON: ✖ Found 1 error(s)

```
1 {  
2   "productId": 1,  
3   "price": 12.50,  
4   "tags": [ "home", "green" ]  
5 }  
6
```

JSON Schema Validator

An online, interactive JSON Schema validator. Supports JSON Schema Draft 3, Draft 4, Draft 6, Draft 7 and Draft 2019-09.

Select schema: Custom

```
1 {  
2   "$schema": "https://json-schema.org/draft/2020-  
3   12/schema",  
4   "$id":  
5   "https://example.com/product.schema.json",  
6   "title": "Product",  
7   "description": "A product from Acme's catalog",  
8   "type": "object",  
9   "properties": {  
10    "productId": {  
11      "description": "The unique identifier for a  
12      product",  
13      "type": "integer"  
14    },  
15    "productName": {  
16      "description": "Name of the product",  
17      "type": "string"  
18    }  
19  },  
20  "required": [ "productId", "productName" ]  
}
```

Input JSON:

```
1 {  
2   "productId": 1,  
3   "productName": "A green door",  
4   "price": 12.50,  
5   "tags": [ "home", "green" ]  
6 }  
7
```

✓ No errors found. JSON validates against the schema



JSON Schema - Ferramentas - Gerador de Esquema

JSONschema.net

Please use GitHub to [report bugs](#)

Github

JSON

Settings

Your JSON is valid ✓

```
1 {  
2     "productId": 1,  
3     "productName": "An ice sculpture",  
4     "price": 12.50,  
5     "tags": ["cold", "ice"],  
6     "dimensions": {  
7         "length": 7.0,  
8         "width": 12.0,  
9         "height": 9.5  
10    },  
11    "warehouseLocation": {  
12        "latitude": -78.75,  
13        "longitude": 20.4  
14    }  
15}
```

RESET

SUBMIT

JSON Schema

Code

Graph

Edit

COPY TO CLIPBOARD

```
1 {  
2     "$schema": "http://json-schema.org/draft-07/schema#",  
3     "$id": "http://example.com/example.json",  
4     "type": "object",  
5     "title": "The root schema",  
6     "description": "The root schema comprises the entire JSON document.",  
7     "default": {},  
8     "examples": [  
9         {  
10            "productId": 1,  
11            "productName": "An ice sculpture",  
12            "price": 12.5,  
13            "tags": [  
14                "cold",  
15                "ice"  
16            ],  
17            "dimensions": {  
18                "length": 7.0,  
19                "width": 12.0,  
20                "height": 9.5  
21            },  
22            "warehouseLocation": {  
23                "latitude": -78.75,  
24                "longitude": 20.4  
25            }  
26        }  
27    ]  
28}
```

half a minute ago

JSON to JSON Schema Converter



JSON Schema - Ferramentas

The screenshot shows the liquid JSON Schema tool interface. At the top, there's a header with the 'liquid' logo and a menu icon. Below it, a title 'Sample JSON Document' is followed by a code editor containing the following JSON:

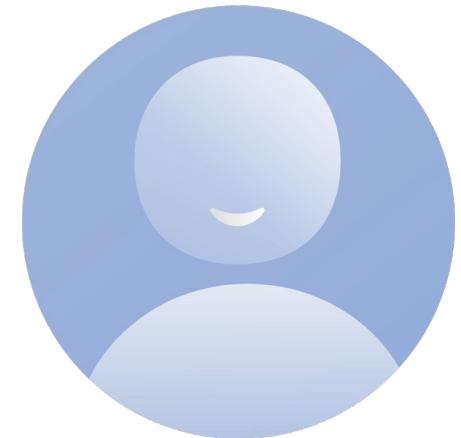
```
1 {  
2   "productId": 1,  
3   "productName": "An ice sculpture",  
4   "price": 12.50,  
5   "tags": [ "cold", "ice" ]  
6 }
```

Below the code editor is an 'Options' dropdown and a 'Generate Schema' button. The bottom section is titled 'Inferred JSON Schema' and displays the generated schema:

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "type": "object",  
  "properties": {  
    "productId": {  
      "type": "integer"  
    },  
    "productName": {  
      "type": "string"  
    },  
    "price": {  
      "type": "number"  
    },  
    "tags": {  
      "type": "array",  
      "items": {  
        "type": "string"  
      }  
    }  
  }  
}
```

Online Tools

- [**JSON Validator**](#)
- [**JSON to JSON Schema**](#)

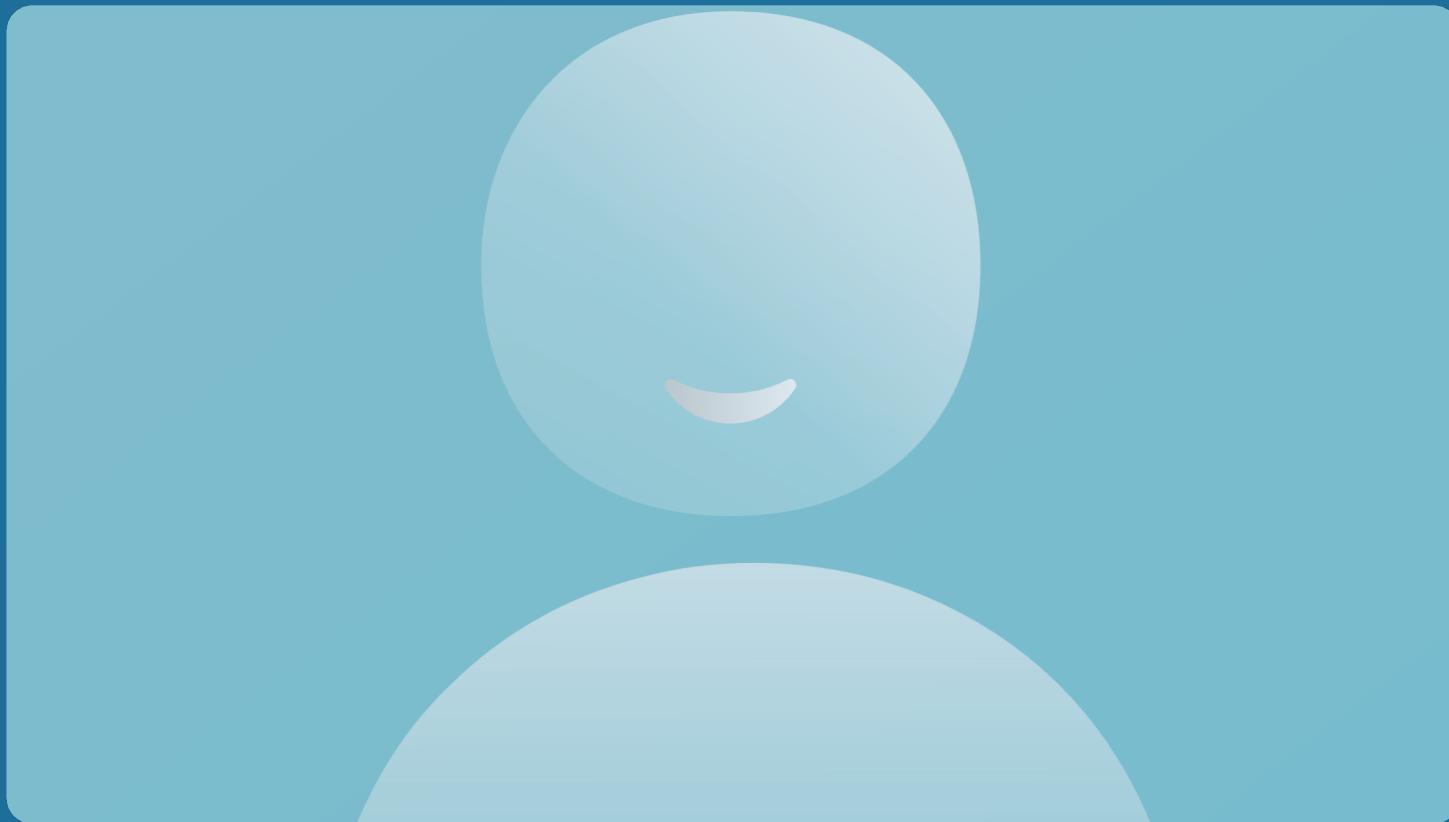


Resumo

- O padrão JSON Schema como alternativa para descrever e validar documentos JSON.
- As necessidades atendidas e a estrutura do padrão JSON Schema .
- As ferramentas para trabalhar com as especificações de documentos baseadas em JSON Schema.



SOAP API

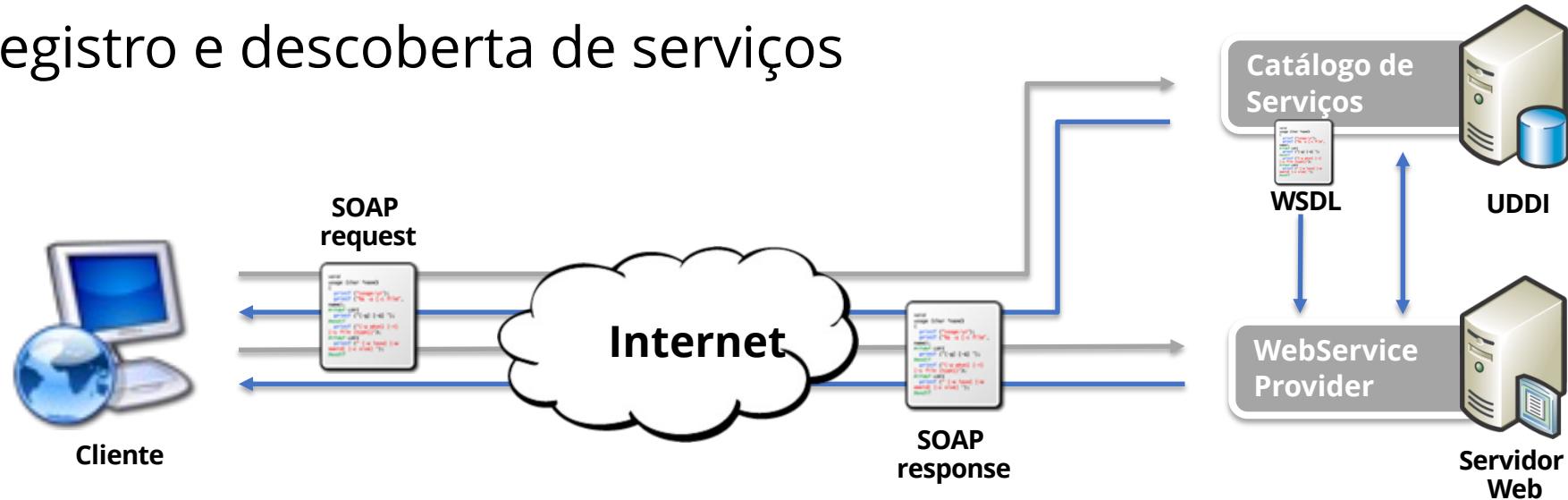


Web Services – SOAP



Arquitetura

- SOAP – Protocolo de troca de dados baseado em XML
- XML – Linguagem de marcação para descrição dos dados
- WSDL – Descritor de serviços baseado em XML
- UDDI – Registro e descoberta de serviços



Web Services – SOAP

- Simple Object Access Protocol (SOAP)
- Protocolo de troca de dados baseado em XML para envio e recebimento de mensagens na Internet
- Independente de plataforma ou linguagem de programação



Web Services – SOAP

Estrutura da Mensagem SOAP

- Envelope
 - Define o início e o final da mensagem
 - É obrigatório
- Header (Cabeçalho)
 - Traz atributos opcionais da mensagem utilizada para o seu processamento
 - É opcional
- Body (Corpo)
 - Possui o conteúdo da mensagem em formato XML
 - É obrigatório



Web Services – SOAP

Estrutura da Mensagem SOAP – Requisição

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn
```

```
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle=
        "http://www.w3.org/2001/12/soap-encoding">
    <soap:Header>
        <t:Transaction xmlns:t="URI"
            soap:mustUnderstand="1" > 5 </t:Transaction>
    </soap:Header>
    <soap:Body xmlns:m="http://www.example.org/stock">
        <m:GetStockPrice>
            <m:StockName>IBM</m:StockName>
        </m:GetStockPrice>
    </soap:Body>
</soap:Envelope>
```

SOAP Envelope

SOAP Header

SOAP Body

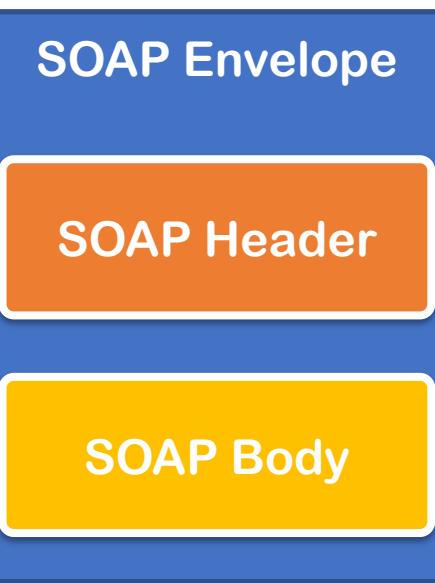


Web Services – SOAP

Estrutura da Mensagem SOAP – Resposta

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn
```

```
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle=
        "http://www.w3.org/2001/12/soap-encoding">
    <soap:Header>
        <t:Transaction xmlns:t="URI"
            soap:mustUnderstand="1" >5</t:Transaction>
    </soap:Header>
    <soap:Body xmlns:m="http://www.example.org/stock">
        <m:GetStockPriceResponse>
            <m:Price>34.5</m:Price>
        </m:GetStockPriceResponse>
    </soap:Body>
</soap:Envelope>
```

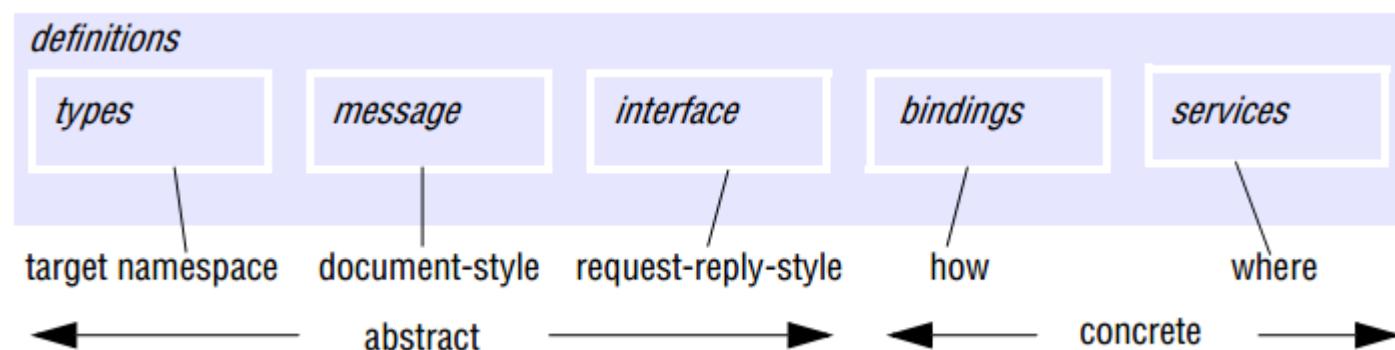


Web Services – SOAP



Web Services Description Language (WSDL)

- Linguagem de descrição de um web service baseada em XML
- Define os seguintes objetos:
 - **Tipos de dados** suportados pelo serviço
 - Padrão de **mensagens** de entrada e saída
 - Protocolos de comunicação permitidos pelo Web Service (**binding**)
 - Serviços e portas de comunicação onde **operações** são disponibilizadas pelo Web Service
 - **Definições** sobre schemas e namespaces utilizados no contexto do Web Service



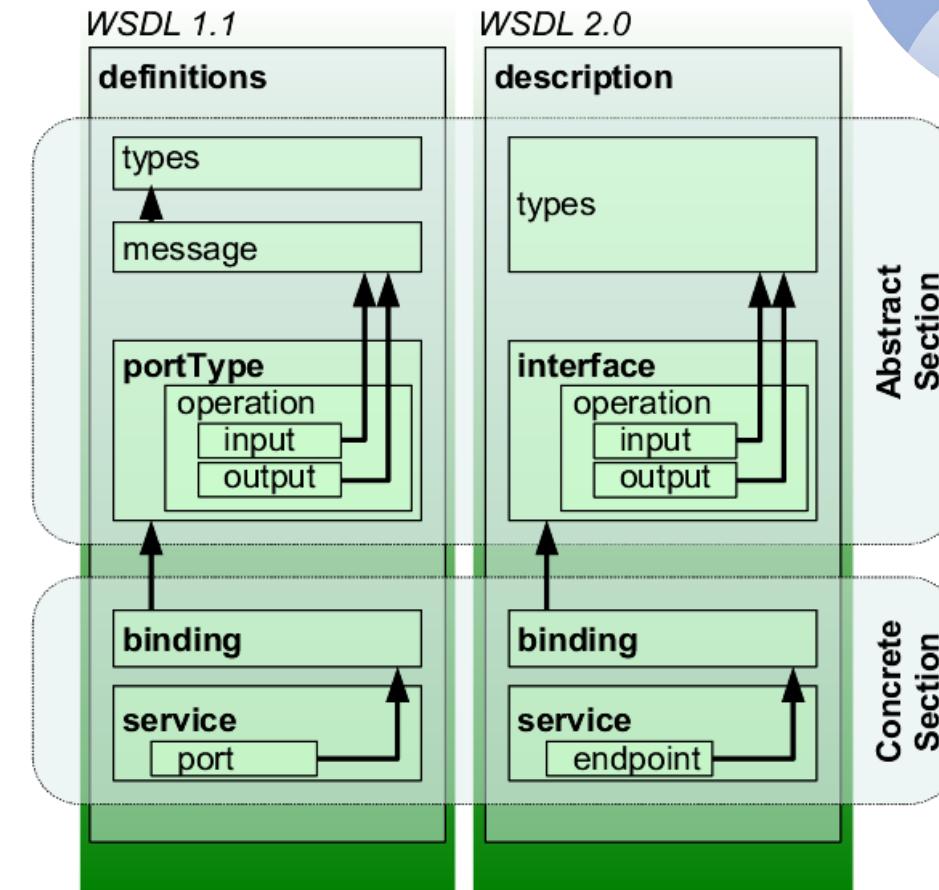
Web Services – SOAP

Web Services Description Language (WSDL) - Exemplo

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>
```



Web Services – SOAP

Universal Description and Discovery Interface (UDDI)

- Serviço de diretório que mantém referência para os Web Services registrados
- Funcionam como páginas amarelas para localização dos Web Services
- Reúne informações como:
 - Nomes endereços e números de telefones dos fornecedores de serviços
 - Serviços oferecidos por cada fornecedor, bem como informações técnicas sobre a interface de cada um



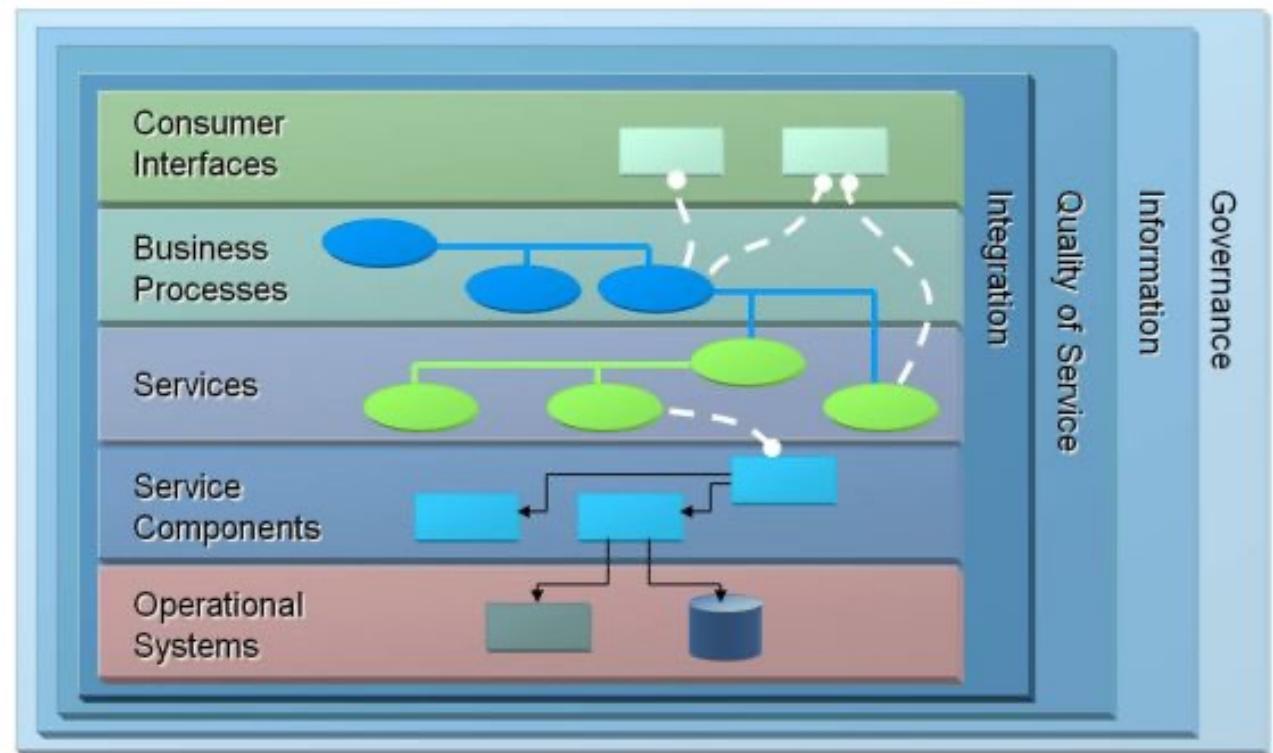
Web Services – Service Oriented Architecture (SOA)

Service Oriented Architecture (SOA)

Estilo de arquitetura de software cujo princípio fundamental está baseado em funcionalidades implementadas por aplicação e disponibilizadas na forma de serviços.

Camadas

- Interface de Usuário
- Processos de Negócios
- Serviços
- Componentes de Serviços
- Sistemas Operacionais

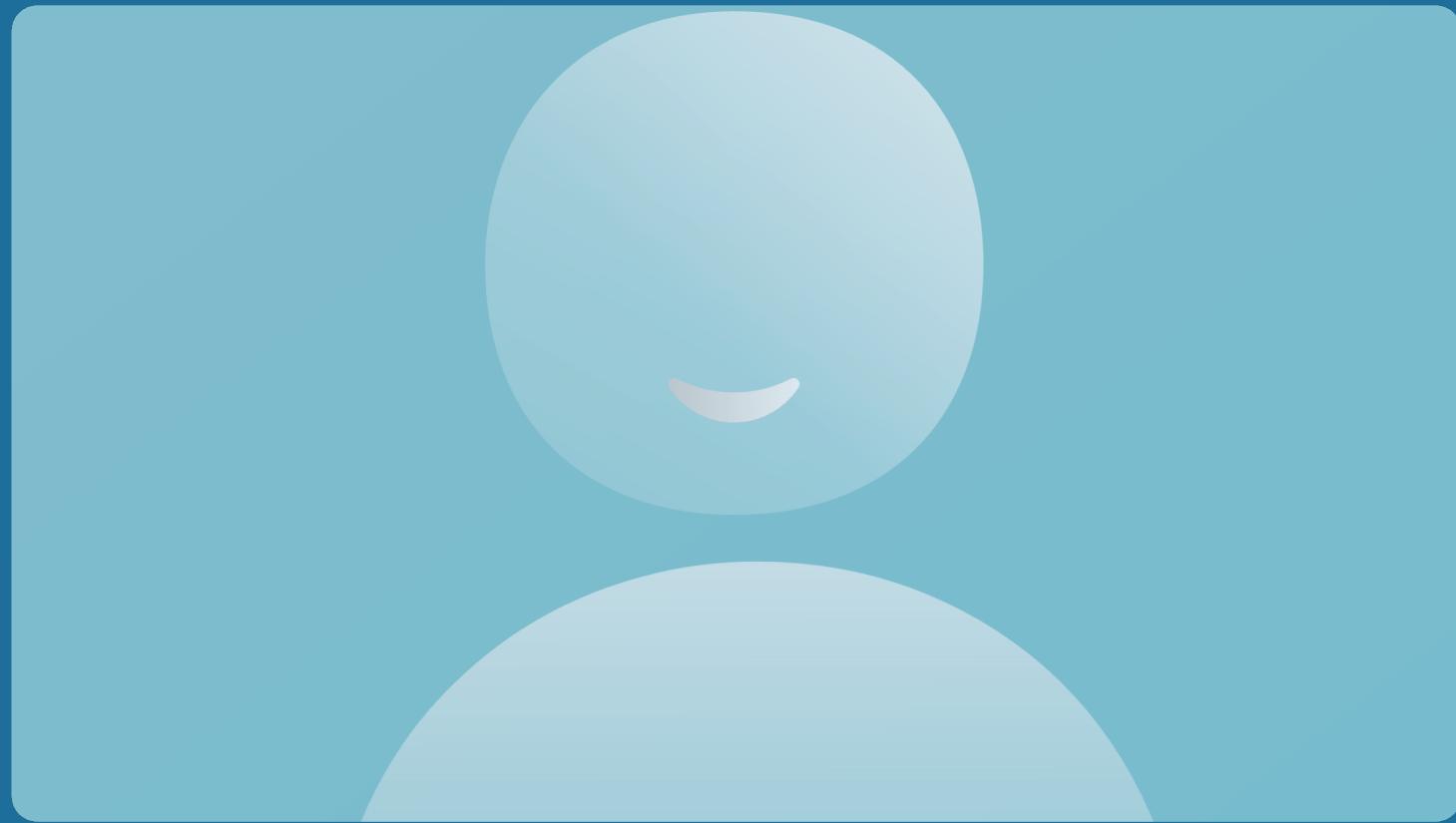


Resumo

- Conceitos, componentes e exemplos das estruturas de mensagens usados em aplicações com APIs baseadas no padrão SOAP.
- O WSDL para documentação de APIs baseadas no padrão SOAP.
- Detalhes da Arquitetura Orientada a Serviços (SOA) e os impactos na estrutura de aplicações.



REpresentational State Transfer (REST)



REST API - Introdução

REST é um estilo arquitetural para construção de serviços Web que significa a Transferência de Estado Representacional (Representational State Transfer).

O termo **RESTful** caracteriza serviços Web que seguem integralmente as recomendações REST, diferentemente daqueles (**RESTlike**) que implementam parcialmente suas recomendações.



Requisitos de APIs REST



Requisito	Descrição
Cliente e Servidor	Cliente e servidor são independentes e não se preocupam com os detalhes de implementação do outro
Stateless	O servidor não mantém estado sobre a sessão do usuário/aplicação e toda requisição deve conter todas as informações requeridas
Cache	As respostas podem ser armazenadas em cache (cliente ou proxies reversos). As respostas devem definir se permitem ou não o cache.
Interface uniforme	A interface entre cliente e servidor deve identificar adequadamente os recursos por meio de uma URI e representações padronizadas.
Sistema em camadas	Os componentes não têm visibilidade além da camada seguinte, podendo interagir com interlocutores intermediários de forma natural

Fonte: [Architectural Styles and the Design of Network-based Software Architectures](#) (Fielding, 2000)



114



Prof. Rommel Vieira Carneiro

REST API – Princípios de design

Princípios importantes de design de uma API RESTful

- Definições do protocolo HTTP
- Estrutura uniforme de Endpoint
- Abordagem stateless (sem estado)
- Abordagem HATEOAS
- Controle do versionamento da API
- Controle adequado do cache
- Documentação clara e adequada da API



Fontes:

- Best Practices for Designing a Pragmatic RESTful API - <https://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api>
- RESTful Web Services: The basics, Alex Rodriguez - <http://www.ibm.com/developerworks/webservices/library/ws-restful/>
- RESTful API Designing guidelines—The best practices - <https://hackernoon.com/restful-api-designing-guidelines-the-best-practices-60e1d954e7c9>
- RESTful API Design. Best Practices in a Nutshell - <https://blog.philippauer.de/restful-api-design-best-practices/>
- API design - <https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design>

REST API – Princípios de design

Definições do protocolo HTTP – Métodos HTTP

Utilizar os métodos HTTP de maneira clara em função do seu propósito semântico:



GET	Recuperar um recurso (SELECT)
POST	Criar um recurso no servidor (INSERT)
PUT PATCH	Alterar um estado de um recurso ou atualizá-lo (UPDATE)
DELETE	Remover um recurso (DELETE)
OPTIONS	Lista as operações de um recurso

IMPORTANTE: Métodos GET e parâmetros de query não devem alterar estado de recursos.

REST API – Princípios de design

Definições do protocolo HTTP – Métodos HTTP

Antes – Criação com GET

```
GET /adduser?name=Robert HTTP/1.1
```

Depois – Criação com POST

```
POST /users HTTP/1.1
```

Host: myserver

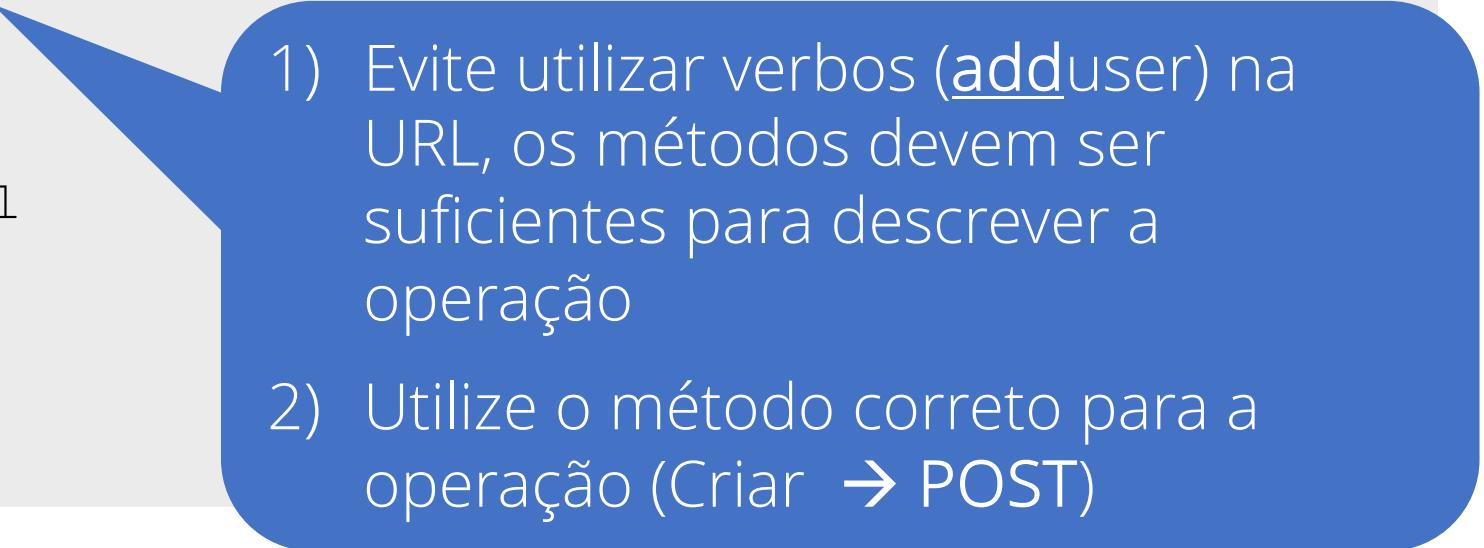
Content-Type: application/xml

```
<?xml version="1.0"?>
```

```
<user>
```

```
    <name>Robert</name>
```

```
</user>
```

- 
- 1) Evite utilizar verbos (adduser) na URL, os métodos devem ser suficientes para descrever a operação
 - 2) Utilize o método correto para a operação (Criar → POST)

REST API – Princípios de design

Definições do protocolo HTTP – Métodos HTTP

Não RESTful		
Verbo	HREF	Ação
POST	/bookmarks/create	Criar (Create)
GET	/bookmarks/show/1	Visualizar (Read)
POST	/bookmarks/update/1	Atualizar (Update)
POST/GET	/bookmarks/delete/1	Apagar (Delete)

RESTful		
Verbo	URI	Ação
POST	/bookmarks	Criar (Create)
GET	/bookmarks/1	Visualizar (Read)
PUT	/bookmarks/1	Atualizar (Update)
DELETE	/bookmarks/1	Apagar (Delete)



REST API – Princípios de design

Definições do protocolo HTTP – Códigos de status HTTP



Code	Propósito	Descrição	Exemplo
1xx	Informacional	Requisição recebida,	Processo em continuidade
		200 – Sucesso na requisição	Requisição de informações (GET) com sucesso
2xx	Sucesso	201 – Recurso Criado	Inclusão de recurso (POST) com sucesso
		202 – Requisição aceita para processamento	Processo assíncrono sem retorno imediato
		204 – Requisição processada com sucesso	Exclusão de recurso (DELETE) com sucesso
3xx	Redirecionamento	301 – Movido permanentemente	Site transferido de servidor
		302 – Movido temporariamente	Recurso temporário no lugar
		400 – Requisição incorreta	Problemas de sintaxe, rotas inexistentes
		401 – Autenticação Requerida	Primeira requisição sem dados de autenticação
4xx	Erro no cliente	403 – Acesso negado	Recurso privado não acessível pelo requisitante
		404 – Recurso não encontrado	Recurso solicitado não existe
		405 – Método não permitido	PUT ou DELETE em endpoints de GET ou POST
5xx	Erro no servidor	O servidor falhou em completar um pedido aparentemente válido	

REST API – Princípios de design

Definições do protocolo HTTP – MIME Types no Content Type

MIME-Type	Content-Type
JSON	application/json
XML	application/xml
XHTML	application/xhtml+xml



```
GET /cliente/2 HTTP/1.1  
Host: http://servidor  
Accept: application/json
```

Requisição

```
HTTP/1.1 200 OK  
Content-Type: application/json; charset=utf-8  
Content-Length: 109  
{  
    id: 2  
    name: 'Rommel Vieira Carneiro',  
    email: 'rommel@email.com',  
    alcohol: '5.21'  
}
```

Resposta

REST API – Princípios de design



Estrutura uniforme de Endpoint

- Uma URI intuitiva direta, auto-documentada, e comprehensível
- Ter uma estrutura hierárquica semelhantes a diretórios

Exemplo

```
http://myservice.org/discussion/{year}/{day}/{month}/{topic}  
http://myservice.org/discussion/2008/12/10/{topic}
```

Orientações adicionais

- Ocultar a extensão da tecnologia empregada (.asp, .jsp, .php, etc)
- Manter tudo em caixa baixa (minúsculo)
- Evitar espaços nos caminhos da URI
- Evite parâmetros de QueryString, o máximo possível

REST API – Princípios de design

Estrutura uniforme de Endpoint – CRUD de Recursos



Ação	Operação	Mapeamento da URL
Incluir um curso	C REATE	POST /cursos/
Obter lista de cursos	R ETRIEVE	GET /cursos
Obter um curso específico	R ETRIEVE	GET /cursos/:id
Pesquisar um curso	R ETRIEVE	GET /cursos?search=param
Alterar um curso	U UPDATE	PUT /cursos/:id PATCH /cursos/:id
Excluir um curso	D ELETE	DELETE /cursos/:id

REST API – Princípios de design

Estrutura uniforme de Endpoint – Relacionamentos



Ação	Mapeamento da URL
Listar alunos do curso	GET /cursos/:id/alunos ou GET /alunos/?curso_id=:id
Obter um aluno do curso	GET /cursos/:id/alunos/:id
Incluir aluno no curso	POST /cursos/:id/alunos
Remover aluno do curso	DELETE /cursos/:id/alunos/:id
Listar cursos do aluno	GET /alunos/:id/cursos ou GET /cursos/?aluno_id=:id
Obter um curso do aluno	GET /alunos/:id/cursos/:id

REST API – Princípios de design



Estrutura uniforme de Endpoint

Exemplo: The Movie DB - Endpoint: <http://api.themoviedb.org/3>

Ação	Mapeamento da URL
Busca por empresas	GET /search/company?query=xxx&page=x
Busca por filmes	GET /search/movie?query=xxx&page=x GET /search/movie?year=XXXX&language=xx
Dados de filme específico	GET /movie/{movie_id}
Artistas e equipe técnica do filme	GET /movie/{movie_id}/credits
Dados de uma empresa específica	GET /company/{company_id}
Dados de uma pessoa específica	GET /person/{person_id}

REST API – Princípios de design



Abordagem stateless (sem estado)

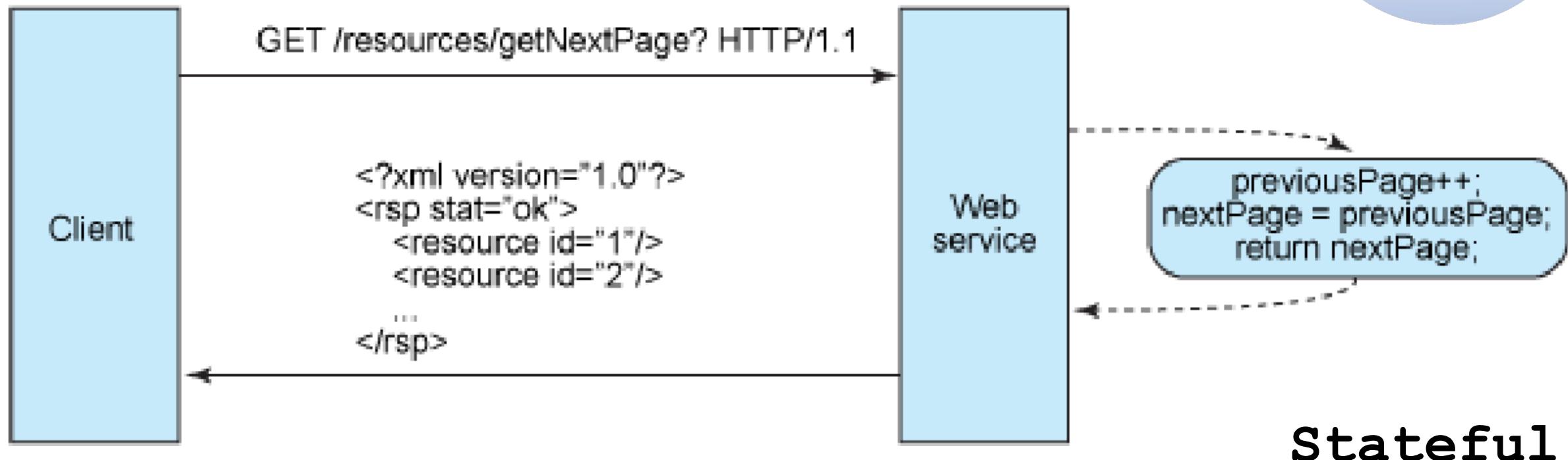
- Servidor não mantém estado sobre sessão do usuário/aplicação
- Toda requisição deve conter todas as informações requeridas (como parte da URL, na query string, no corpo ou no cabeçalho)

Benefícios

- Simplifica o servidor
- Maior escalabilidade uma vez que o servidor não mantém informações sobre sessão
- Servidores de平衡amento de carga não precisam se preocupar com dados de sessão
- Maior confiabilidade (recuperação de falhas)

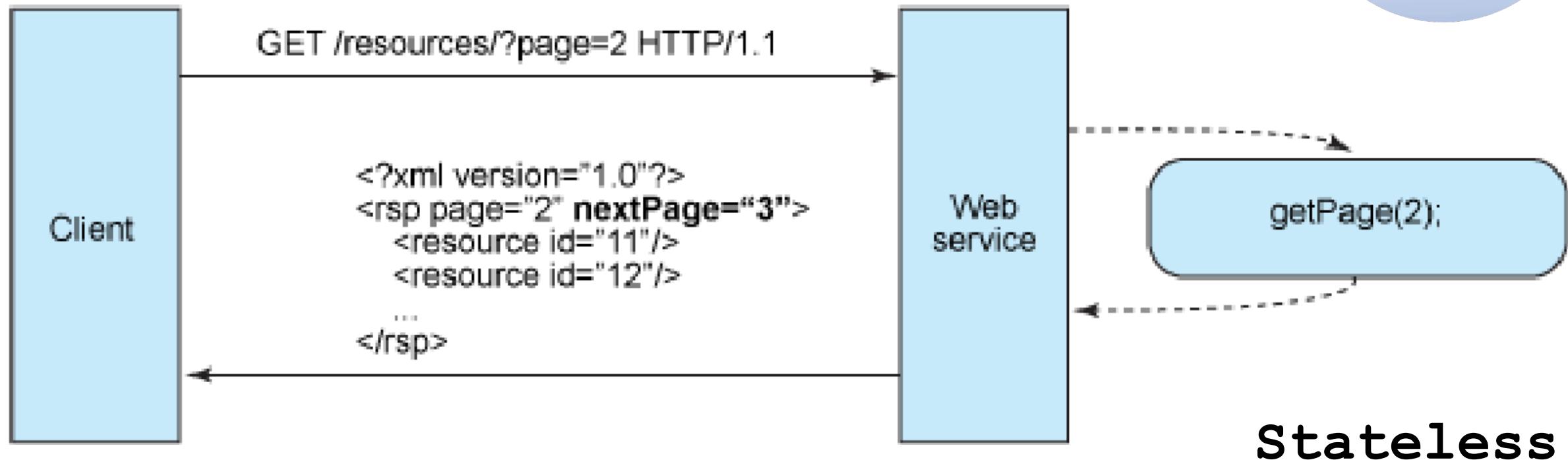
REST API – Princípios de design

Abordagem stateless (sem estado)



REST API – Princípios de design

Abordagem stateless (sem estado)



REST API - Princípios de design

Abordagem HATEOAS - Hipertexto como mecanismo de estado da aplicação

```
GET /account/12345 HTTP/1.1
```

```
-----
```

```
HTTP/1.1 200 OK
```

```
<?xml version="1.0"?>
<account>
  <account_number>12345</account_number>
  <balance currency="usd">100.00</balance>
  <link rel="deposit" href="/account/12345/deposit" />
  <link rel="withdraw" href="/account/12345/withdraw" />
  <link rel="transfer" href="/account/12345/transfer" />
  <link rel="close" href="/account/12345/close" />
</account>
```

Resposta p/ saldo de 100,00



Fonte: <http://en.wikipedia.org/wiki/HATEOAS>

REST API - Princípios de design

Abordagem HATEOAS - Hipertexto como mecanismo de estado da aplicação

```
GET /account/12345 HTTP/1.1
```

```
-----
```

```
HTTP/1.1 200 OK
```

```
<?xml version="1.0"?>
<account>
  <account_number>12345</account_number>
  <balance currency="usd">-25.00</balance>
  <link rel="deposit" href="/account/12345/deposit" />
</account>
```

Resposta p/ saldo de -25,00



Fonte: <http://en.wikipedia.org/wiki/HATEOAS>

REST API – Princípios de design



Versionamento da API

- Incorporar a versão do serviço na URI
`http://myservice.org/v1/discussion/{year}/{day}/{month}/{topic}`
- Incluir informações da versão na representação de estado do recurso

```
<beer version="1.0">
  <name>Asahi Draft Beer</name>
  <brewer>
    <name>Asahi</name>
    <country>Japan</country>
  </brewer>
  <calories>41</calories>
  <alcohol>5.21</alcohol>
</beer>
```

REST API – Princípios de design

Controle adequado do cache – Cabeçalhos HTTP



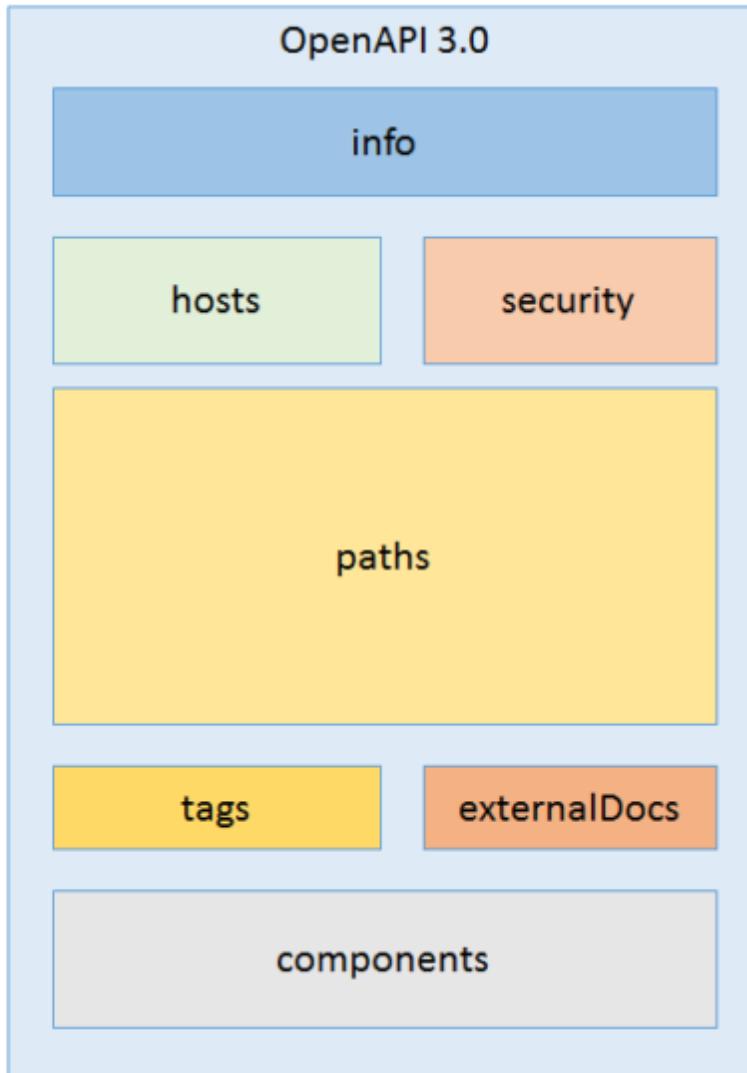
CABEÇALHO	APLICAÇÃO
Date	Data e hora de geração da representação.
Last Modified	Data e hora de última alteração da representação.
Cache-Control	O cabeçalho utilizado pelo HTTP 1.1 para controle de cache.
Expires	Data e hora de expiração da representação. Para compatibilizar com clientes HTTP 1.0.
Age	Tempo passado em segundos desde que a representação foi extraído do servidor. Pode ser inserido por um cache intermediário.
ETag	Indicador de versão específica de uma representação. Se houver alteração, a terá um novo valor.

REST API – Princípios de design

Controle adequado do cache – Valores para o cabeçalho Cache-Control

DIRETIVA	APLICAÇÃO
Public	Valor padrão. Indica que a representação pode ser armazenada em qualquer cache.
Private	Indica que a representação é para um único cliente e caches intermediários não podem armazenar a representação, apenas um cache privativo.
no-cache/no-store	Armazenamento da representação em cache desabilitado.
max-age	Tempo máximo de validade da representação (em segundos), tomando como base a data informada no cabeçalho "Date".
s-maxage	Similar a max-age, porém válido apenas para caches intermediários.
must-revalidate	Indica que a representação deve ser revalidada pelo servidor evitando o uso de recursos expirados
proxy-validate	Similar ao must-revalidate porém se aplica apenas aos caches intermediários.

REST API – Princípios de design



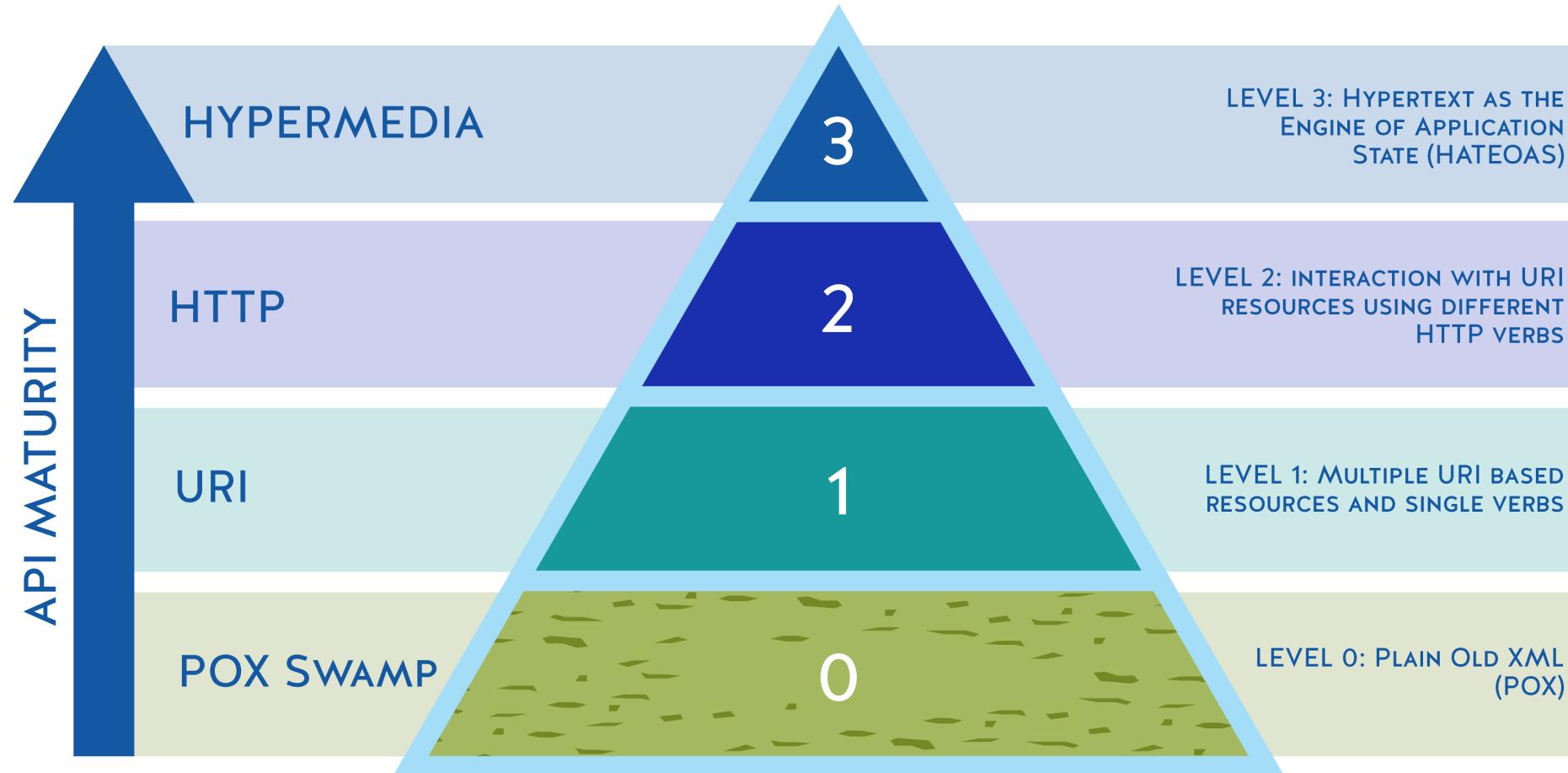
Documentação clara e adequada da API

OpenAPI Specification

- Padrão para definição de APIs RESTful
- Independente de linguagem de programação
- Permite a geração automática de código



Richardson Maturity Model



Fontes:

- [Richardson Maturity Model](#)
- [What is the Richardson Maturity Model?](#) (NORDICAPIS)

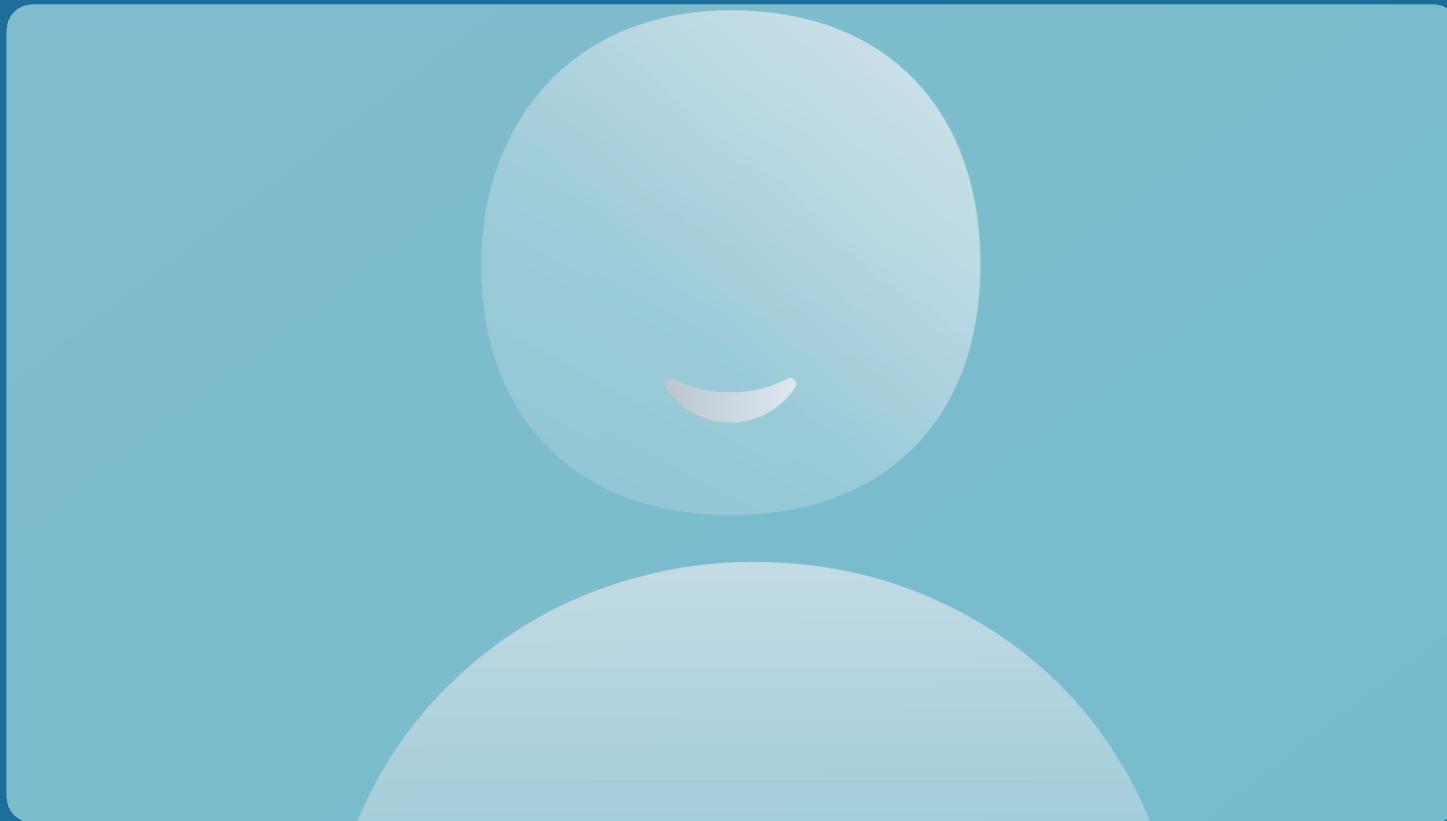


Resumo

- Conceitos e fundamentos do estilo arquitetural REST.
- Requisitos para uma APIs ser considerada compatível com o padrão REST.
- Princípios e boas práticas envolvidas na construção de APIs REST com a apresentação de exemplos práticos.
- Modelo de maturidade para a avaliação de APIs REST.



Graph Query Language (GraphQL)



GraphQL

GraphQL é uma linguagem de consultas para dados em grafos e um ambiente que executa operações nos dados voltada para a criação de APIs.



Vantagens

- Endpoint único ao invés de diversas entradas como no padrão REST
- Independente de linguagem, protocolo ou framework
- Maior controle do cliente via linguagem de consulta (Graph Query Language)
- Acabar com o Overfetching e o Underfetching
- Redução do tráfego de dados e o número de requisições
- Reduz a necessidade de manutenções de uma API

Fontes:

- [Introduction to GraphQL \(GraphQL\)](#)
- [GraphQL: Introdução a linguagem do Facebook para APIs](#)
- [Você conhece o GraphQL e como ele pode te ajudar? | Blog TreinaWeb](#)



GraphQL - Conceitos importantes



Schema

- Especificação de uma API GraphQL, escrita em uma linguagem denominada Schema Definition Language (SDL).
- Estabelece os tipos de dados possíveis de serem em uma API baseada em GraphQL.
- Define as operações de consulta (Queries) e alteração (Mutations) disponíveis no serviço que oferece a API GraphQL.

```
type User {  
    id: ID!  
    name: String!  
    email: String!  
}  
  
type Query {  
    getUser(id: ID!): User  
    getAllUsers: [User]  
}  
  
type Mutation {  
    createUser(name: String!, email: String!): User  
    updateUser(id: ID!, name: String, email: String): User  
    deleteUser(id: ID!): ID  
}
```

GraphQL - Conceitos importantes



Tipo Objeto

- Type define um objeto, o elemento mais básico do Schema GraphQL

Tipos escalares

- String – Para propriedades baseadas em texto (UTF-8).
- Integer – Para propriedades numéricas.
- Float – Para propriedades numéricas com uma parte decimal.
- Boolean – Para propriedades binárias de um objeto (true ou false).
- Unique identifiers (ID) – Para descrever um identificador único para um objeto. São serializadas com strings, porém possuem tratamento diferenciado.

```
type Product {  
    id: ID! # "Product" é um tipo de objeto  
    name: String! # "!" indica que é um campo obrigatório  
    price: Float!  
    description: String!  
}
```

GraphQL - Conceitos importantes

Tipos enumerados

- Tipo especial restrito a um conjunto particular de valores

```
enum Status { # "Status" pode ser "ACTIVE" ou "INACTIVE"  
  ACTIVE  
  INACTIVE  
}
```

Listas

- Definidas por meio do modificador de colchetes [e].

```
...  
  Categories: [String!]. # "[]" indica que é um array de categorias  
...
```



GraphQL - Conceitos importantes

Tipo Query

- Operações disponíveis em uma API GraphQL que permitem obter dados do servidor



```
type Query {  
    users: [User!]! # "[]" indica que é um array de usuários  
    user(id: ID!): User # "user" recebe um id e retorna um usuário  
    products: [Product!]! # "[]" indica que é um array de produtos  
    product(id: ID!): Product # "product" recebe um id e retorna um produto  
    categories: [Category!]!  
    category(id: ID!): Category  
}
```

GraphQL - Conceitos importantes

Mutation

- Operações disponíveis em uma API GraphQL que permitem modificar os dados no servidor e recuperar os dados modificados.



```
type Mutation {  
...  
    createProduct (name: String!, price: Float!,  
                  description: String!, categories: [ID!]!): Product  
    updateProduct(id: ID!, name: String, price: Float,  
                  description: String, categories: [ID!]): Product  
    deleteProduct(id: ID!): Boolean  
...  
}
```

GraphQL - Conceitos importantes

Resolvers

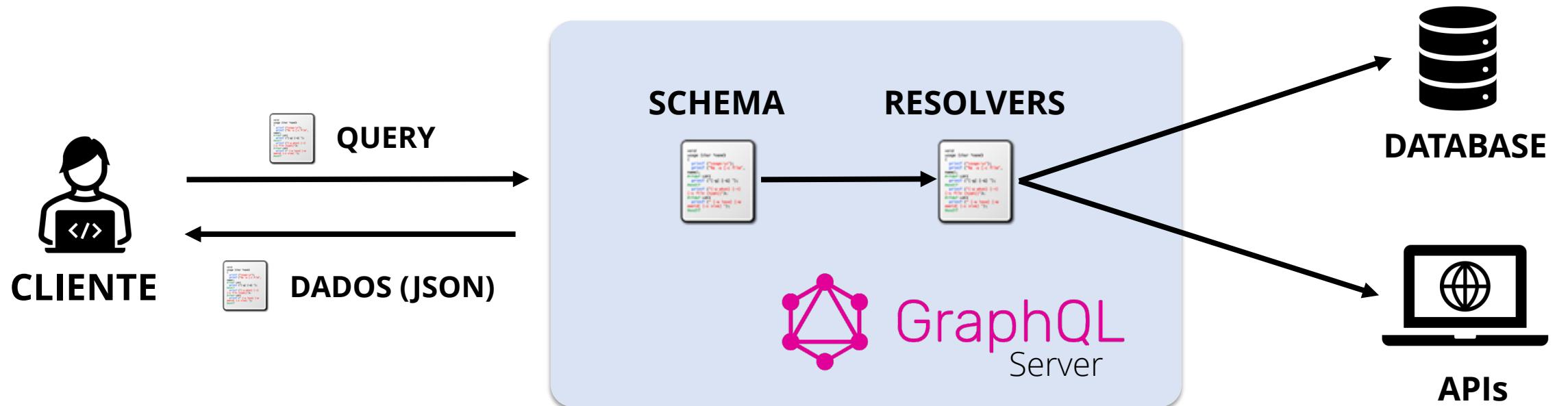
- Funções que implementam a lógica por traz das queries e mutations definidas no Schema de um serviço.
- Cada campo no Schema, deve ter um resolver correspondente que implementa o que é necessário para buscar os dados ou executar ações relacionadas.



```
const resolvers = {
  Query: {
    ...
    products: () => products,
    product: (_, { id }) => products.find(product => product.id === id),
    ...
  },
  Mutation: {
    createUser: (_, { name, email, password }) => {
      const user = { id: generateID(), name, email, password }
      users.push(user)
      return user
    }
  }
}
```



GraphQL - Arquitetura



GraphQL - Query

```
query {  
  posts  
}
```

Consulta simples para obter todos os dados de todos os posts



```
query {  
  posts {  
    likes  
    comments  
    shares  
  }  
}
```

Consulta para obter apenas alguns dados dos posts (likes, comments e shares)

```
query {  
  posts (userId: 157321){  
    likes  
    comments  
    shares  
  }  
}
```

Consulta para obter apenas alguns dados de posts específicos por meio de filtro por ID de usuário

Fonte: [Você conhece o GraphQL e como ele pode te ajudar?](#)



145

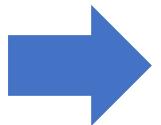


Prof. Rommel Vieira Carneiro

GraphQL - Query

tbl_professor		
PK	id_professor	integer
	dsc_nome_professor	character varying(200)
	dsc_num_telefone	character varying(20)
	dsc_email	character varying(200)
	flag_casa	boolean
	dsc_titulacao	character varying(20)
	dsc_email2	character varying(200)
	dsc_email3	character varying(200)
	dsc_num_telefone2	character varying(20)

Consulta para a entidade de professores, utilizando a definição de uma variável que é utilizada como filtro para seleção dos professores na base de dados.



```
query ($id: Int!) {  
  professores (id_professor: $id) {  
    id_professor  
    dsc_nome_professor  
    flag_casa  
    dsc_email  
  }  
}
```

Query String → id=37

Query

```
{  
  "data": {  
    "professores": [  
      {  
        "id_professor": 37,  
        "dsc_nome_professor": "Fulano de Tal",  
        "flag_casa": true,  
        "dsc_email": "fulanotal@gmail.com"  
      }  
    ]  
  }  
}
```

Resultado

GraphQL - Playground Tool



```
1 ▼ query ($id: Int!) {  
2   professores (id_professor: $id) {  
3     id_professor  
4     dsc_nome_professor  
5     flag_casa  
6     dsc_email  
7   }  
8 }
```

A large white play button icon is positioned in the center of the code area.

```
  ▼ {  
    ▼ "data": {  
      ▼ "professores": [  
        {  
          "id_professor": 37,  
          "dsc_nome_professor": "Rommel Vieira Carneiro",  
          "flag_casa": true,  
          "dsc_email": "rommelcarneiro@gmail.com"  
        }  
      ]  
    }  
  }
```

QUERY VARIABLES HTTP HEADERS (1)

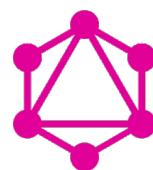
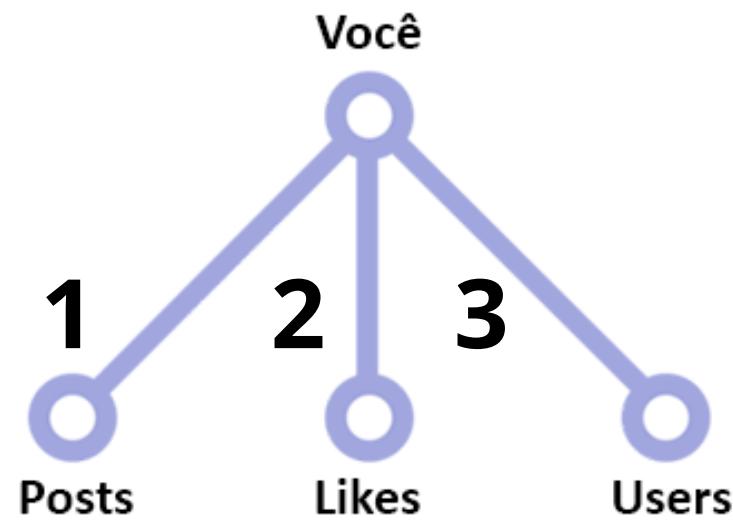
```
1 {  
2   "id": 37  
3 }
```

GraphQL

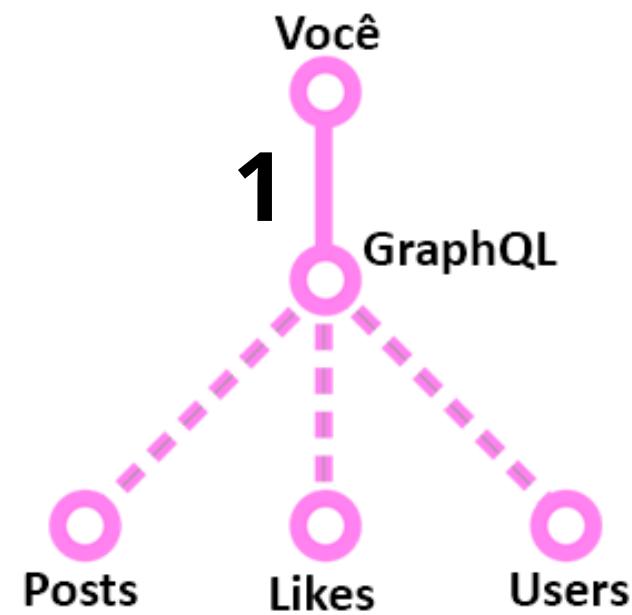
Imagine um cenário em que precisamos mostrar detalhes de um post, seus likes e os usuários associados ...



{ REST:API }



GraphQL



GraphQL vs REST

GraphQL	REST
Linguagem de consulta oferece eficiência e flexibilidade para resolver problemas comuns ao integrar APIs	Um estilo arquitetural amplamente visto como padrão para o desenho de APIs
Disponível via um único endpoint HTTP que provê todas as funcionalidades do serviço	Disponível via um conjunto de URLs, cada uma expondo um único recurso.
Utiliza uma arquitetura baseada no cliente	Utiliza uma arquitetura baseada no servidor
Dificulta o uso do mecanismo de cache	Utiliza cache de maneira simplificada e automática
Não trabalha com versionamento de API	Suporta múltiplas versões da API
Trabalha apenas com JSON	Suporta múltiplos formatos de dados (JSON, XML, etc)
Oferece uma única forma de documentação: GraphiQL	Oferece uma faixa de opções para documentação automatizada (OpenAPI e API Blueprint)
Dificulta o uso de códigos de status do protocolo HTTP para identificação de erros.	Utiliza os códigos de status do protocolo HTTP para identificar facilmente os erros

Fonte: [GraphQL vs. REST: Which Is the Best for API Development?](#)

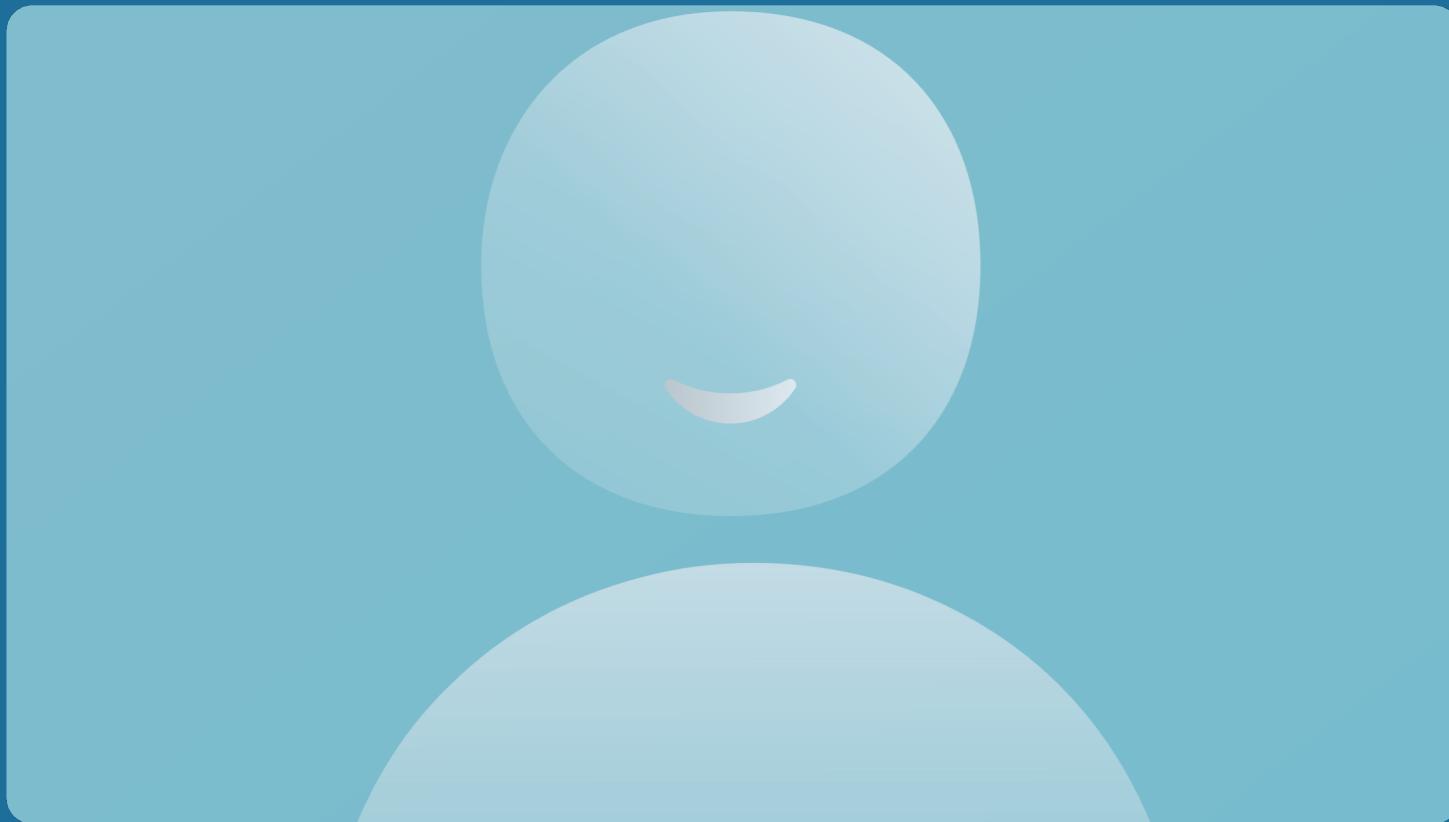


Resumo

- Conceitos fundamentais envolvidos em APIs baseadas em GraphQL.
- Exemplos de schema e resolvers utilizados em cenários reais de APIs GraphQL.
- Visão geral da arquitetura de sistema de um serviço baseado no padrão GraphQL.
- Comparativo entre APIs GraphQL e APIs REST.



Webhooks



Webhooks

Webhooks, as vezes chamados de callbacks HTTP são um tipo de API, baseado em eventos, que permite o envio de notificações entre aplicações.

Características

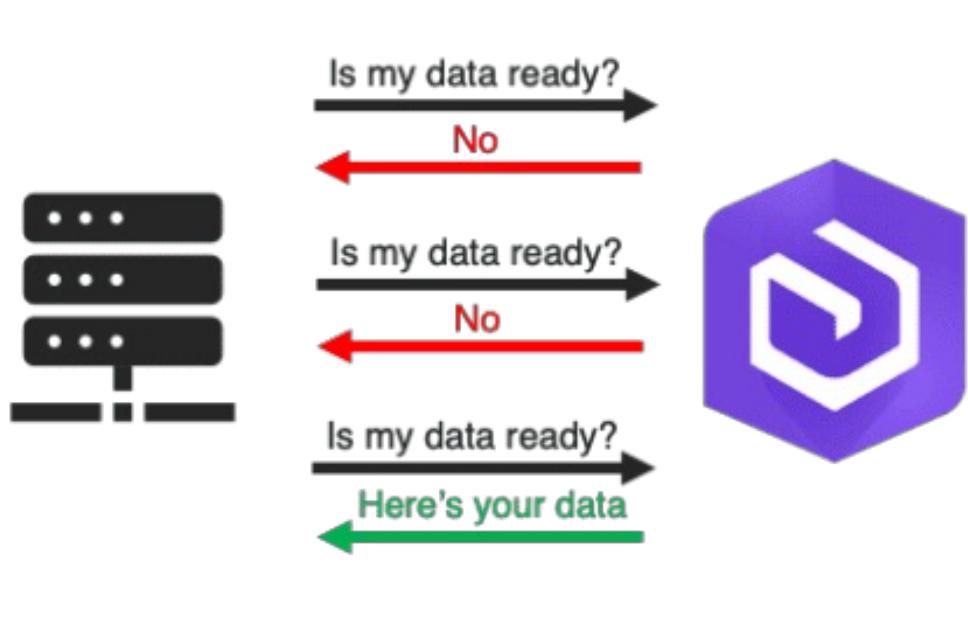
- Disparados a partir de eventos específicos:
ex: alteração em um pagamento, issues em um repositório GitHub
- Acessíveis via URL disponível publicamente (endpoints)
- Normalmente implementado via HTTP Post com payload JSON

* Empresas que utilizam Webhooks:
[GitHub](#), [Zapier](#), [Stripe](#).



Webhooks

Polling



Webhooks



Fonte: [What is a Webhook & How to use it to Track Email Marketing Activity](#)

Webhooks com Node.js

```
const express = require('express');
const app = express();

app.use(express.json());

// Rota para receber o payload JSON do webhook
app.post('/webhook', (req, res) => {
  const payload = req.body;
  database.push(payload);
  sendNotification(payload);

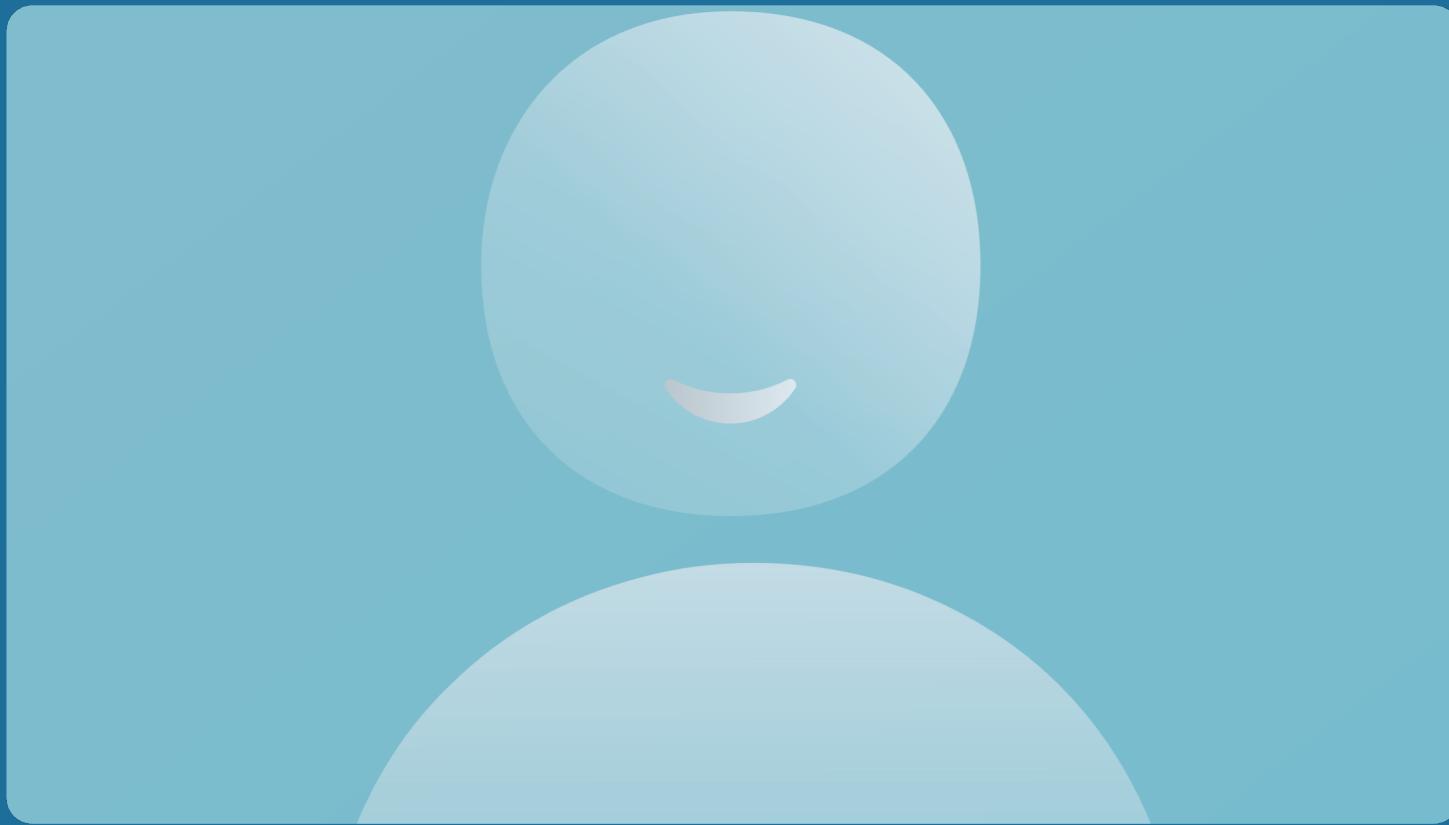
  res.status(200).send('Webhook recebido.');
});

// Dispara notificação para o usuário
function sendNotification(payload) {
  console.log('Notifica usuário', payload);
}

app.listen(3000, () => { console.log('Servidor Ok') });
```



WebSockets



Websockets

Protocolo aberto que permite comunicação em duas vias entre clientes e servidores



Características

- Conexão persistente entre cliente e servidor
- Permite envio de mensagens a partir do servidor de maneira independente

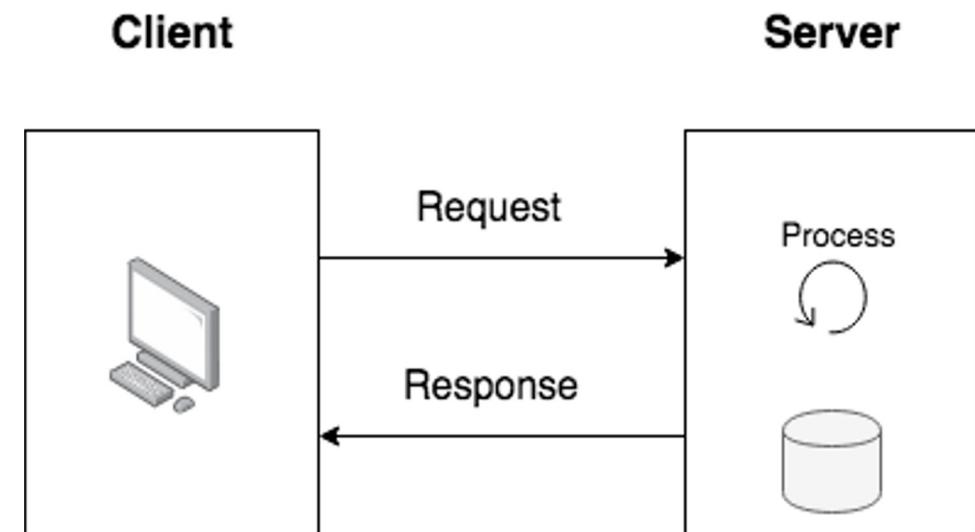


Fonte: [The WebSocket Protocol \(RFC-6455\)](#)

Web Sockets vs Comunicação HTTP

Comunicação HTTP

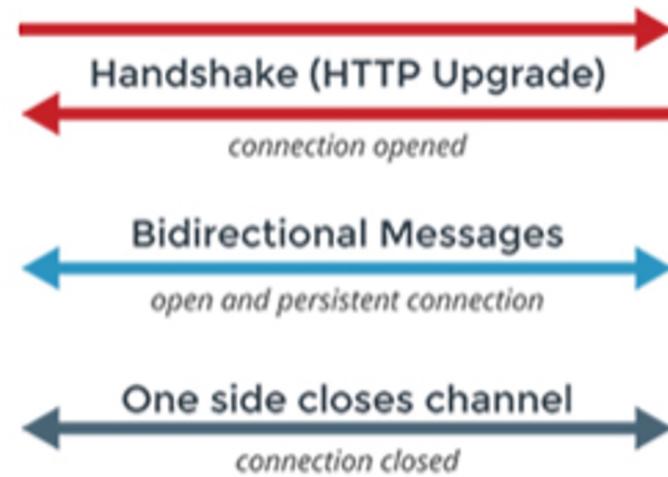
- Conexão não persistente
Requisições exigem a negociação do protocolo TCP setup and teardown
- Comunicação sempre iniciada pelo cliente
Sem envio de mensagens pelo servidor
- Associada apenas com HTTP/HTTPS



Comunicação entre cliente e servidor (websockets/wss)



Client



Server



Fonte: <https://pushfyi.com/an-introduction-to-websockets-beginners-guide>

Comunicação entre cliente e servidor



Cliente

1

Navegador faz requisição e solicita upgrade

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhBxBsZSBr25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```



Servidor
Web

2

Servidor exige autenticação

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept:
s3pPLMBiTxaQ9kYGzhZBk+x0o=
Sec-WebSocket-Protocol: chat
```

Fonte: [Introduction to websockets - a beginners guide.](#)



159



Prof. Rommel Vieira Carneiro

Casos de uso dos WebSockets

- Sistemas em tempo real
- Transmissão de jogos em sites
- Edição colaborativa
- Jogos multiplayer
- Aplicações de Chat
 - Whatsapp;
 - Messenger;
 - Telegram;
 - BLiP.



WebSockets com Node

Criando o servidor HTTP e WebSocket

```
const webSocketServer = require('websocket').server;
const http = require('http');
const webSocketServerPort = 8000;

// Start the http server and the websocket server
const server = http.createServer();
server.listen(webSocketServerPort);

const wsServer = new webSocketServer({
  httpServer: server
});
```



Fonte: [WebSockets Tutorial: Going Real-time with Node and React](#)

WebSockets com Node

Processando conexões

```
const clients = {};  
  
const generateUniqueID = () => {  
  const s4 = () => Math.floor((1 + Math.random()) * 0x10000).toString(16).substring(1);  
  return s4() + '-' + s4() + '-' + s4();  
};  
  
wsServer.on('request', function(request) {  
  var userID = generateUniqueID();  
  
  console.log(new Date(), 'Conexão de ' + request.origin);  
  
  const connection = request.accept(null, request.origin);  
  
  clients(userID] = connection;  
  console.log('connected: ' + userID)  
});
```

Fonte: [WebSockets Tutorial: Going Real-time with Node and React](#)



Socket.io

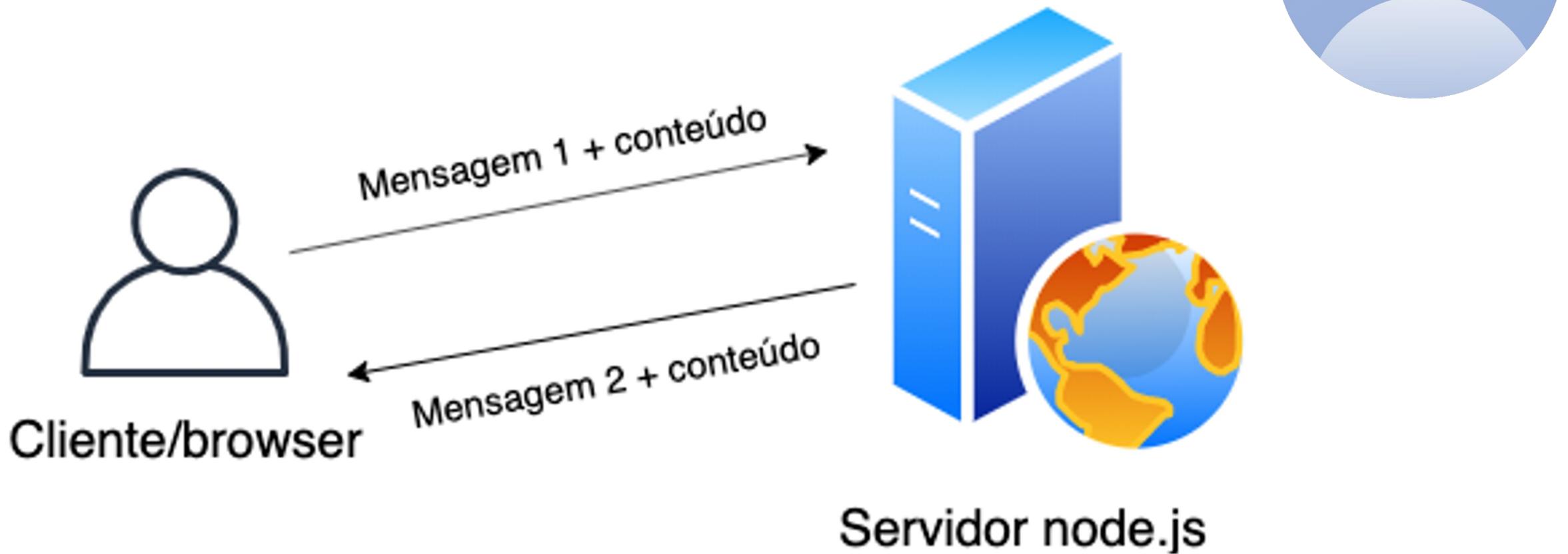


- Interface para criação de websockets no node;
 - Funciona no cliente e no servidor;
- Muito utilizado para criação de aplicativos de mensagens e chatbots;
- Arquitetura orientada a eventos.



socket.io

Socket.io



Socket.io (connection)

Cliente

```
1 <script src="/socket.io/socket.io.js"></script>
2 <script>
3   const socket = io('http://localhost:3000');
4 </script>
```



Servidor

```
1 io.on('connection', socket => {
2   console.log(`Usuário conectado com o id: ${socket.id}`)
3 })
```

Socket.io (sender)



```
1 // Strings  
2 socket.emit('mensagem', 'Conteúdo da mensagem')  
3  
4 // Objetos  
5 socket.emit('mensagem', { author: 'Samuel', bio: 'Hello world!' })  
6  
7 // Arrays  
8 socket.emit('mensagem', [{ text: 'Hello' }, { text: 'World' }])
```

Socket.io (receiver)



Mensagem

Callback

```
1 socket.on('mensagem', (conteudo) => {  
2   console.log(conteudo)  
3 });
```

Socket.io (broadcasting)

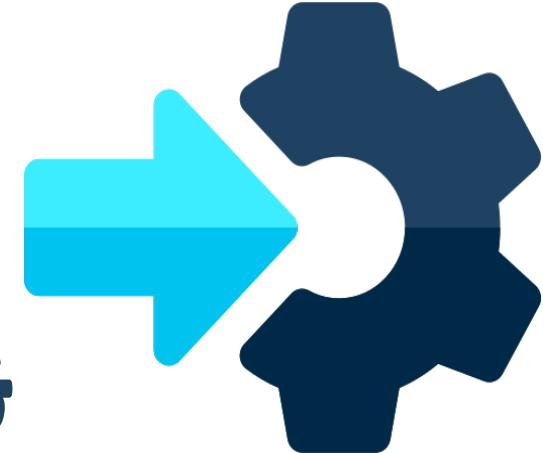
- Envia mensagens para todos os usuários;
- Caso esteja no lado do servidor, o objeto “socket” **sempre** deve ser utilizado dentro do método
io.on('connection', (socket) => { ... })



```
1 io.on('connection', (socket) => {
2     socket.on('mensagem', conteudo => {
3         socket.broadcast.emit('mensagemParaTodos', conteudo)
4     })
5 })
```

Obrigado!

APIs &
Web Services



Prof. Rommel Vieira Carneiro

in [/rommelcarneiro](https://www.linkedin.com/in/rommelcarneiro)



<http://rommelcarneiro.me/>