



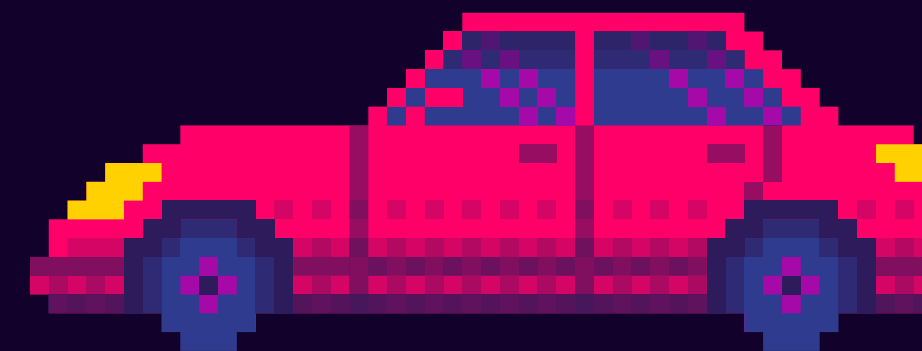
FETCH API

O QUE É A FETCH API?

A Fetch API é uma interface moderna para realizar requisições HTTP assíncronas (como GET, POST, PUT, DELETE) no navegador. Substitui o antigo objeto XMLHttpRequest, sendo mais fácil de usar e oferecendo uma sintaxe mais limpa baseada em promises.

Vantagens da Fetch API:

- Mais simples e legível que XMLHttpRequest.
- Baseada em promises, facilitando o uso de código assíncrono.
- Suporte nativo à maioria dos navegadores modernos.



ESTRUTURA BÁSICA DE UMA REQUISIÇÃO

Uma requisição básica com `fetch()` tem a seguinte estrutura:

```
fetch(url, options)
  .then(response => {
    // Processa a resposta
  })
  .catch(error => {
    // Trata erros
  });
```

- `url`: A URL para onde a requisição será enviada.
- `options`: Um objeto opcional com configurações como método, headers, body, etc.



REQUISIÇÃO GET

A requisição GET é usada para obter dados de um servidor. Neste exemplo, vamos buscar dados de um JSON público:

1. `fetch()` faz a requisição para a URL.
2. O método `.ok` verifica se a resposta foi bem-sucedida (status 200-299).
3. `response.json()` transforma os dados em formato JSON.
4. O resultado é manipulado no próximo `.then()`.
5. O `.catch()` trata erros, como problemas de rede.

```
fetch('https://jsonplaceholder.typicode.com/posts')
  .then(response => {
    // Verifica se a resposta foi bem-sucedida
    if (!response.ok) {
      throw new Error('Erro na requisição: ' + response.status);
    }
    return response.json(); // Converte o JSON da resposta
  })
  .then(data => {
    console.log(data); // Manipula os dados recebidos
  })
  .catch(error => {
    console.error('Erro:', error);
  });
```

REQUISIÇÃO POST

A requisição POST é usada para enviar dados ao servidor. Veja um exemplo de como enviar um objeto JSON:

1. method: 'POST' define o método HTTP da requisição.
2. headers define que estamos enviando dados no formato JSON.
3. body é o corpo da requisição, que deve ser transformado em string usando `JSON.stringify()`.
4. A resposta também é convertida em JSON para facilitar a manipulação.

```
const newPost = {  
  title: 'Novo Post',  
  body: 'Este é o conteúdo do post.',  
  userId: 1  
};  
  
fetch('https://jsonplaceholder.typicode.com/posts', {  
  method: 'POST', // Especifica o método  
  headers: {  
    'Content-Type': 'application/json' // Tipo de conteúdo sendo enviado  
  },  
  body: JSON.stringify(newPost) // Converte o objeto em string JSON  
)  
  .then(response => response.json()) // Transforma a resposta em JSON  
  .then(data => console.log(data)) // Exibe a resposta  
  .catch(error => console.error('Erro:', error));
```

ERROS COMUNS

01

CORS: Algumas APIs podem bloquear requisições vindas de domínios diferentes (cross-origin). Isso é chamado de erro de CORS. Esse tipo de erro não pode ser resolvido no front-end, é preciso que o servidor permita a requisição.

02

Verificação da resposta: Nem toda resposta HTTP com status 200 significa sucesso total. Verifique sempre se os dados são os esperados.





SINTAXE ASSÍNCRONA COM ASYNC/AWAIT

Para deixar o código mais limpo e fácil de entender, podemos usar async/await:

```
async function getPosts() {  
  try {  
    const response = await fetch('https://jsonplaceholder.typicode.com/posts');  
    if (!response.ok) {  
      throw new Error('Erro na requisição: ' + response.status);  
    }  
    const data = await response.json();  
    console.log(data);  
  } catch (error) {  
    console.error('Erro:', error);  
  }  
}  
  
getPosts();
```

01

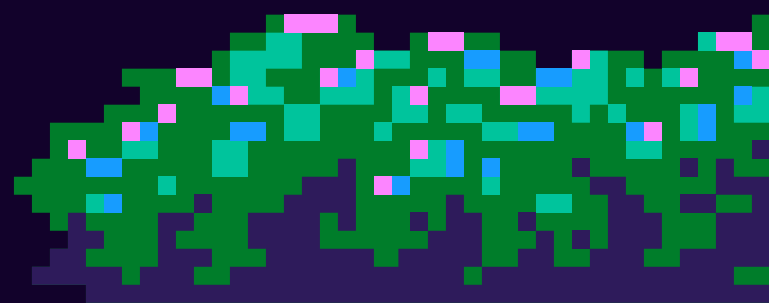
função é marcada como async para permitir o uso de await.

02

await pausa a execução até que a promessa seja resolvida.

03

O bloco try...catch trata os erros de forma mais clara, facilitando a depuração.



EXERCÍCIO PRÁTICO

01

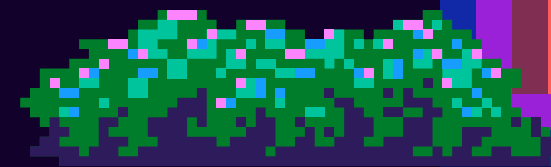
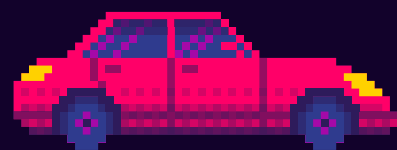
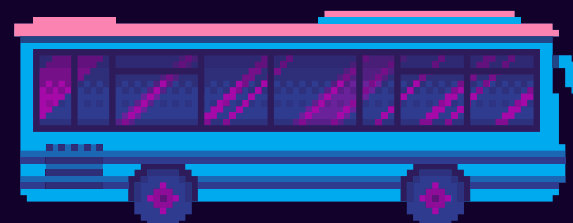
Faça uma requisição GET para uma API pública de sua escolha e exiba os dados recebidos na página.

02

Implemente um formulário que permite ao usuário enviar dados para a mesma API usando uma requisição POST, e exiba a resposta na tela.

Dicas:

- Use `fetch()` para as requisições.
- Manipule erros adequadamente.
- Explore headers e opções da requisição.





THANK
YOU