








Manipulação de APIs com Fetch

Prof. Bruno Torres


Objetivos

-  Entender o que é uma API
-  Usar `fetch()` para consumir dados
-  Trabalhar com JSON
-  Tratar erros de requisição
-  Criar um projeto usando API pública

O que é uma API?

- API = Application Programming Interface
- Ponto de comunicação entre aplicações
- APIs Web: enviam e recebem dados pela internet




Exemplo de API pública

- <https://jsonplaceholder.typicode.com/>
-  API de teste com dados simulados (usuários, posts etc.)

Como funciona o ciclo de uma requisição?

1. O navegador faz uma requisição HTTP para a API
2. A API processa e retorna uma resposta (geralmente em JSON)
3. O JavaScript interpreta e usa esses dados no site

O que é JSON?

-  JSON = JavaScript Object Notation
-  Formato leve para troca de dados
-  Fácil de ler e escrever para humanos e máquinas

```
{  
  "nome": "Bruno",  
  "idade": 25  
}
```

Diferença entre métodos HTTP

Método	Uso principal
GET	Buscar dados
POST	Enviar dados
PUT	Atualizar dados
DELETE	Apagar dados

Introdução ao `fetch()`

```
fetch("https://api.exemplo.com/dados")  
  .then(res => res.json())  
  .then(data => console.log(data));
```




Resposta como JSON

```
fetch("http://localhost:3000/users")  
  .then(res => res.json())  
  .then(usuarios => {  
    console.log(usuarios);  
  });
```




Tratando erros

```
fetch(url)
  .then(res => {
    if (!res.ok) throw new Error("Erro na requisição");
    return res.json();
  })
  .catch(err => console.error(err));
```

Async/Await com Fetch

```
async function carregarDados() {  
  try {  
    const res = await fetch(url);  
    const dados = await res.json();  
    console.log(dados);  
  } catch (e) {  
    console.error("Erro:", e);  
  }  
}
```

Lidando com erros de rede e API

-  Sempre use `try/catch` ou `.catch()`
-  Exiba mensagens amigáveis para o usuário
-  Exemplo: mostrar um alerta se a API estiver fora do ar



Exibindo no DOM

```
dados.forEach(usuario => {  
  const li = document.createElement("li");  
  li.textContent = usuario.name;  
  lista.appendChild(li);  
});
```



Projeto: Cadastrando usuários com Fetch

Vamos criar um pequeno sistema para cadastrar usuários, enviando o nome para uma API e exibindo a lista de usuários cadastrados.

⚙️ Configurando a API simulada com json-server

- Os dados serão **salvos** no arquivo `db.json` pelo json-server.
- O arquivo `db.json` deve conter inicialmente uma lista chamada `users`:

```
{  
  "users": [  
    { "id": 1, "name": "Bruno" },  
    { "id": 2, "name": "Maria" }  
  ]  
}
```

⚙️ Configurando a API simulada com json-server

- Para rodar a API simulada, use:

```
npm install -g json-server  
json-server --watch db.json --port 3000
```

- O endpoint será `http://localhost:3000/users` .
- Sempre trate erros com `try/catch` para evitar problemas em produção.

1 Estrutura HTML do formulário

Crie um formulário simples com um campo de texto para o nome e um botão para cadastrar. Abaixo, uma lista exibirá os usuários.

```
<form id="formUsuario">  
  <input type="text" id="nome" placeholder="Digite o nome" required />  
  <button type="submit">Cadastrar</button>  
</form>  
<ul id="listaUsuarios"></ul>
```

2 Função para cadastrar usuário (POST)

Vamos criar uma função que será chamada ao enviar o formulário. Ela irá:

- Impedir o recarregamento da página
- Pegar o nome digitado
- Enviar para a API usando `fetch` com método POST
- Limpar o campo de texto
- Atualizar a lista de usuários

2 Função para cadastrar usuário (POST) - Usando json-server

```
document.getElementById("formUsuario").addEventListener("submit", cadastrarUsuario);

async function cadastrarUsuario(event) {
  event.preventDefault(); // Impede o reload da página

  const nome = document.getElementById("nome").value;

  // Envia o nome para a API local (json-server)
  await fetch("http://localhost:3000/users", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ name: nome })
  });

  document.getElementById("nome").value = ""; // Limpa o campo
  carregarUsuarios(); // Atualiza a lista
}
```

3 Função para carregar e exibir usuários (GET)

Agora, vamos buscar os usuários da API local e mostrar na lista.

```
async function carregarUsuarios() {  
  const res = await fetch("http://localhost:3000/users");  
  const usuarios = await res.json();  
  
  const lista = document.getElementById("listaUsuarios");  
  lista.innerHTML = ""; // Limpa a lista antes de exibir  
  
  usuarios.forEach(u => {  
    const li = document.createElement("li");  
    li.textContent = u.name;  
    lista.appendChild(li);  
  });  
}  
carregarUsuarios();
```

4 Resumo do fluxo

1. Usuário digita o nome e clica em "Cadastrar"
2. O JS envia o nome para a API com POST
3. Após cadastrar, o JS busca todos os usuários (GET)
4. A lista é atualizada na tela



Exemplo: Exibindo detalhes do usuário

```
async function carregarDetalhes(id) {  
  const res = await fetch(`http://localhost:3000/users/${id}`);  
  const usuario = await res.json();  
  alert(`Nome: ${usuario.name}\nEmail: ${usuario.email}`);  
}
```



Métodos HTTP comuns

- GET → buscar dados
- POST → enviar dados
- PUT → atualizar dados
- DELETE → apagar dados






Dica: usar `console.log()`

- 👁 Verifique a estrutura dos dados
- 🔍 Entenda como acessar os campos
- 📊 Use `console.table()` para visualizar listas

Recursos para aprender mais

- [MDN Web Docs: Fetch API](#)
- [json-server \(simulador de API\)](#)
- [Documentação HTTP](#)

Conclusão

-  `fetch()` é a base para comunicação com APIs
-  JSON é o formato mais comum
-  Sempre trate erros para evitar bugs

 **Obrigado!**

Dúvidas? Vamos praticar!