Programação Front End – AULA 17

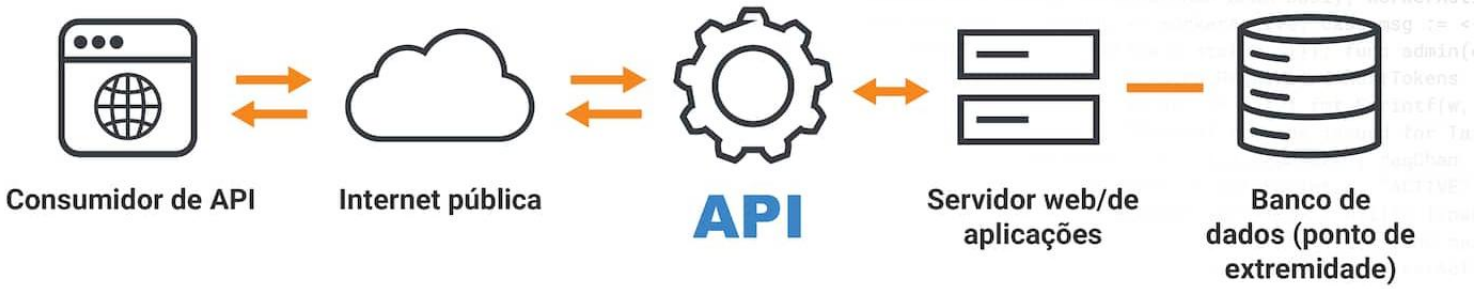
Matheus Moresco
Engenharia de Software – 3 Período
2025/01

Introdução

- Entender o que é uma API e como ela funciona.
- Aprender a consumir APIs usando fetch e axios.
- Manipular dados JSON e exibir resultados no DOM.
- Lidar com erros e requisições assíncronas.

O que é uma API?

- **API (Application Programming Interface)** é um conjunto de regras que permite que diferentes sistemas ou softwares conversem entre si.



Exemplo No Mundo Real

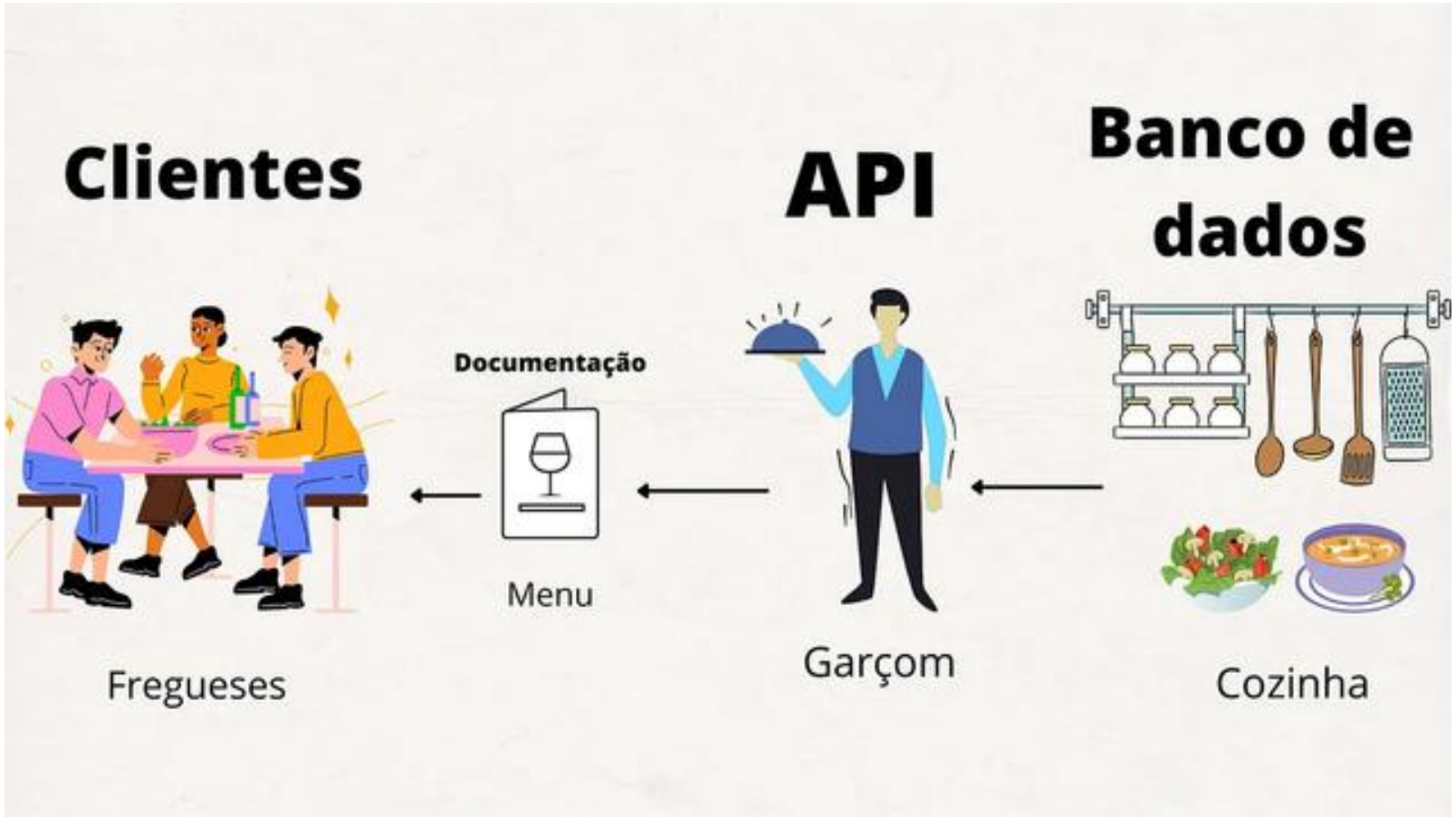
Imagine que você vai a um restaurante.

- Você quer pedir uma comida, mas não pode entrar na cozinha.
- Então você olha o cardápio, que mostra o que pode ser pedido.
- Você chama o garçom, faz o pedido com base no cardápio.
- O garçom vai até a cozinha, pega a comida e traz de volta pra você.

Nesse caso:

- **Você** = Aplicação (navegador, app, site...)
- **Cozinha** = Servidor onde estão os dados ou funcionalidades
- **Garçom** = API, que faz a ponte entre você e a cozinha
- **Cardápio** = Documentação da API, mostrando o que pode ser pedido

Exemplo No Mundo Real



Como funciona uma API?

- Cliente faz uma **requisição HTTP** para a API
- API processa e responde com dados (geralmente em JSON)
- A aplicação usa esses dados para exibir algo no frontend

Como uma API se comporta

Uma API geralmente funciona com requisições HTTP, como:

- **GET** → pegar dados (ex: lista de usuários, produtos, etc)
- **POST** → enviar dados (ex: cadastrar um novo usuário)
- **PUT** ou **PATCH** → atualizar dados
- **DELETE** → apagar algo

Tipos de API – Forma de acesso

- **APIs Públicas** (ou abertas)
  - Qualquer um pode usar. Geralmente gratuitas (ou com plano gratuito limitado).
  - Exemplo: PokeAPI, JSONPlaceholder, The Cat API
- **APIs Privadas**
  - Só podem ser usadas dentro de uma empresa/sistema.
  - Ex: API interna de um banco para acessar dados de clientes
- **APIs Parceiras**
  - Compartilhadas entre empresas parceiras.
  - Ex: API entre um app de delivery e um sistema de restaurante.
- **APIs Comerciais (ou pagas)**
  - Requerem chave de acesso (API key).
  - Planos pagos por número de requisições.
  - Ex: Google Maps API, OpenWeatherMap, Twitter API

Tipos de API – Aquitetura

- **REST (ou RESTful APIs)**
  - Usa HTTP e URLs organizadas por recursos.
  - Métodos: GET, POST, PUT, DELETE.
  - Muito usadas em sistemas web.
  - Ex: GET /users/1 → retorna usuário com ID 1
- **SOAP (Simple Object Access Protocol)**
  - Baseada em XML.
  - Mais rígida e pesada.
  - Muito usada em sistemas legados e corporativos (bancos, governo).

Tipos de API – Aquitetura

- **GraphQL**
  - Criada pelo Facebook.
  - Você define exatamente **quais campos quer receber**.
  - Útil para otimizar chamadas em apps modernos (ex: mobile).
- **gRPC**
- Criada pelo Google.
- Usa Protobuf (formato binário, mais rápido que JSON).
- Muito usada entre microsserviços.

API REST

REST significa **Representational State Transfer**.

É um estilo de arquitetura criado por Roy Fielding, com um conjunto de boas práticas para como sistemas devem se comunicar usando a web (HTTP).

Principais princípios do REST:

- **Cliente-servidor** → quem consome é separado de quem fornece os dados.
- **Sem estado** (stateless) → cada requisição traz todas as informações necessárias.
- **Cacheável** → as respostas podem ser armazenadas (cache).Interface uniforme → uso consistente de URLs e métodos HTTP (GET, POST, etc).
- **Baseado em recursos** → URLs representam objetos (ex: /users, /products).
- **Camadas** → a comunicação pode passar por várias camadas (servidores intermediários).

Uma **API RESTful** é uma API que **aplica corretamente os princípios REST**.

Ou seja: se a API segue as regras do REST, ela é chamada de RESTful.

Diferença: REST x RESTful

REST = conjunto de princípios de arquitetura

RESTful = API que segue corretamente esses princípios

- Exemplo RESTful:
  - GET /users → lista
  - GET /users/1 → detalhe
  - POST /users → cria
  - DELETE /users/1 → remove
- Exemplo não RESTful:
  - GET /getUser?id=1
  - POST /createUser
  - POST /deleteUser

Consumo de APIs no JavaScript

Em JavaScript moderno, existem **duas formas principais** de consumir APIs:

1. Usando **fetch**()
   - fetch() é uma função nativa do JavaScript para fazer requisições HTTP.

2. Usando **Axios** (biblioteca externa)
   - Axios é uma **biblioteca** que facilita as requisições HTTP.

Usando o fetch

- A função de fetch em JavaScript fax uma requisição para uma api de um determinado servidor, a partir de um endereço.
- É uma função assíncrona, logo, devemos usar o marcador **await** para determinar que o código precisa esperar a resposta do servidor.
- A função retorna um objeto do tipo Response

Usando o fetch

- Podemos usar o fetch de duas maneiras distintas:
- Com **async/await**:

```
async function carregarDados() {
    try {
      const res = await fetch('https://jsonplaceholder.typicode.com/users');
      const data = await res.json();
      console.log(data);
    } catch (err) {
      console.error('Erro ao carregar:', err);
    }
  }
```

- Com **.then():**

```
fetch('https://jsonplaceholder.typicode.com/users')
  .then(res => res.json())
  .then(data => console.log(data))
  .catch(err => console.error('Erro:', err));
```

Usando o Axios

- Axios é uma **biblioteca** que facilita as requisições HTTP.
  - Resposta já vem como JSON
  - Melhor tratamento de erros
  - Cancelamento de requisições
  - Função de TimeOut
  - Requisições simultâneas com
  - Facilita a mudança do tipo de métodos

- Precisa ser instalado:
  - Com Ou com npm/yarn: ***npm install axios***

  - Com CDN:

```
<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
```

Usando o Axios

- Podemos usar o Axios de duas maneiras distintas:
- Com **async/await**:

```
async function buscarUsuarios() {
    try {
      const resposta = await axios.get('https://jsonplaceholder.typicode.com/users');
      console.log(resposta.data);
    } catch (erro) {
      console.error('Erro ao buscar usuários:', erro);
    }
  }
```

- Com **.then():**

```
axios.get('https://jsonplaceholder.typicode.com/users')
  .then(res => console.log(res.data))
  .catch(err => console.error('Erro:', err));
```

HTML + Consumo de APIs

- Listar elementos na tela automaticamente.

```
<ul id="lista"></ul>

<script>
  fetch('https://pokeapi.co/api/v2/pokemon?limit=5')
    .then(res => res.json())
    .then(data => {
      const lista = document.getElementById('lista');
      data.results.forEach(pokemon => {
        const li = document.createElement('li');
        li.textContent = pokemon.name;
        lista.appendChild(li);
      });
    });
</script>
```

Boas práticas

- Sempre tratar erros com try/catch
- Usar async/await para legibilidade
- Testar APIs com Postman ou Insomnia
- Ler a documentação da API sempre

Exemplo prático

- Explorar APIS publicas
- Usar uma API para listar itens na tela