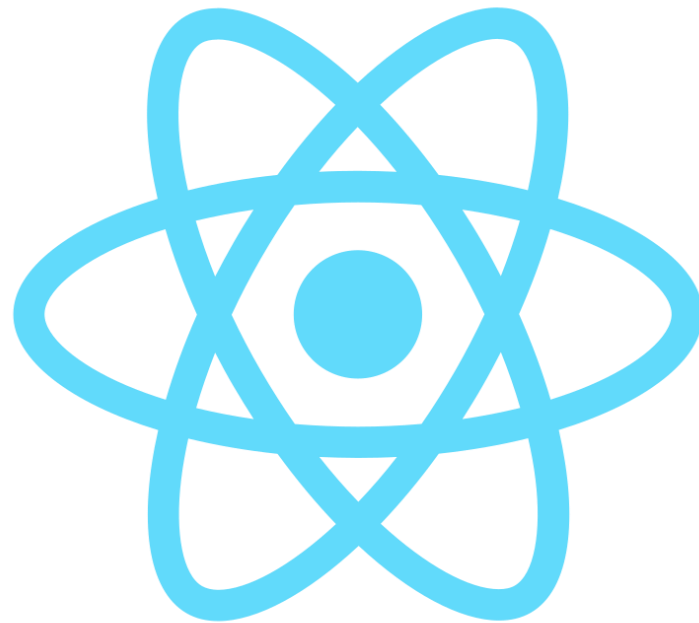


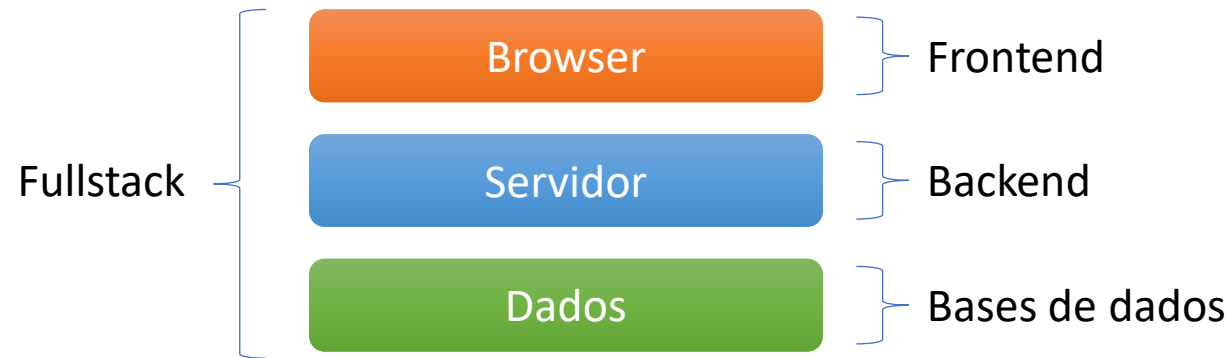
Introdução a React

Por Osvaldo Santos



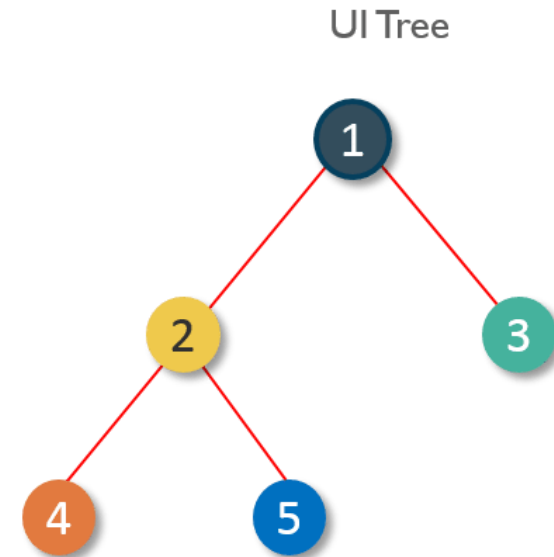
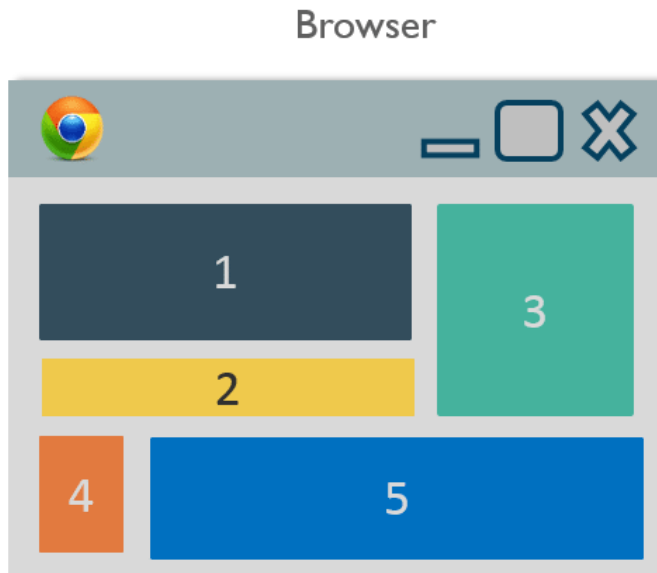
Versão 1.0, nov 2021

Desenvolvimento Web moderno



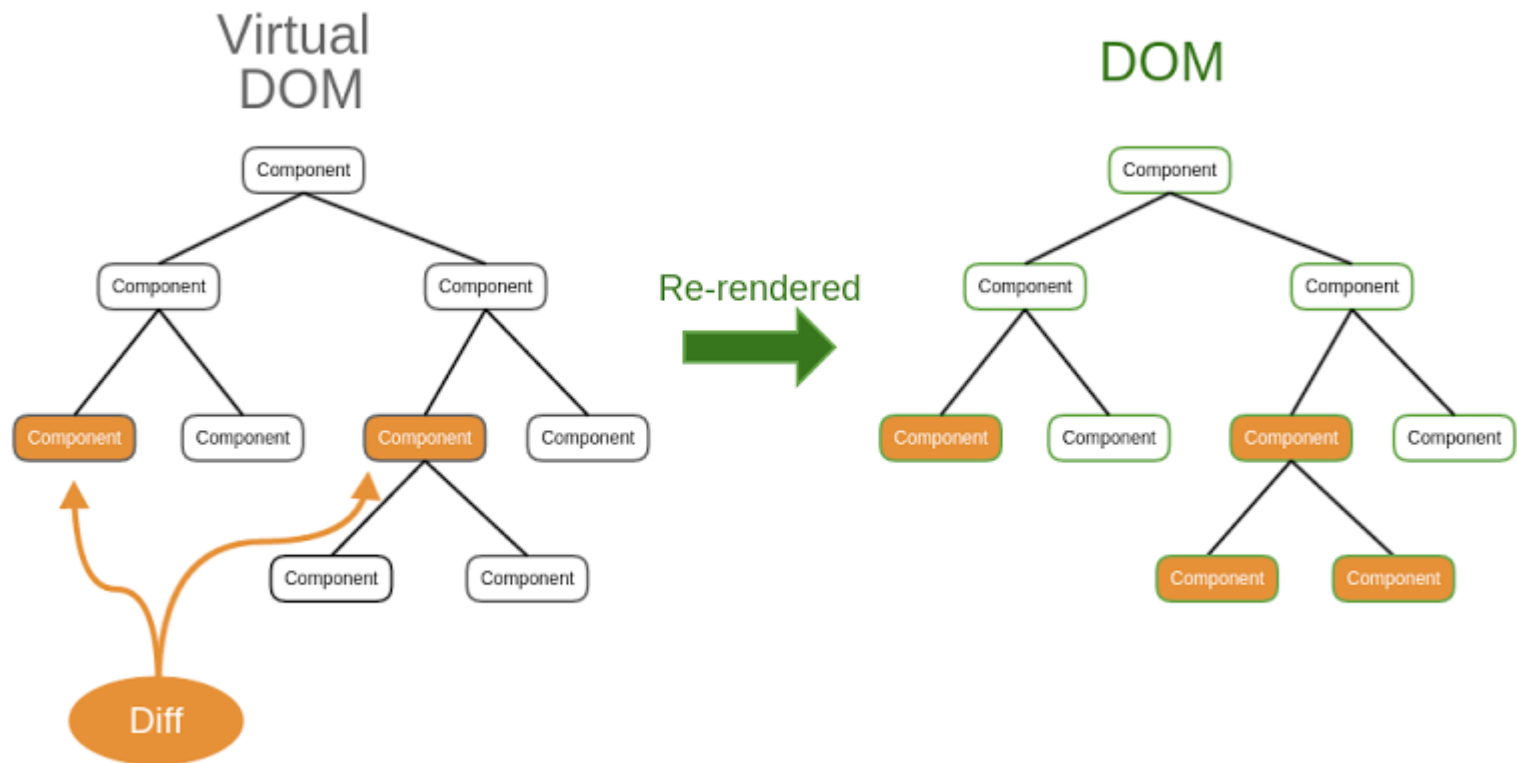
Biblioteca React

- React é uma biblioteca Javascript de código aberto para frontend, cada vez mais popular, usada em aplicações single-page
- Foi criada em 2011 por Jordan Walke, um engenheiro de software do Facebook e é usada em muitos websites conhecidos
- A interface web é dividida em várias partes independentes chamadas componentes, que podem ser reutilizados
- Cada componente representa um determinado estado e se esse estado mudar, a interface é renderizada automaticamente



React Virtual DOM

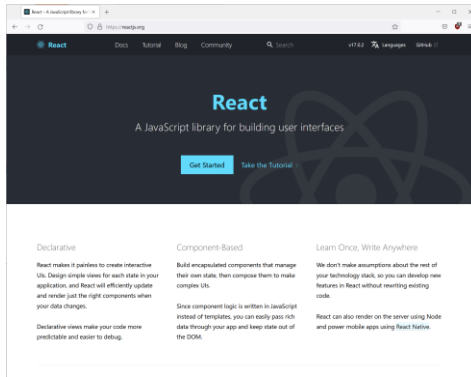
- React cria um virtual DOM onde faz todo o processamento e manipulação
- Depois compara o virtual DOM com o DOM real do browser e apenas faz as alterações no DOM real que correspondem às diferenças
- Isto torna a renderização da página muito rápida, mesmo que o virtual DOM tenha sido recriado de raiz



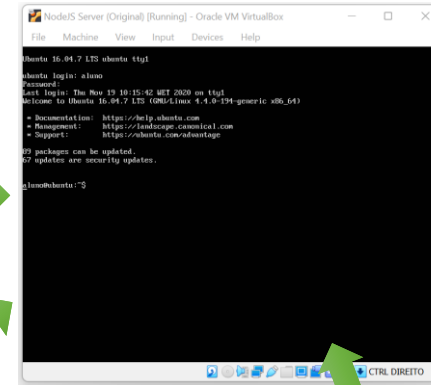
Fonte: Naukri Engineering

Ambiente de testes e aprendizagem

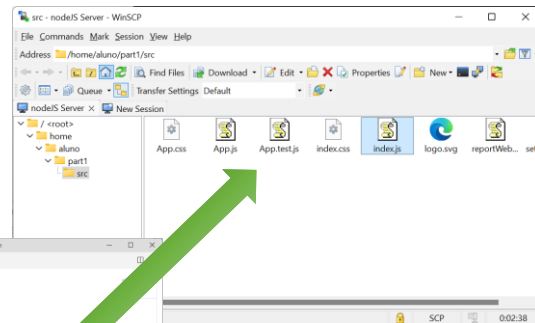
Browser



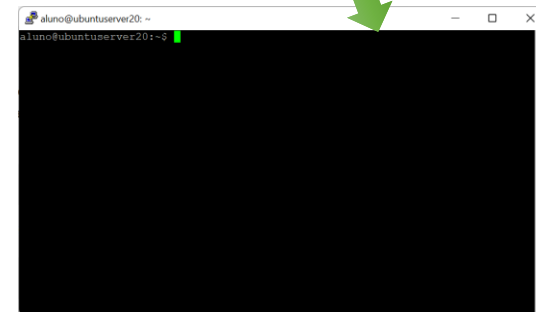
Ubuntu 20 Server VM



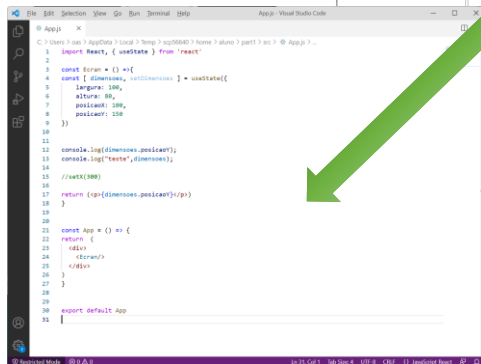
Node.JS



WinSCP



Putty

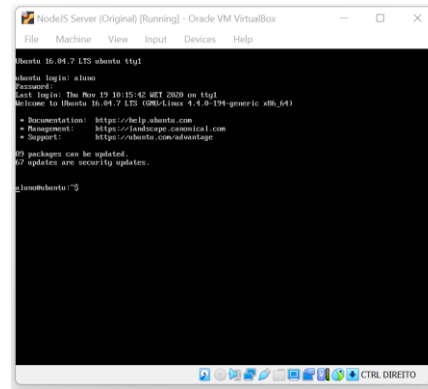


Visual Studio
Ou Notepad++

Instalação de Node.JS e NPM no Ubuntu

NPM = Node Package Manager

Ubuntu 20 Server VM



```
sudo apt update
```

```
curl -sL https://deb.nodesource.com/setup_14.x -o nodesource_setup.sh
```

```
sudo bash nodesource_setup.sh
```

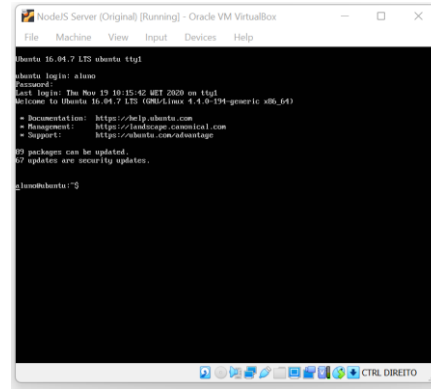
```
sudo apt install nodejs
```

```
sudo apt install npm
```

```
node -v #este comando deve mostrar a versão 14 ou superior
```

Hello World

Ubuntu 20 Server VM



```
$ npx create-react-app hello
$ cd hello
$ npm start
```

Define um componente
chamado App



App.js

```
import React from 'react'
const App = () => {

  return (
    <div>
      <p>Hello world</p>
    </div>
  )
}

export default App
```

Faz o render do componente
dentro do elemento html com o
id root



index.js

```
import ReactDOM from 'react-dom'
import App from './App'

ReactDOM.render(
  <App />,
  document.getElementById('root')
)
```

Incorporar javascript no JSX com { }

App.js

```
import React from 'react'

const App = () => {

  let data = new Date()

  return (
    <div>
      <p>A data atual é: {data.toString()}</p>
    </div>
  )

}

export default App
```


JSX - JavaScript XML

O JSX permite escrever HTML em React de uma forma simplificada

Sintaxe usando JSX

```
return (  
  <div>  
    <p>A data atual é: {data.toString()}</p>  
  </div>  
)
```

Sintaxe sem usar JSX

```
return React.createElement(  
  'div',  
  null,  
  React.createElement(  
    'p', null, 'A data atual é:', data.toString()  
  )  
)
```

JSX - Incorporar um componente

```
import React from 'react'

const Hello = () => {
  return (<p>Hello world</p>)
}

const App = () => {
  return (
    <div>
      <Hello />
      <Hello />
      <Hello />
    </div>
  )
}

export default App
```

Filosofia REACT: as aplicações Web são feitas à custa de componentes reutilizáveis

Modularidade dos componentes

- De modo a melhorar a modularidade do Código, cada componente deve ser implementado num ficheiro independente
- O componente deve depois ser importado
- O nome do ficheiro tem que começar por uma letra maiúscula

Ficheiro Hello.js (nota: tem que começar por letra maiúscula)

```
const Hello = () => {  
  return (<p>Hello world</p>)  
}  
  
export default Hello
```

```
import React from 'react';  
  
import ReactDOM from 'react-dom';  
  
import Hello from './Hello.js';
```

Passar dados para um componente

```
import React from 'react'

const Hello = (props) => {
  return (<p>Hello {props.name}</p>)
}

const App = () => {
  return (
    <div>
      <Hello name="Luis"/>
      <Hello name="Teresa"/>
      <Hello name="Iris"/>
    </div>
  )
}

export default App
```

Passar múltiplos dados

```
import React from 'react'

const Hello = (props) => {
  return (<p>Hello {props.name} from {props.country}</p>)
}

const App = () => {
  return (
    <div>
      <Hello name="Luis" country="Portugal"/>
      <Hello name="Teresa" country="Spain"/>
      <Hello name="Iris" country="France"/>
    </div>
  )
}

export default App
```

Regras sobre o JSX

- Tem que ser devolvido um único elemento raiz (usualmente uma div), ou então um array de componentes

```
const App = () => {  
  return (  
    <h1>Greetings</h1>  
    <Hello name="Maya" age={26 + 10} />  
    <Footer />  
  )  
}
```



```
const App = () => {  
  return (  
    <div>  
      <h1>Greetings</h1>  
      <Hello name="Maya" age={26 + 10} />  
      <Footer />  
    </div>  
  )  
}
```



Regras sobre o JSX

- Todos os elementos têm que ter etiqueta de fecho

```
const App = () => {  
  return (  
      
  )  
}
```



```
const App = () => {  
  return (  
    </img>  
  )  
}
```



Regras sobre o JSX

- Elementos html devem começar com letra minúscula
- Componentes React devem começar com uma letra maiúscula

```
const App = () => {  
  return (  
    <div>  
        
      <hello name="Iris"/>  
    </div>  
  )  
}
```



```
const App = () => {  
  return (  
    <div>  
        
      <Hello name="Iris"/>  
    </div>  
  )  
}
```



Regras sobre o JSX

- A palavra **class** é reservada em ES6, por isso não podemos usá-la no JSX. Deve ser substituída por **className**

```
const App = () => {  
  return (  
    </img>  
  )  
}
```

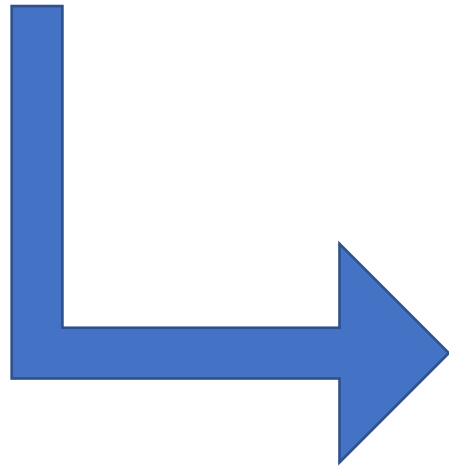


```
const App = () => {  
  return (  
    </img>  
  )  
}
```



Desestruturação : pode ser útil

```
pessoa = {  
  name: 'Carla Silva',  
  age: 25,  
};  
  
console.log("name: " + pessoa.name);  
console.log("age: " + pessoa.age);
```



```
pessoa = {  
  name: 'Carla Silva',  
  age: 25,  
};  
  
const { name, age } = pessoa;  
  
console.log("name: " + name);  
console.log("age: " + age);
```

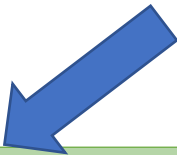
Processamento de eventos

```
const Clicar = () =>{  
  
  const handleClick = () => {  
    console.log("cliqueu com o rato")  
  }  
  
  return (  
  
    <p onClick={handleClick}>Clique aqui</p>  
  
  )  
  
}
```

Tornar os componentes stateful

- Por vezes é útil reutilizar um componente várias vezes na mesma página
- Cada instância do componente deve ter o seu próprio estado (variáveis, etc)
- É um pouco o que sucede com os vários objetos (instâncias) de uma classe em programação orientada a objetos
- O REACT tem um mecanismo para obter e alterar o estado de uma determinada instância de um componente – chama-se **State Hook**
- Um **Hook** é uma função especial que nos permite usar funcionalidades especiais do React

Importa a hook function **useState**



```
import React, { useState } from 'react'

const Clicar = () =>{
  const [ counter, setCounter ] = useState(0)
  const handleClick = () => { setCounter(counter+1) }

  ... Código omitido
```

Estado: exemplo completo

Nota: sempre que o estado de um componente muda, ele é renderizado

```
import React, { useState } from 'react'

const Clicar = () =>{
  const [ counter, setCounter ] = useState(0)
  const handleClick = () => { setCounter(counter+1) }
  return (<p onClick={handleClick}>Clicou aqui {counter} vezes !</p>)
}

const App = () => {
  return (
    <div>
      <Clicar/><Clicar/><Clicar/>
    </div>
  )
}

export default App
```

Estados mais complexos

- No exemplo anterior, o estado era apenas uma variável. Como podemos utilizar estados mais complexos, com múltiplas variáveis?
- A forma mais simples é usar a função **useState** várias vezes

```
import React, { useState } from 'react'

const Ecran = () => {
  const [ largura, setLargura ] = useState(300)
  const [ altura, setAltura ] = useState(80)
  const [ posicaoX, setX ] = useState(100)
  const [ posicaoY, setY ] = useState(150)

  console.log(posicaoX)

  ... Código omitido...

}
```

Estados mais complexos

- Não é obrigatório usar variáveis simples para o estado
- Também podem ser usados objetos
- Exemplo anterior, guardando o estado num objeto em vez de 4 variáveis independentes

```
import React, { useState } from 'react'

const Ecran = () => {
  const [ dimensoes, setDimensoes ] = useState({
    largura: 100,
    altura: 80,
    posicaoX: 100,
    posicaoY: 150
  })

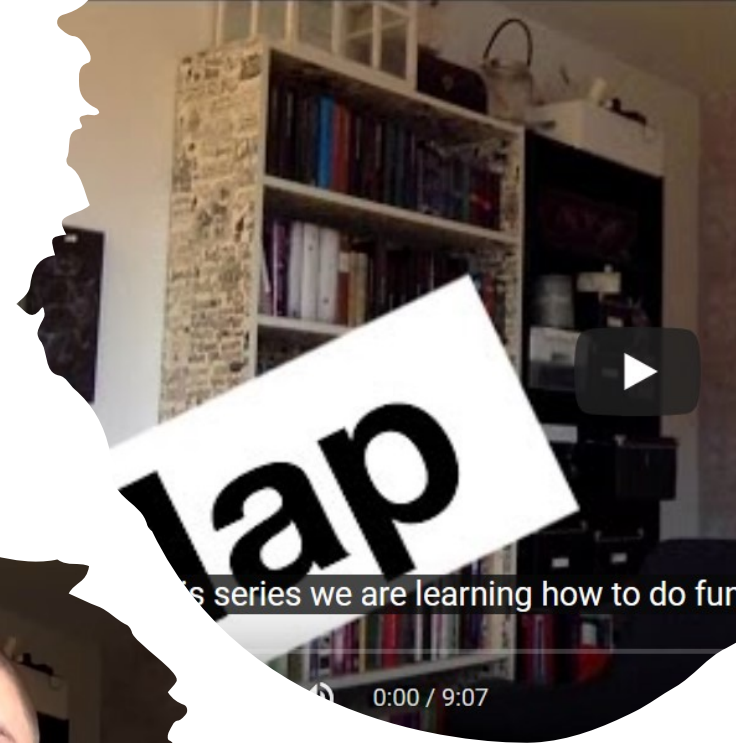
  console.log(dimensoes.posicaoX)

  ... Código omitido...

}
```

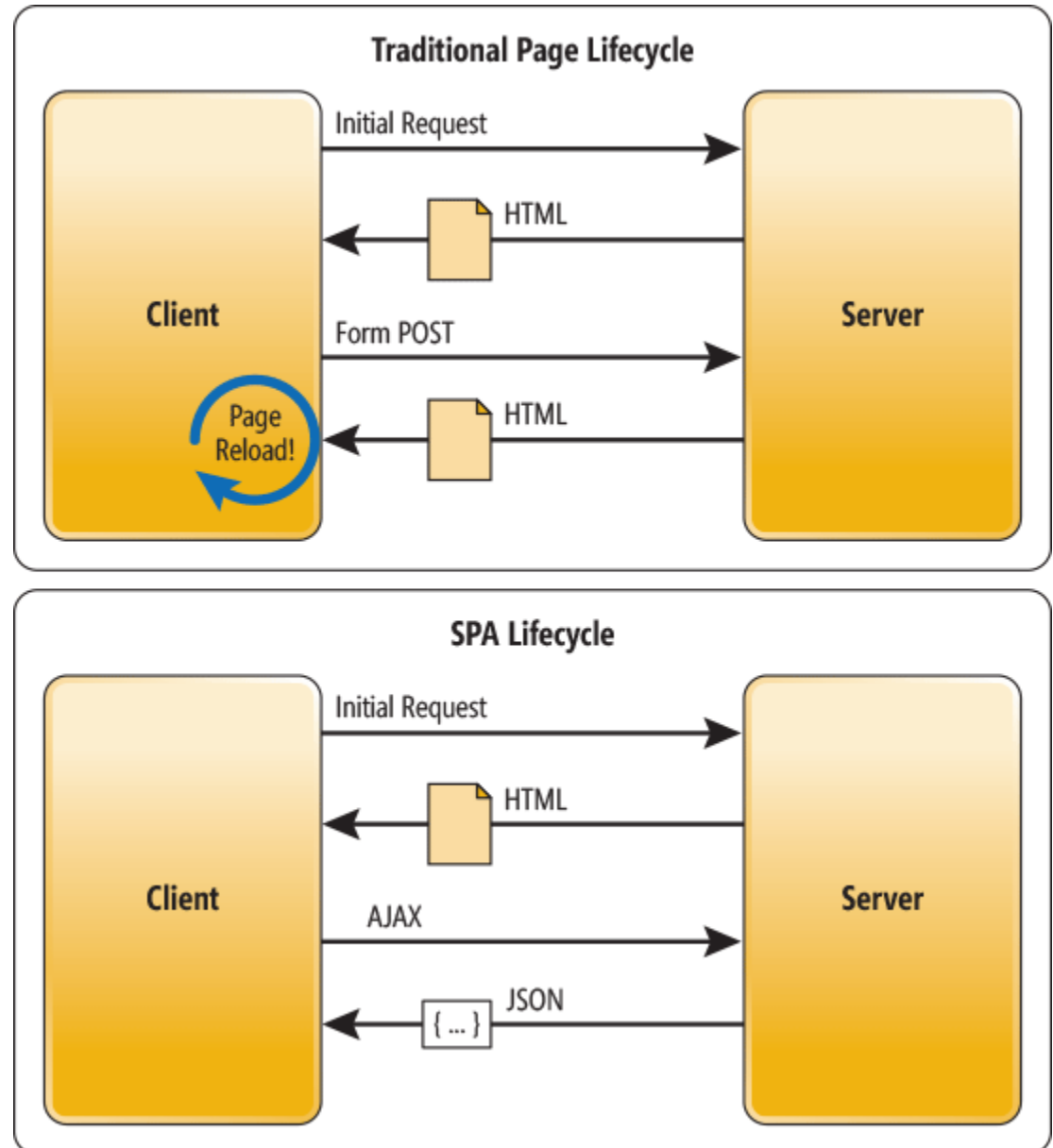
Três vídeos que merecem ser vistos

- [Higher-order functions](#)
- [Map](#)
- [Reduce basics](#)



Comunicação assíncrona com servidor

- Em aplicações SPA (Single Page Applications) é crucial saber como interagir com o servidor
- Esta interação é feita através de javascript
- O browser não atualiza a página automaticamente, tem que ser o código a fazê-lo (neste caso é função do React)
- A comunicação é assíncrona – o código não fica à espera depois de fazer os pedidos
- As duas formas principais de fazer esta comunicação são: API Fetch ou biblioteca Axios



Uma das formas é usar a API Fetch

- É a forma mais simples de interagir com API no servidor
- Não esquecer a “same origin policy” e/ou CORS: podem ocorrer problemas quando o URL da API não está no mesmo host do código que invoca a API
- Não esquecer também que as respostas ao pedido chegam de forma assíncrona – **todo o código tem que ser pensado de forma assíncrona**

```
fetch("API_URL")
  .then((resp) => resp.json())
  .then(function(data) {
    //processamento dos dados JSON
  })
  .catch(function(err) {
    //processamento do erro
  });
```

Outra forma é usar a biblioteca axios

- A utilização é semelhante
- Contudo tem algumas diferenças (ver próximo slide)
- Para usar a biblioteca axios é preciso instalá-la e depois importá-la no código

```
npm install axios
```

```
import axios from 'axios'

axios
  .get('API_URL')
  .then(response => {
    console.log("resposta:", response.data)
  })
  .catch(err => console.log("erro: ", err))
```

Diferenças entre API fetch e AXIOS

| Axios | Fetch |
|--|---|
| Axios has url in request object. | Fetch has no url in request object. |
| Axios is a stand-alone third party package that can be easily installed. | Fetch is built into most modern browsers; no installation is required as such. |
| Axios enjoys built-in XSRF protection. | Fetch does not. |
| Axios uses the data property. | Fetch uses the body property. |
| Axios' data contains the object . | Fetch's body has to be stringified . |
| Axios request is ok when status is 200 and statusText is ok property . | Fetch request is ok when response object contains the 'OK' . |
| Axios performs automatic transforms of JSON data . | Fetch is a two-step process when handling JSON data- first, to make the actual request; second, to call the <code>.json()</code> method on the response. |
| Axios allows cancelling request and request timeout . | Fetch does not. |
| Axios has the ability to intercept HTTP requests . | Fetch, by default, doesn't provide a way to intercept requests. |
| Axios has built-in support for download progress . | Fetch does not support upload progress. |
| Axios has wide browser support . | Fetch only supports Chrome 42+, Firefox 39+, Edge 14+, and Safari 10.1+ (This is known as Backward Compatibility). |

Instalação de servidor JSON para testes


Instalar json-server:

```
sudo npm install -g json-server
```

Criar uma pasta e um ficheiro json para testes

```
mkdir json-server  
cd json-server  
nano db.json
```

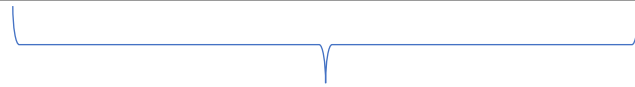
Os dados estão
no próximo slide




Correr o json server

```
npx json-server --host 192.168.1.97 --port 3001 --watch db.json
```

Endereço IP e port onde o servidor vai receber os pedidos. Atenção, não pode ser o mesmo port do node ou de outro servidor ativo



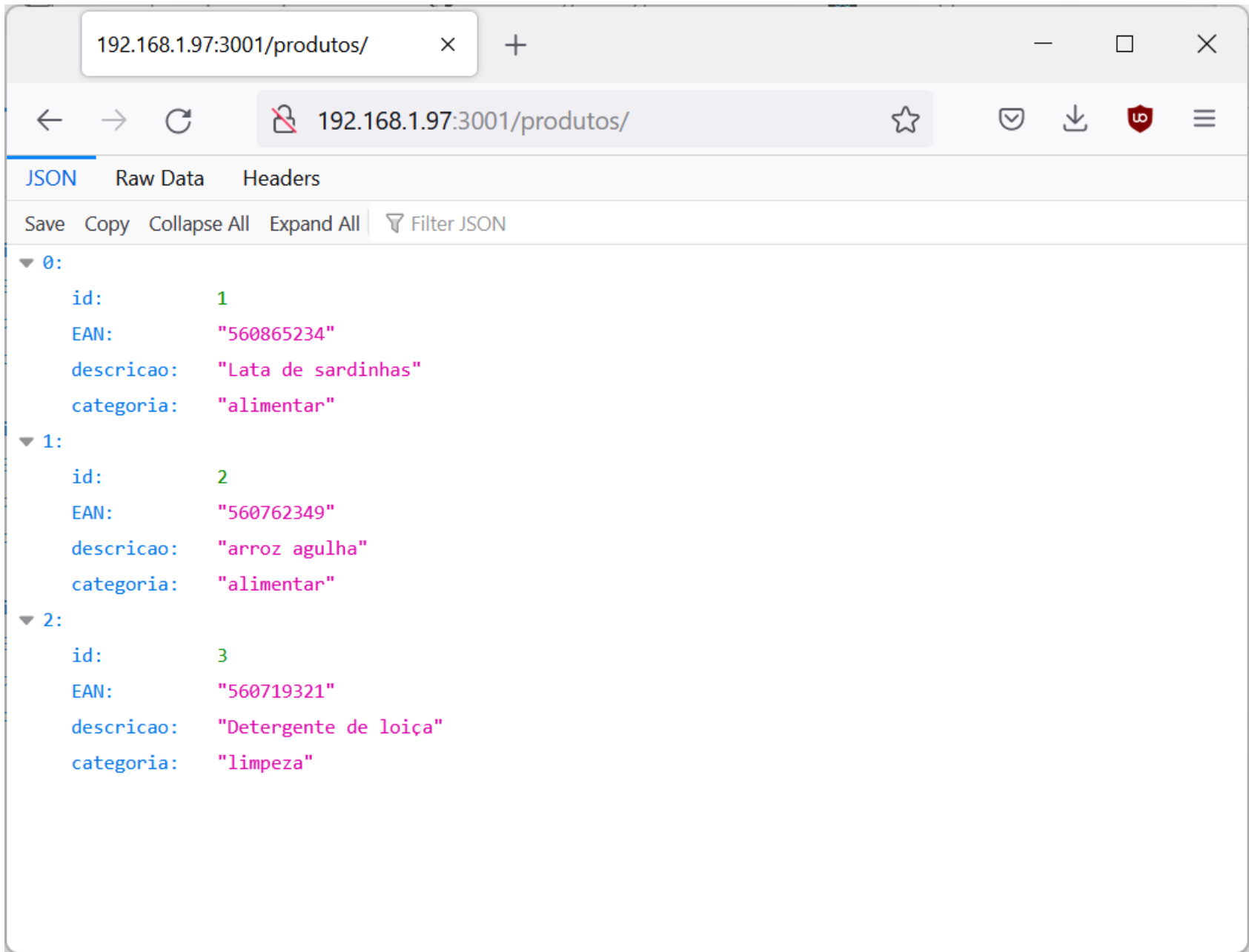
Ficheiro que vai ser servido



Ficheiro db.json com alguns dados

```
{
  "produtos": [
    {
      "id": 1,
      "EAN": "560865234",
      "descricao": "Lata de sardinhas",
      "categoria": "alimentar"
    },
    {
      "id": 2,
      "EAN": "560762349",
      "descricao": "arroz agulha",
      "categoria": "alimentar"
    },
    {
      "id": 3,
      "EAN": "560719321",
      "descricao": "Detergente de loiça",
      "categoria": "limpeza"
    }
  ]
}
```

Teste 1 de funcionamento do json-server



The screenshot shows a web browser window with the address bar displaying `192.168.1.97:3001/produtos/`. The browser's developer tools are open, showing the JSON response from the API. The response is a JSON array with 3 items, each containing the following fields:

- `id`: 1
- `EAN`: "560865234"
- `descricao`: "Lata de sardinhas"
- `categoria`: "alimentar"

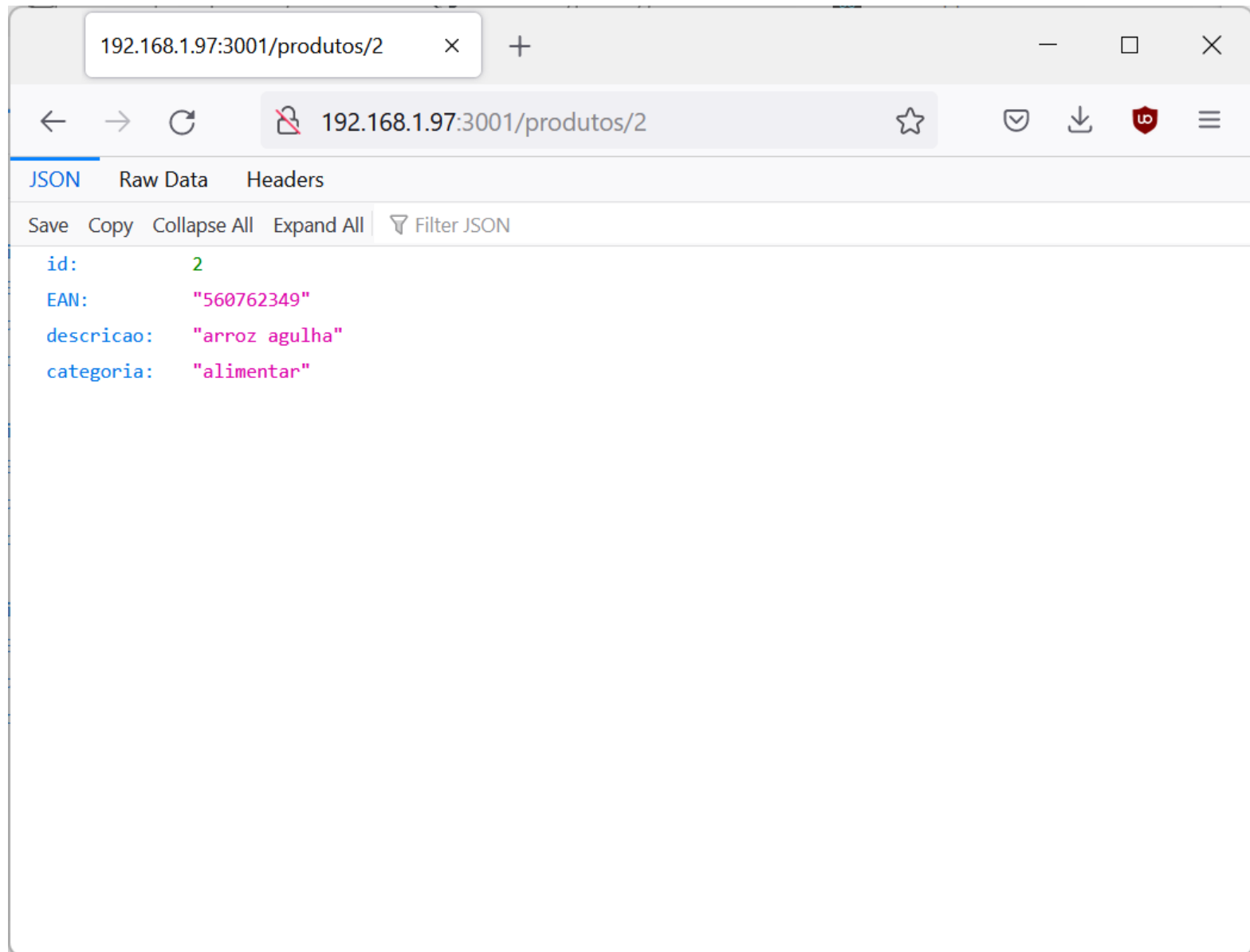
The second item in the array is:

- `id`: 2
- `EAN`: "560762349"
- `descricao`: "arroz agulha"
- `categoria`: "alimentar"

The third item in the array is:

- `id`: 3
- `EAN`: "560719321"
- `descricao`: "Detergente de loiça"
- `categoria`: "limpeza"

Teste 2 de funcionamento do json-server

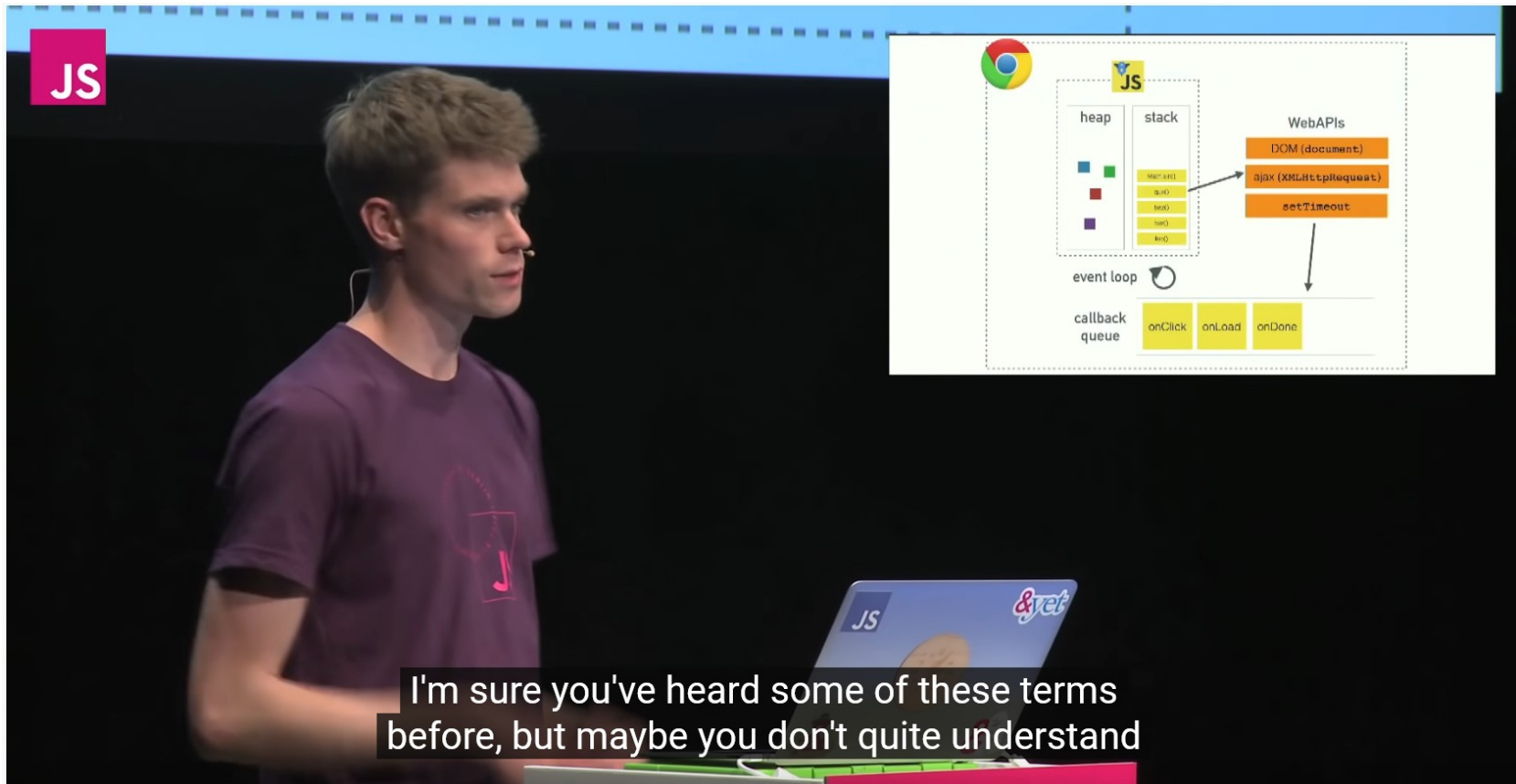


Exercício

Crie uma página React que aceda à API acabada de criar e mostre os dados recebidos **na consola**

Use a biblioteca axios

Vídeo que explica o event loop do JS



The video frame shows a presenter on the left and a diagram of the JavaScript event loop on the right. The presenter is a young man with short brown hair, wearing a purple t-shirt with a pink logo. He is standing next to a laptop with a blue lid featuring the 'JS' logo and the '&yet' logo. The diagram on the right is titled 'JS' and illustrates the event loop mechanism. It shows a 'heap' with four colored squares (blue, green, red, purple) and a 'stack' with four yellow boxes labeled 'setTimeout', 'setTimeout', 'setTimeout', and 'setTimeout'. An arrow points from the top 'setTimeout' box in the stack to a 'WebAPIs' section. The 'WebAPIs' section contains three orange boxes: 'DOM (document)', 'ajax (XMLHttpRequest)', and 'setTimeout'. An arrow points from the 'setTimeout' box in 'WebAPIs' to a 'callback queue' section. The 'callback queue' section contains three yellow boxes: 'onClick', 'onLoad', and 'onDone'. An 'event loop' icon (a circular arrow) is positioned between the 'stack' and the 'callback queue'.

JS

heap

stack

WebAPIs

DOM (document)

ajax (XMLHttpRequest)

setTimeout

event loop

callback queue

onClick

onLoad

onDone

JS

&yet

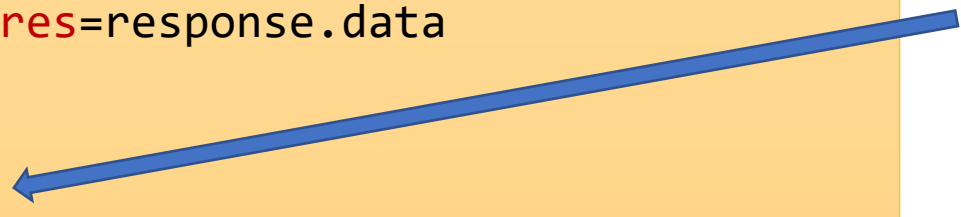
I'm sure you've heard some of these terms before, but maybe you don't quite understand

Usando o fetch ou axios num componente

- Como a resposta aos pedidos não é síncrona, quando o componente retorna um valor, provavelmente a resposta ainda não chegou
- Isto representa um problema caso se pretenda que um componente lide com pedidos e respostas de um servidor
- A seguinte estratégia **não funciona**

```
const App = () => {  
  
  axios  
    .get('http://192.168.1.97:3001/produtos/')  
    .then(response => {  
      const res=response.data  
    })  
  
  return (  
    <div>  
      {res}  
    </div>  
  );  
}
```

Este código vai ser interpretado antes de chegar a resposta



Os Effect-hooks

- Um componente React usa **props** e/ou o **estado** para calcular a saída, o valor de retorno JSX
- Se o componente também faz operações que não são usadas diretamente no valor de saída, estas operações são chamadas **side-effects**
- Os **side-effects** não devem ser processados ao mesmo tempo que o cálculo da saída, mas apenas depois de o DOM estar atualizado e renderizado

```
function Greet({ name }) {  
  const message = `Hello, ${name}!`;  
  
  // Errado!  
  document.title = `Greetings to ${name}`; // Side-effect!  
  
  return <div>{message}</div>;  
}
```

- Os **side-effects** são ideais para manipular diretamente o DOM, manipular pedidos AJAX e usar funções de temporização como o `setTimeout()`
- Artigo para aprofundar este tema: <https://blog.logrocket.com/guide-to-react-useeffect-hook/>

Os Effect-hooks

```
import { useEffect } from 'react';

function Greet({ name }) {
  const message = `Hello, ${name}!`;

  useEffect(() => {
    // Assim está ok
    document.title = `Greetings to ${name}`; // Side-effect!
  }, [name]);

  return <div>{message}</div>;
}
```

- O hook `useEffect` aceita dois argumentos:
 - O **primeiro** é a função que contém a lógica do side-effect. Este código só será interpretado **depois das alterações** no DOM (render).
 - O **segundo argumento**, opcional, é um array de dependências. A função do side-effect só será interpretada se as dependências foram alteradas desde o último render. Se este argumento for um array vazio `[]` a função só é interpretada uma vez.

Exercício

Crie uma página React que aceda à API acabada de criar e mostre a lista de produtos da categoria “alimentar”

Use a biblioteca axios

Tente escrever código o mais modular e limpo possível

Dicas: use hooks state e effect ; use o método filter do array para filtrar os elementos JSON ; use o método map do array para criar o JSX de saída

Outro exercício

Crie uma página React que permita acrescentar produtos à lista, usando a API anterior

Use a biblioteca axios

Tente escrever código o mais modular e limpo possível

Dicas: agora vai usar o método post da biblioteca axios. Não precisa de enviar o id, o JSON server acrescenta-o automaticamente

Correr um projeto em produção

- O projeto em modo de desenvolvimento tem muitas funcionalidades de debugging que tornam o código mais lento
- Em produção, deve ser usada a versão de produção, que é otimizada para desempenho
- Isso é feito desta forma:

```
npm run build  
  
sudo npm install -g serve  
  
serve -s build
```

- Ou em alternativa, copiar o conteúdo da pasta build para a raiz de um virtual host apache ou nginx, ou para um bucket S3 da Cloud AWS
- **De facto, do lado do servidor, os conteúdos React são estáticos, todo o processamento é feito do lado do cliente !!!**

E finalmente...

