OpenAPI na Prática

Um guia prático para dominar a documentação de APIs e impulsionar sua carreira como desenvolvedor

Quem Sou Eu

Julio Cauan, 26 anos

Analista Sênior na Accenture

Especialista em Java, entusiasta de Linux e do movimento Software Livre, porque conhecimento só cresce quando é compartilhado.

Aprendizado técnico vem com persistência, muita prática e mais erros ainda. Já o crescimento real vem das soft skills. Saber ouvir, comunicar ideias com clareza e colaborar com pessoas diferentes!

Como Sistemas Web Modernos se Comunicam

Cliente

Front-end faz requisição HTTP

Rede

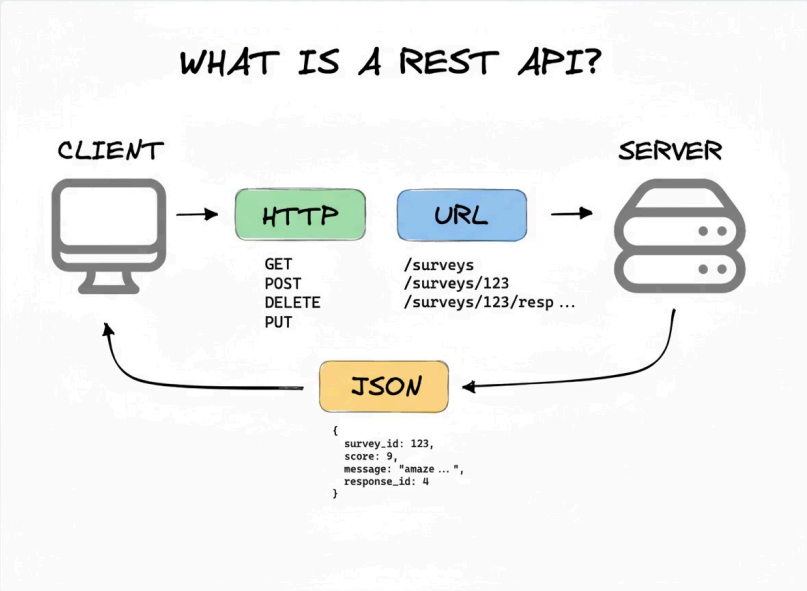
Dados trafegam como bits pela internet

Servidor

Back-end processa e responde

O modelo cliente-servidor é a base da web moderna. APIs funcionam como pontes, permitindo que diferentes sistemas conversem entre si através de endpoints bem definidos e métodos HTTP padronizados.

Diferente dos WebSockets, que mantêm uma conexão persistente, as requisições HTTP são **stateless** (sem estado). Isso significa que cada requisição é independente e a conexão é finalizada após a resposta, não guardando informações sobre requisições anteriores. Essa característica torna o HTTP mais simples e escalável, mas requer mecanismos como cookies ou tokens para manter contexto entre requisições.



Padrões de Comunicação

SOAP

Baseado em XML, mais antigo e burocrático. Usado em sistemas legados corporativos que exigem contratos rígidos e protocolos complexos.

REST

Simples, direto e amplamente usado. É o padrão da indústria, utilizando métodos HTTP intuitivos (GET, POST, PUT, DELETE) e formato JSON.

GraphQL

Alternativa moderna e flexível. Permite que o cliente solicite exatamente os dados necessários, reduzindo over-fetching e under-fetching.

O Fluxo de Dados REST

Antes de entender o fluxo, é importante conhecer os **métodos HTTP** fundamentais: **GET** (buscar dados), **POST** (criar novos recursos), **PUT** (atualizar recursos existentes), **DELETE** (remover recursos) e **PATCH** (atualizações parciais). Cada método tem um propósito específico e segue convenções REST para operações CRUD (Create, Read, Update, Delete).

01

Cliente prepara dados

Cria objetos DTO (Data Transfer Object) com informações estruturadas

02

Serialização JSON

Converte objetos em formato JSON legível

03

Transmissão de bits

Dados viajam pela rede como sequências binárias

04

Deserialização

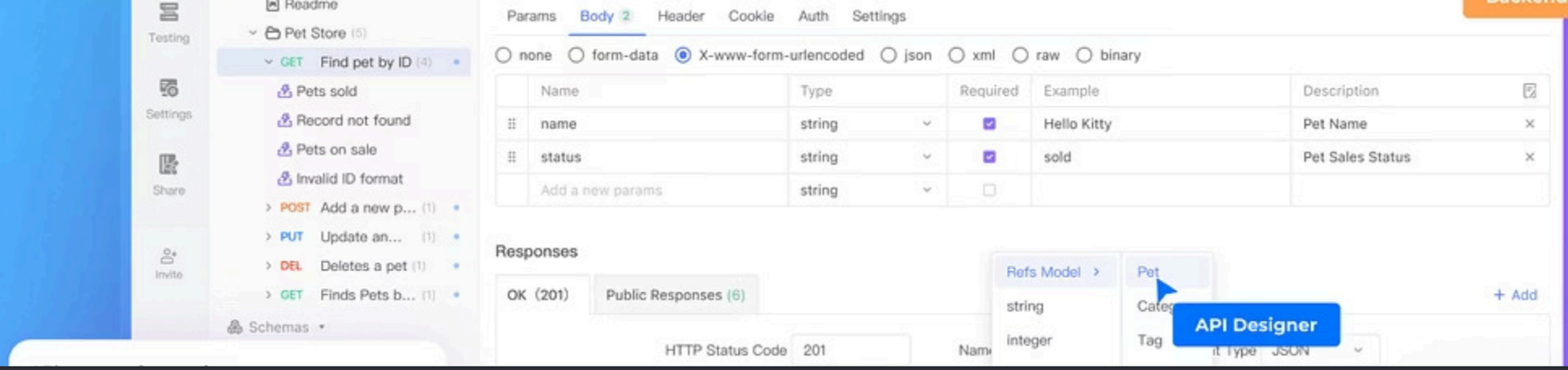
Back-end reconstrói objetos DTO a partir do JSON recebido

05

Processamento

Servidor executa lógica de negócio e retorna resposta

Exemplo prático: Quando você faz login, suas credenciais seguem exatamente esse fluxo até serem validadas no servidor!



Por Que Documentação é Essencial

Contratos Claros

Define expectativas entre front-end e back-end, eliminando ambiguidades

Facilita Onboarding

Novos desenvolvedores entendem rapidamente como integrar sistemas

Reduz Erros

Evita retrabalho e bugs de integração causados por má comunicação

Manutenção Simplificada

Facilita evolução e troubleshooting de APIs em produção

Empresas como Banco do Brasil, C6 Bank e Shopify adotam OpenAPI/Swagger como padrão para garantir qualidade e eficiência.

OpenAPI: O Padrão da Indústria

Swagger: A Visão Original

Iniciativa **open source** criada para facilitar a **documentação** e **padronização** de APIs REST.

2015: Aquisição e Doação

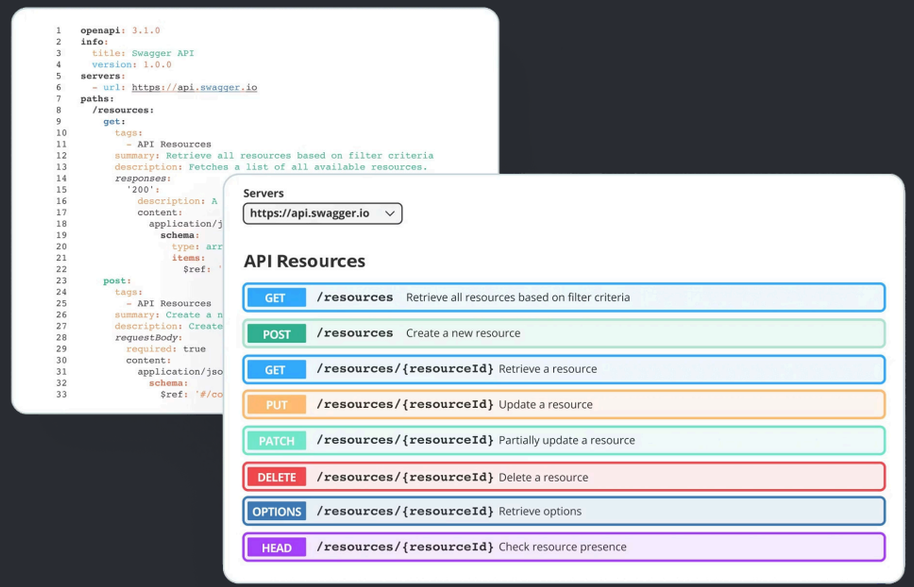
Em **2015**, o Swagger foi **adquirido pela SmartBear Software**. Sua especificação foi **doada à Linux Foundation**.

O Nascimento do Padrão

A doação originou a **OpenAPI Specification (OAS)** — hoje o **padrão oficial** mantido pela **OpenAPI Initiative**.

Separação de Conceitos

Desde então, "**Swagger**" refere-se apenas ao **conjunto de ferramentas** (Editor, UI, Codegen), enquanto "**OpenAPI**" é a **especificação**.



Ferramentas do Ecossistema OpenAPI

Postman & Insomnia

Importam especificações OpenAPI para testar endpoints rapidamente, validando requests e responses

Swagger UI & Editor

Geram documentação interativa, permitindo que usuários testem a API diretamente no navegador

OpenAPI Generator

Cria automaticamente código client/server em Node, Java, Python e mais de 50 linguagens

Comparação: OpenAPI vs Documentação Legada

APIs Modernas com OpenAPI

Banco do Brasil Documentation

C6 Bank Documentation

Documentação interativa, padronizada e fácil de usar.

Documentação Legada

e-Social Documentation

Documentação estática, complexa e difícil de navegar.

SEMPRE OLHE A DOCUMENTAÇÃO!

A documentação é a espinha dorsal do desenvolvimento e consumo de APIs!! Por mais que a inteligência artificial possa ajudar, **nunca confie apenas nela**.

Fluxo de Documentação OpenAPI

Da especificação à implementação e consumo, a documentação OpenAPI guia o ciclo de vida da API, garantindo clareza e consistência.

Para saber mais sobre as ferramentas e padrões, consulte:

- https://swagger.io/
- https://openapi-generator.tech/

Estrutura Principal da Especificação OpenAPI

O documento OpenAPI funciona como um contrato detalhado para sua API. Ele descreve a API de forma que tanto pessoas quanto máquinas a entendam sem precisar do código-fonte.

openapi

Versão da especificação.

info

Dados gerais da API.

servers

Endereços da API.

tags

Categorias da API.

paths

Rotas e operações.

components

Elementos reutilizáveis.

schemas

Estrutura dos dados.

Componentes Essenciais e Funções

1. openapi

Indica a versão da Especificação OpenAPI que o documento segue.

Propósito: Garante que ferramentas e parsers interpretem o documento corretamente.

Componentes Essenciais e Funções

2. info

Contém metadados importantes sobre a API, como título, descrição, versão, informações de contato e licença.

Propósito: Ajuda desenvolvedores a entender rapidamente do que se trata a API.

Componentes Essenciais e Funções

3. `servers`

Lista os URLs base onde a API está disponível, permitindo definir diferentes ambientes (ex: desenvolvimento, homologação e produção).

Propósito: Informa onde a API pode ser acessada e utilizada.

Componentes Essenciais e Funções

4. tags

Permite agrupar operações relacionadas da API para uma melhor organização na documentação gerada.

Propósito: Categorizar e organizar visualmente a API.

Componentes Essenciais e Funções

5. paths

Esta é a parte central, descrevendo todos os endpoints da API (caminhos) e as operações HTTP (GET, POST, PUT, DELETE) disponíveis em cada um.

Propósito: Detalha as rotas e métodos específicos da API, incluindo seus parâmetros e respostas.

Componentes Essenciais e Funções

6. components

Define estruturas reutilizáveis como esquemas de dados, parâmetros e respostas. Isso evita duplicação e garante consistência.

Propósito: Promover a reutilização e padronização de elementos da API.

Componentes Essenciais e Funções

7. schemas

Define a estrutura dos dados utilizados na API, incluindo modelos de request e response. Os schemas descrevem os tipos de dados, propriedades obrigatórias e validações.

Propósito: Padronizar a estrutura dos dados e permitir validação automática de requests e responses.

Tipos de Dados Fundamentais no OpenAPI

A especificação OpenAPI utiliza um subconjunto do JSON Schema para descrever os tipos de dados de forma padronizada.

String

Dados textuais. Pode ser complementado por formatos para maior especificidade:

- **date:** Ex: "2024-01-20" (RFC 3339)
- **date-time:** Ex: "2024-01-20T14:30:00Z" (RFC 3339)
- **email:** Ex: "usuario@exemplo.com"
- **uuid:** Ex: "123e4567-e89b-12d3-a456-426614174000"
- **byte:** Sequência de octetos codificados em Base64
- **binary:** Qualquer sequência de octetos (upload de arquivo)
- **password:** Usado para campos de senha, geralmente oculto

```
{ "type": "string", "format": "email" }
```

Number

Números de ponto flutuante. Formatos para precisão:

- **float:** Ponto flutuante de precisão simples
- **double:** Ponto flutuante de precisão dupla

```
{ "type": "number", "format": "float" }
```

Integer

Números inteiros (sem casas decimais). Formatos para tamanho:

- **int32:** Inteiro assinado de 32 bits
- **int64:** Inteiro assinado de 64 bits (equivalente a 'long')

```
{ "type": "integer", "format": "int64" }
```

Boolean

Valores verdadeiro (true) ou falso (false). Não possui formatos adicionais.

```
{ "type": "boolean" }
```

Array

Uma lista ordenada de itens do mesmo tipo. O tipo dos itens é definido pela propriedade `items`.

```
{ "type": "array", "items": { "type": "string" } }
```

Object

Um mapa de propriedades nomeadas, onde cada propriedade tem um nome (string) e um valor (outro tipo de dado).

```
{
  "type": "object",
  "properties": {
    "nome": { "type": "string" },
    "idade": { "type": "integer", "format": "int32" }
  }
}
```

Testando na Prática: Petstore Example

O Petstore é um exemplo clássico da especificação OpenAPI, amplamente utilizado para demonstrar e testar APIs. Ele simula uma API para gerenciar um pet shop, permitindo operações como adicionar, atualizar e buscar animais, usuários e pedidos.

Acessando o Petstore Online

Comece navegando até a interface interativa do Petstore OpenAPI.
https://petstore3.swagger.io/

Baixando a Especificação OpenAPI

Localize e baixe o arquivo de especificação OpenAPI (geralmente `openapi.json` ou `swagger.json`) diretamente do site.

Gerando Código Cliente/Servidor

Com o OpenAPI Generator CLI, você pode gerar automaticamente código cliente (SDK) ou código de servidor em várias linguagens, usando a especificação que você baixou.

```
npx @openapitools/openapi-generator-cli generate -i petstore.json -g javascript -o .
```

Dica: O comando acima gera um cliente JavaScript no diretório atual. Experimente outras linguagens e analise a estrutura do código gerado para entender como ele interage com a API. Isso é útil para acelerar o desenvolvimento.

Você pode adicionar propriedades de configuração extras usando `--additional-properties`, como por exemplo: `--additional-properties=useInheritance=true,usePromises=true`. As configurações disponíveis variam por linguagem, por isso é importante consultar a documentação específica de cada gerador para ver todas as opções.

Importando no Postman

Importe o arquivo JSON da especificação OpenAPI que você baixou. O Postman irá criar automaticamente uma coleção com todos os endpoints e modelos da API, prontos para serem usados.

O que esperar: Você verá uma coleção "Petstore" com pastas para "pet", "store" e "user", cada uma contendo requisições pré-configuradas.

Executando Testes de API

No Postman, selecione uma das requisições geradas (por exemplo, `GET /pet/findByStatus`), ajuste os parâmetros conforme necessário (como `status=available`) e clique em "Send". Observe a resposta da API para verificar se está de acordo com o esperado.

Dica: Experimente diferentes métodos (GET, POST, PUT, DELETE) e insira dados válidos e inválidos para entender o comportamento da API e como ela lida com erros.

OpenAPI em Arquiteturas de Microsserviços

Descoberta de Serviços

APIs bem documentadas facilitam para desenvolvedores encontrarem e entenderem rapidamente a funcionalidade de outros serviços.

Contratos Claros

A especificação OpenAPI atua como um contrato formal entre serviços, evitando quebras de integração inesperadas e garantindo que todos os lados tenham uma compreensão unificada da API.

Geração Automática de Clientes

Ferramentas como o OpenAPI Generator podem criar SDKs (kits de desenvolvimento de software) automaticamente para cada serviço, em diversas linguagens, reduzindo o trabalho manual e garantindo consistência.

Versionamento Controlado

Mudanças nos parâmetros ou na estrutura da API são explicitamente documentadas, permitindo um planejamento e gerenciamento mais eficazes do versionamento.

Testes Automatizados

A definição formal da API permite a criação de testes automatizados que validam continuamente os contratos entre os serviços.

Empresas como Netflix, Uber e Amazon, que operam com centenas ou até milhares de microsserviços, utilizam a documentação padronizada como um pilar fundamental para gerenciar a complexidade e escalar suas operações de forma eficiente.

Para lidar com mudanças de parâmetros e garantir a estabilidade do sistema, é prática comum aplicar o versionamento semântico (semantic versioning) e garantir a compatibilidade retroativa (backward compatibility) nas APIs. Isso significa que novas versões da API devem ser planejadas para não quebrar integrações existentes ou, quando necessário, as quebras devem ser claramente comunicadas e gerenciadas.

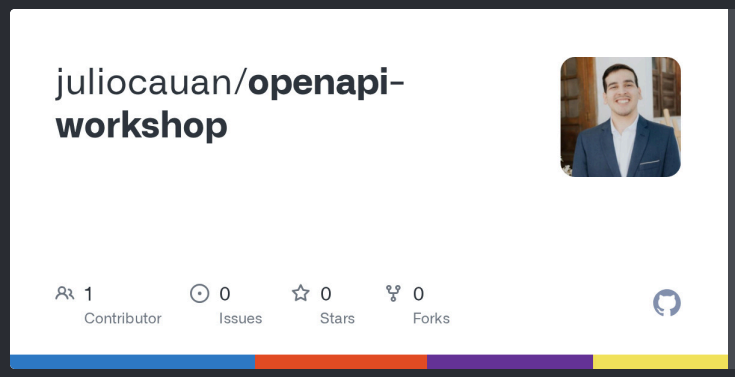
Projeto Guiado: Mão na Massa!

Formação de Grupos

Dividam-se em equipes colaborativas

Documente sua API

Corrijam implementação de Catálogo de Produtos utilizando o OpenAPI e OpenAPI Generator



GitHub

GitHub – juliocauan/openapi-workshop

Contribute to juliocauan/openapi-workshop development by creating an account on GitHub.

Visualize no Swagger UI

Renderizem documentação interativa

Teste no Postman

Validem endpoints com requisições reais

Compartilhe Resultados

Feedback coletivo e aprendizado em grupo

Continue Sua Jornada

Próximos Passos

- Explore OpenAPI 3.1 e suas novidades
- Conheça AsyncAPI para APIs assíncronas
- Domine versionamento de APIs
- Pratique em projetos pessoais

Mensagem Final

A documentação não é apenas técnica — é comunicação, é profissionalismo, é respeito pelo próximo desenvolvedor.

Persistência, aprendizado contínuo e propósito vão te levar mais longe do que você imagina. Pratique, erre e aprenda!

Sucesso na sua jornada!