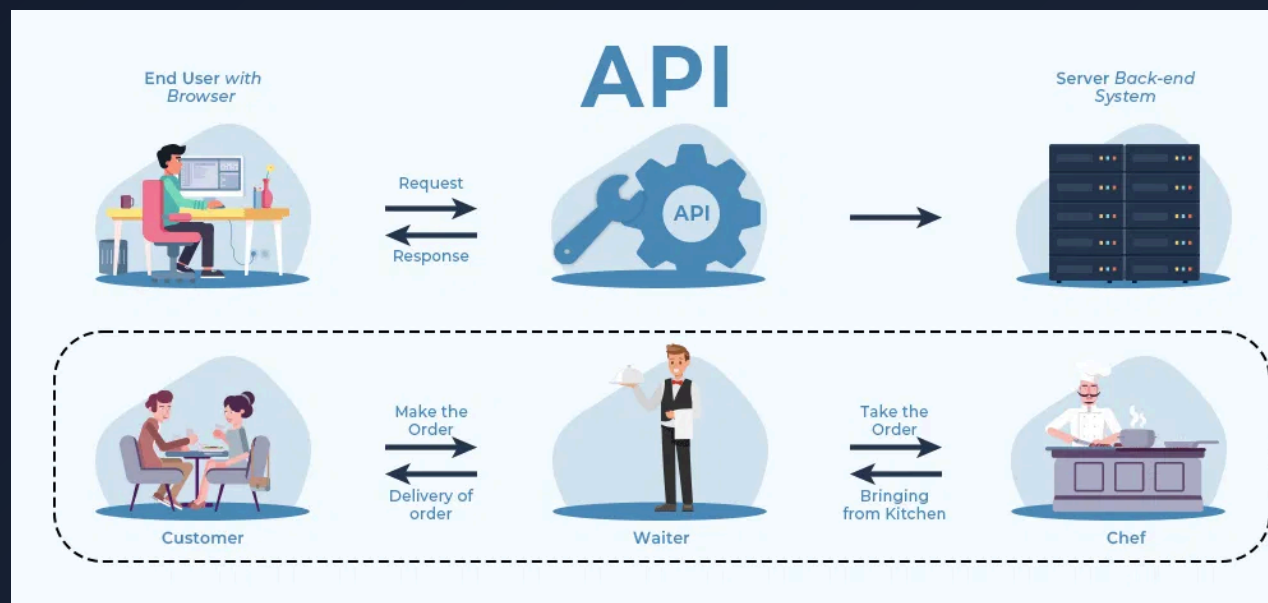


# Introdução às APIs

Conceitos, funcionamento e implementação prática em três linguagens

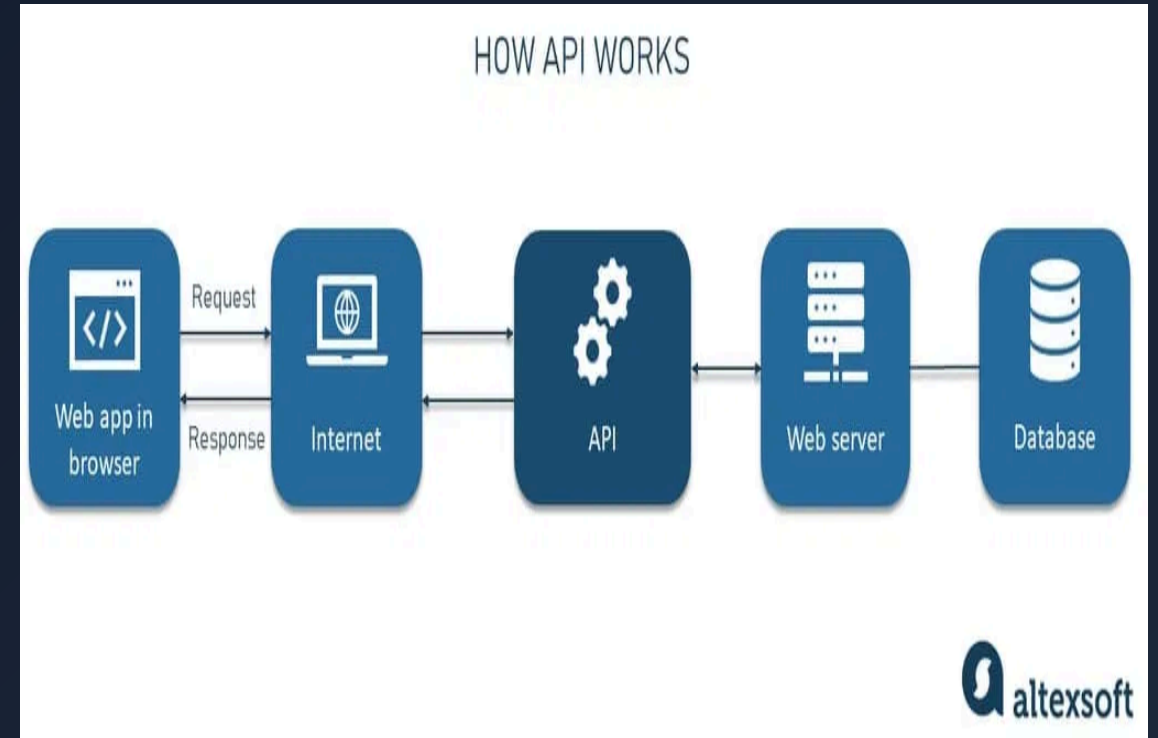


# O que é uma API?

**API = Application Programming Interface**

Uma forma padronizada de dois sistemas trocarem dados e funcionalidades

- 📍 Google Maps API
- ☁️ APIs de previsão do tempo
- 🎵 APIs de streaming de música
- 🏦 APIs bancárias



# Por que usar APIs?



## Reutilização de funcionalidades

Evita recriar soluções já existentes, economizando tempo e recursos de desenvolvimento.



## Separação de responsabilidades

Divide claramente as funções entre cliente e servidor, permitindo desenvolvimento independente.



## Escalabilidade e integração

Facilita o crescimento dos sistemas e a comunicação entre diferentes serviços e plataformas.



## Segurança e controle de acesso

Permite implementar camadas de autenticação e autorização para proteger dados e funcionalidades.

# Como funciona uma API Web?

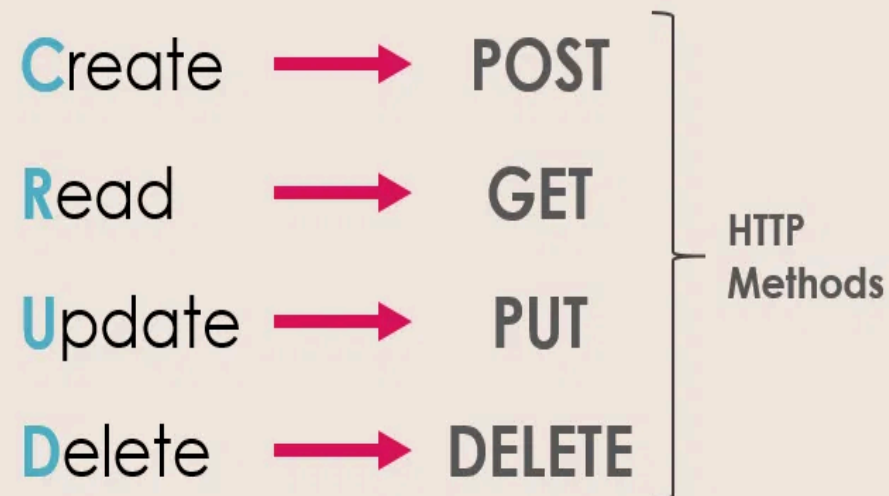
- Protocolos comuns: **HTTP, HTTPS**
- Respostas geralmente em **JSON**

**GET** Obter dados

**POST** Criar dados

**PUT** Atualizar dados

**DELETE** Remover dados



## Exemplo:

Cliente → GET /produtos → Servidor → JSON

# Componentes Essenciais



## Rotas/Endpoints

URLs que identificam recursos e operações específicas



## Métodos HTTP

Definem o tipo de operação a ser realizada



## Controllers

Processam requisições e coordenam respostas



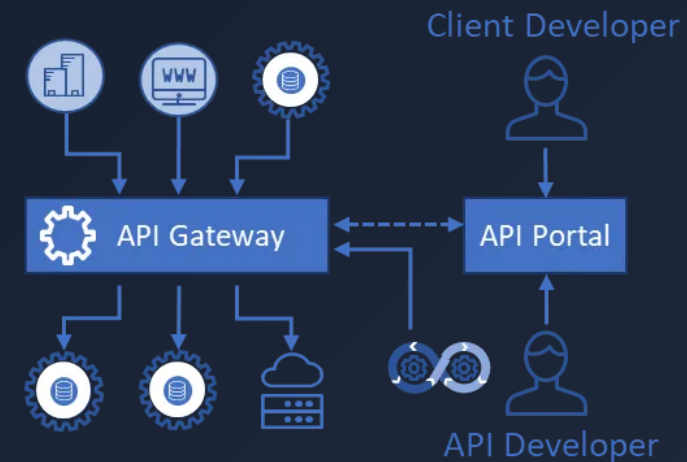
## Models

Representam dados e lógica de negócio



## Responses

Formatação e envio de respostas ao cliente



# Exemplo: Java (Spring Boot)

```
@RestController
public class ProdutoController {

    @GetMapping("/produtos")
    public List<Produto> listarTodos() {
        return produtoService.findAll();
    }

    @GetMapping("/produtos/{id}")
    public Produto buscarPorId(
        @PathVariable Long id) {
        return produtoService.findById(id);
    }
}
```

## Características do Spring Boot

- ✓ Tipagem forte e segura
- ✓ Uso de anotações para configuração
- ✓ POJOs (Plain Old Java Objects)
- ✓ Injeção de dependências
- ✓ Robusto e escalável

**i** **@GetMapping** mapeia requisições HTTP GET

# Exemplo: Python (Flask)

```
from flask import Flask, jsonify
from flask_restful import Resource, Api

app = Flask(__name__)
api = Api(app)

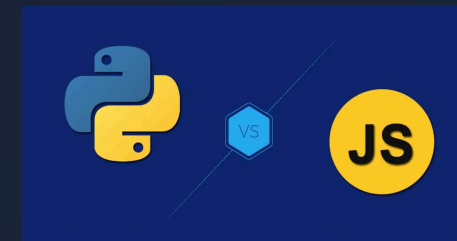
class Produtos(Resource):
    def get(self):
        return jsonify({'produtos': produtos})

class Produto(Resource):
    def get(self, produto_id):
        return jsonify(produtos[produto_id])

api.add_resource(Produtos, '/produtos')
api.add_resource(Produto,
                 '/produtos/<int:produto_id>')
```

## Características do Flask

- ✓ Leve e minimalista
- ✓ Flexível e extensível
- ✓ Fácil de aprender e usar
- ✓ Ideal para APIs pequenas e médias
- ✓ Excelente para prototipagem rápida



**i** **Flask** é um microframework web para Python

# Exemplo: JavaScript (Express)

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

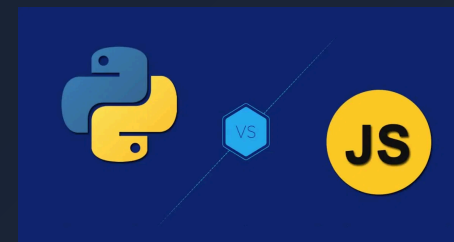
// Middleware para processar JSON
app.use(express.json());

// Rota para listar todos os produtos
app.get('/produtos', (req, res) => {
  return res.json(produtos);
});

// Rota para buscar produto por ID
app.get('/produtos/:id', (req, res) => {
  const { id } = req.params;
  return res.json(produtos[id]);
});
```

## Características do Express







- ✓ Assíncrono e orientado a eventos
- ✓ Leve e minimalista
- ✓ Middleware para processamento de requisições
- ✓ Ideal para aplicações web modernas
- ✓ Ecossistema rico de pacotes npm



**i** Express é um framework web para Node.js



# Comparativo das Linguagens

Linguagem	Framework	Características
 <b>Java</b>	Spring Boot	 Robusto e escalável
 <b>Python</b>	Flask	 Simples e rápido para desenvolvimento
 <b>JavaScript</b>	Express	 Leve e assíncrono

## Quando usar Java?

Aplicações empresariais, sistemas complexos e de alta performance

## Quando usar Python?

Prototipagem rápida, aplicações científicas e análise de dados

## Quando usar JavaScript?

Aplicações web modernas, tempo real e full-stack com Node.js

# Testando sua API

## Ferramentas de Teste

✓ Postman

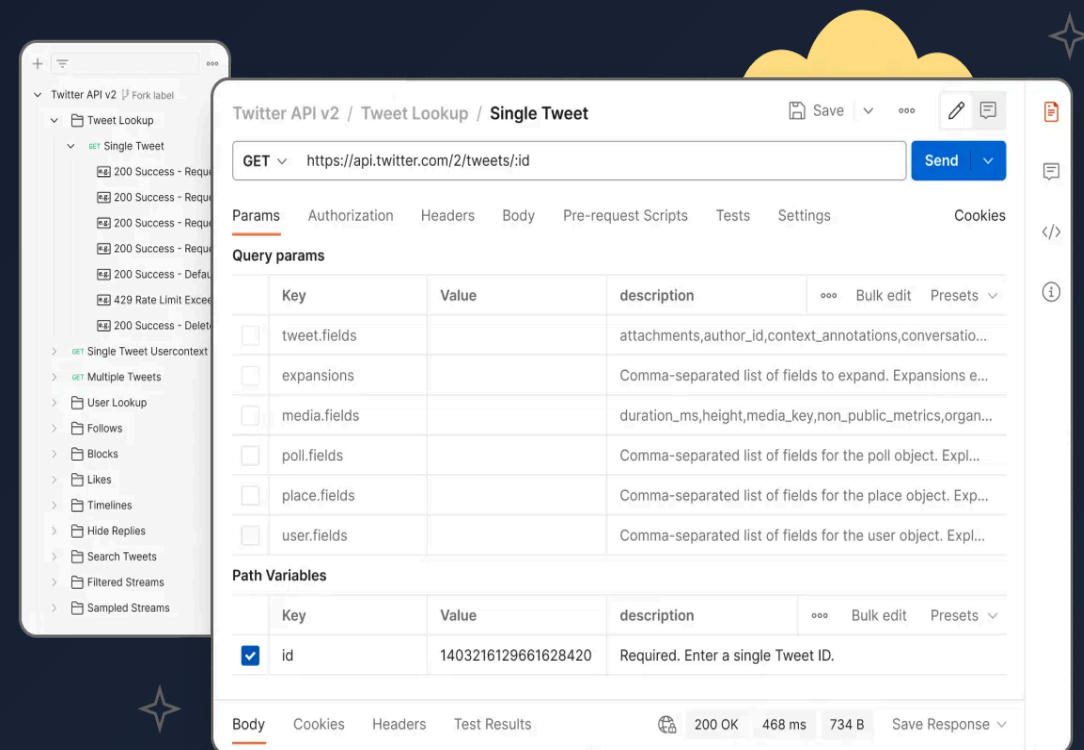
> cURL

</> Insomnia

Navegador (DevTools)

## O que testar?

- Requisições GET, POST, PUT, DELETE
- Parâmetros e headers
- Respostas em JSON



**i** O Postman facilita o teste de APIs com interface visual

# Boas Práticas



## Documentação (Swagger)

Documente endpoints, parâmetros e respostas



## Tratamento de erros

Retorne códigos HTTP e mensagens apropriadas



## Autenticação e autorização

Implemente JWT, OAuth ou outro mecanismo seguro



## Endpoints RESTful

Use nomes de recursos no plural e métodos HTTP adequados



## Versionamento

Adicione versão na URL ou nos headers



## Paginação e filtros

Implemente para coleções grandes de dados



Uma API bem projetada é mais fácil de usar, manter e escalar!

# Onde as APIs são Utilizadas?



## Aplicações Web e Mobile

Comunicação entre frontend e backend, permitindo interfaces ricas e responsivas



## Integração com Bancos de Dados

Acesso seguro e controlado a dados, com camadas de abstração e validação



## Microserviços

Comunicação entre diferentes componentes de sistemas distribuídos



## Sistemas Desktop

Integração com serviços web e recursos online em aplicações tradicionais



As APIs são a espinha dorsal da integração de sistemas modernos!

# Conclusão

## APIs conectam sistemas

Permitem a comunicação entre diferentes aplicações, independente da linguagem ou plataforma.

## Cada linguagem tem pontos fortes

Java, Python e JavaScript oferecem diferentes vantagens para o desenvolvimento de APIs.

## O que aprendemos:

- ✓ Conceitos fundamentais de APIs
- ✓ Implementações práticas
- ✓ Funcionamento e protocolos
- ✓ Boas práticas e casos de uso