Consumo de API com JavaScript

Por Escola Dnc

Introdução

Bem-vindos ao nosso módulo de consumo de API com JavaScript! Neste ebook, trarei para vocês o máximo de informações possíveis e da forma mais clara que puder sobre como fazer uma requisição e utilizar uma API para pegar informações de lá e consumir essa informação no front-end.

Além disso, vamos aprender junto uma ferramenta muito útil chamada Bootstrap, que nos permite montar um front-end utilizando alguns componentes já prontos. Não será algo muito robusto, mas veremos algumas facilidades que o Bootstrap oferece. E o legal é que tudo que vocês aprenderem sobre Bootstrap poderá ser aplicado em qualquer outro framework moderno como React, Angular ou Vue.

O que vamos aprender?

Os principais tópicos que veremos são:

- Como fazer requisições a APIs
- Entender as diferenças entre código síncrono e assíncrono em JavaScript
- Utilizar o Bootstrap para montar interfaces
- Consumir e exibir dados de APIs meteorológicas e de CEP
- Boas práticas de SEO e Markdown

Requisições a APIs

Antes de mais nada, o que é uma API?

API significa Application Programming Interface, ou Interface de Programação de Aplicações em português. Uma API é um conjunto de rotinas, protocolos e ferramentas que permite a integração entre diferentes sistemas e aplicações.

As APIs permitem que serviços e dados que estão armazenados em um lugar possam ser acessados e manipulados por outra aplicação de forma padronizada e controlada.

Em outras palavras, uma API serve como uma ponte entre diferentes sistemas, permitindo a comunicação e troca de informações entre eles.

Quando fazemos uma requisição a uma API, estamos fazendo uma solicitação para obter ou enviar algum dado. Por exemplo, quando usamos o Facebook, fazemos várias requisições à API do Facebook para obter nosso feed de notícias, enviar novos posts, etc.

Para consumir uma API, utilizamos o protocolo HTTP através de requisições feitas via JavaScript. As principais são:

GET: Solicita a representação de um recurso específico. É tipicamente usado para recuperar informações.

POST: Envia dados para serem processados pelo recurso identificado da API. Geralmente usado para criar novos registros.

PUT: Atualiza dados de um recurso especificado pela API.

DELETE: Remove um recurso disponibilizado pela API.

Vamos aprender na prática como fazer requisições GET para obter dados de APIs públicas sobre clima e CEP.

JavaScript Síncrono vs Assíncrono

Antes de vermos como consumir APIs, precisamos entender como o JavaScript lida com código síncrono e assíncrono.

O código síncrono é executado linha a linha, de cima para baixo. Cada linha de código só é executada depois que a linha anterior terminou sua execução. É um fluxo linear, onde tudo acontece em uma determinada ordem.

Já o código assíncrono não precisa esperar uma linha terminar para executar a próxima. As linhas são executadas de forma não linear, possibilitando operações simultâneas.

Isso permite que enquanto uma operação mais demorada acontece em background (como buscar dados de uma API), outras partes do código possam ser executadas normalmente. Quando os dados ficam prontos, o código assíncrono avisa que pode ser consumido.

Em JavaScript, funções como setTimeout() e requisições à APIs (que veremos mais à frente) são operações assíncronas. Elas iniciam uma operação que pode levar algum tempo, mas permitem que outras linhas do código continuem executando normalmente.

Entender a diferença entre código síncrono e assíncrono é muito importante para evitar bugs e poder criar aplicações mais responsivas.

Bootstrap

O Bootstrap é um framework front-end muito popular, que fornece uma série de componentes e estilos pré-definidos para criarmos interfaces de forma rápida e responsiva.

Alguns dos benefícios do Bootstrap são:

- Responsividade nativa, funcionando bem em dispositivos mobile
- Grande variedade de componentes reutilizáveis como botões, menus, forms, modal e etc
- Estilos e ícones elegantes que economizam nosso tempo
- Compatibilidade com praticamente todos frameworks modernos como React, Angular e Vue
- Comunidade enorme, com grande número de recursos disponíveis

Iremos utilizar algumas funcionalidades básicas do Bootstrap nesse módulo, como containers, rows, columns, cards e forms. Tudo que vocês aprenderem pode ser reaplicado facilmente em projetos mais complexos no futuro.

Consumindo uma API de Clima

Chegou a hora de colocarmos a mão na massa!

Primeiro, vamos consumir uma API pública que nos retorna dados meteorológicos. A documentação dela está disponível em:

https://www.weatherapi.com

Não se preocupem em entender todos os detalhes da documentação agora. O que importa é sabermos que essa API disponibiliza dados do clima, podendo filtrar por cidade e estado, e nos retorna um JSON com várias informações meteorológicas.

Vamos criar um arquivo index.html básico:

```
<!DOCTYPE html><html lang="pt-BR"><head>  <meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-
scale=1.0">  <title>Clima Tempo</title></head><body>  <script
src="app.js"></script></body></html>
```

E também um arquivo app.js, que irá conter todo nosso código JavaScript.

Começaremos fazendo uma requisição GET na API passando São Paulo como cidade e SP como estado:

```
fetch(`https://api.weatherapi.com/v1/current.json?key=SUA_KEY&q=São
Paulo&aqi=no`)  .then(response => response.json())  .then(data => {
console.log(data);   });
```

Explicando o código:

- Utilizamos o fetch para fazer a requisição HTTP GET na API
- Passamos a key de acesso (cada um deve usar a própria)
- Indicamos São Paulo,SP como localização
- Na primeira arrow function, convertamos a resposta em JSON
- Na segunda, exibimos os dados no console

Se tudo ocorrer bem, você verá um grande objeto JSON com todas as informações meteorológicas sendo printado no seu console.

Agora, vamos exibir essas informações visualmente:

```
function exibirClima(data) {  let temperatura = document.createElemen
temperatura.innerText = `Temperatura: ${data.current.temp_c} graus`;
document.createElement('p');  umidade.innerText = `Umidade: ${data.cu
let visibilidade = document.createElement('p');  visibilidade.innerTe
`Visibilidade: ${data.current.vis_km} Km`;  let container =
document.querySelector('body');  container.appendChild(temperatura);
container.appendChild(umidade);
container.appendChild(visibilidade);}fetch(`https://api.weatherapi.co
key=SUA_KEY&q=São Paulo&aqi=no`)  .then(response => response.json())
exibirClima(data));
```

Criamos uma função exibirClima() que recebe os dados da API e cria elementos p com as informações de temperatura, umidade e visibilidade. Essas tags são inseridas no body para serem renderizadas.

Perceba que nosso código está assíncrono. A requisição fetch é iniciada, mas enquanto aguardamos os dados, a função exibirClima() continua disponível para ser executada depois, quando os dados chegarem.

Interface com Bootstrap

Até agora os dados são exibidos de forma básica, sem nenhuma estilização. Vamos deixar nossa aplicação mais bonita usando o Bootstrap?

Primeiro precisamos adicionar a biblioteca CSS do Bootstrap no head:

```
<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.3.1/dist/css/bootstrap
integrity="sha384-
ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">
```

Depois vamos refatorar nosso HTML:

```
<div class="container mt-5">  <div class="row">    <div class="col-
md-4">      <div id="climate" class="card">        <div
class="card-body">          <!-- Dados serão renderizados aqui -->
</div>      </div>    </div>    <div class="col-md-8">      <form>
<div class="form-group">          <input type="text"
placeholder="Digite uma cidade" class="form-control">
</div>        <button type="submit" class="btn btn-
primary">Buscar</button>      </form>    </div>  </div></div>
```

Ficou bem melhor! Adicionamos:

- Container e rows do Bootstrap para responsividade
- Columns para dividir em grids
- Um card para exibir os dados
- Formulário de busca por cidades

Agora precisamos renderizar os dados dentro do card. Vamos alterar a função exibirClima():

```
function exibirClima(data) {  let card =
document.getElementById('climate');  let cidade =
document.createElement('h4');  cidade.innerText =
data.location.name;  let img = document.createElement('img');
img.src = data.current.condition.icon;  let temperatura =
document.createElement('p');  temperatura.innerText = `Temperatura:
${data.current.temp_c} °C`;  let sensacao =
document.createElement('p');  sensacao.innerText = `Sensação
térmica: ${data.current.feelslike_c} °C`;  card.innerHTML = ''; //
Limpa dados antigos  card.appendChild(cidade);
card.appendChild(img);   card.appendChild(temperatura);
card.appendChild(sensacao);}
```

Agora sim! Estamos exibindo os dados de forma mais elegante e com Bootstrap. Aproveitamos e adicionamos mais algumas informações como imagem do tempo atual e sensação térmica.

Consumindo uma API de CEP

Agora que sabemos consumir APIs, usar Bootstrap e exibir dados, vamos praticar com outra API.

Desta vez utilizaremos uma API que, passando um CEP, nos retorna endereço completo, cidade, estado e outras informações.

Documentação:https://viacep.com.br/

Começaremos fazendo a requisição:

```
fetch(`https://viacep.com.br/ws/01001000/json/`)  .then(res =>
res.json())  .then(data => {    console.log(data);  });
```

Teste com seu próprio CEP. Você deve ver um objeto com rua, bairro, cidade e estado sendo printado.

Agora criaremos uma interface para digitar o CEP e exibir essas informações.

No HTML:

```
<div class="container mt-5">  <div class="row">    <div class="col-
md-6">      <h3>Busca CEP</h3>      <form>        <div class="form-
group">          <input id="cep" type="text" placeholder="Digite o
CEP" class="form-control">          </div>        <button
type="submit" class="btn btn-primary">Buscar</button>      </form>
<div id="address">        <!-- Endereço será renderizado aqui -->
</div>    </div>  </div>  </div>
```

E no JavaScript:

```
const form = document.querySelector('form');const inputCep =
document.getElementById('cep');const address =
document.getElementById('address');form.addEventListener('submit',
(e) => {  e.preventDefault();
fetch(`https://viacep.com.br/ws/${inputCep.value}/json/`)
.then(res => res.json())      .then(data => {
if(!data.erro) {                let rua =
document.createElement('p');        rua.innerText =
data.logradouro;        let bairro = document.createElement('p');
bairro.innerText = data.bairro;        let cidade =
document.createElement('p');        cidade.innerText =
data.localidade;        let estado = document.createElement('p');
estado.innerText = data.uf;                address.innerHTML = '';
address.appendChild(rua);        address.appendChild(bairro);
address.appendChild(cidade);        address.appendChild(estado);
}    });});
```

Estamos consumindo a API dentro do submit do form, pegando o CEP digitado e limpando o container sempre que uma nova busca for realizada.

Assim construímos de forma rápida uma aplicação fullstack consumindo API!

Boas práticas de SEO e Markdown

Para finalizar, algumas dicas rápidas de SEO e Markdown que podem ajudar na divulgação do seu projeto:

SEO

- Escolha títulos descritivos das páginas
- Utilize textos claros e objetivos
- Inclua palavras-chave nos textos
- Otimize as imagens com textos alternativos
- Crie um site responsivo

Markdown

O Markdown é uma linguagem simples de marcação que converte texto em HTML. Com alguns caracteres já conseguimos formatar títulos, listas, código e outros elementos.

Alguns exemplos:

```
# Título ## Subtítulo- Item 1- Item 2`Código` **Negrito**[Link](https://www.example.com)
```

Isso já é o suficiente para começar! Depois você pode explorar recursos mais avançados.

Considerações Finais

Espero que você tenha gostado desse material completo sobre como consumir APIs com JavaScript. Vimos na prática os conceitos de código assíncrono, requisições HTTP, Bootstrap para interface e diferentes formas de exibir dados dinâmicos.

Há ainda muito o que explorar, esse foi apenas o começo. Conforme você for estudando mais, poderá criar aplicações web inteiras utilizando frameworks modernos e APIs, criando soluções para problemas reais.

Qualquer dúvida, deixe nos comentários!