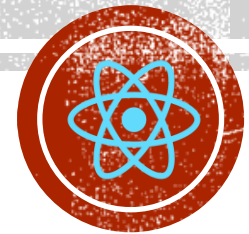


SINGLE PAGE APPLICATIONS COM REACT-DOM



Douglas Nassif Roma Junior

 /douglasjunior

 /in/douglasjunior

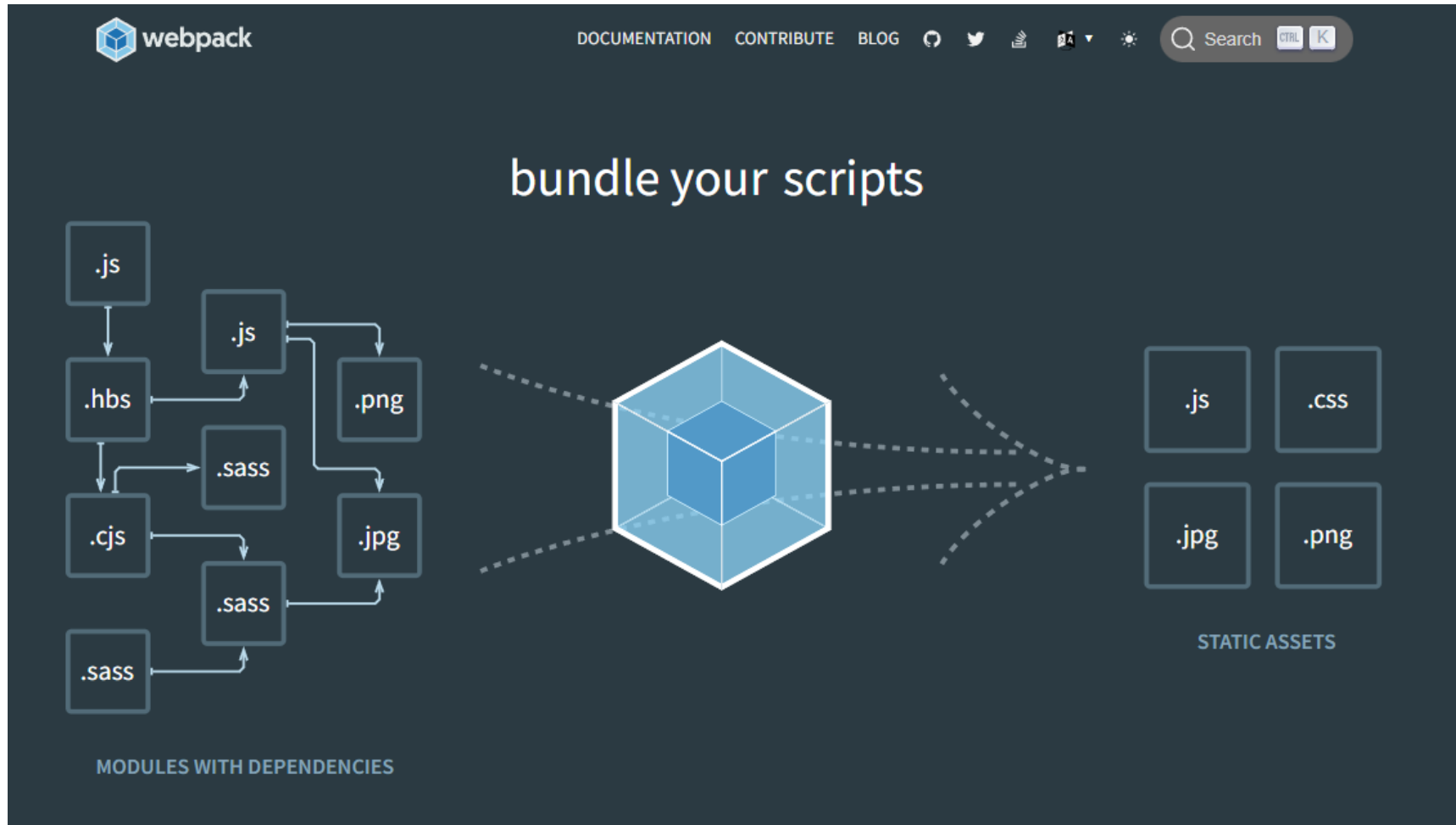
 douglas@smarppy.com

AGENDA

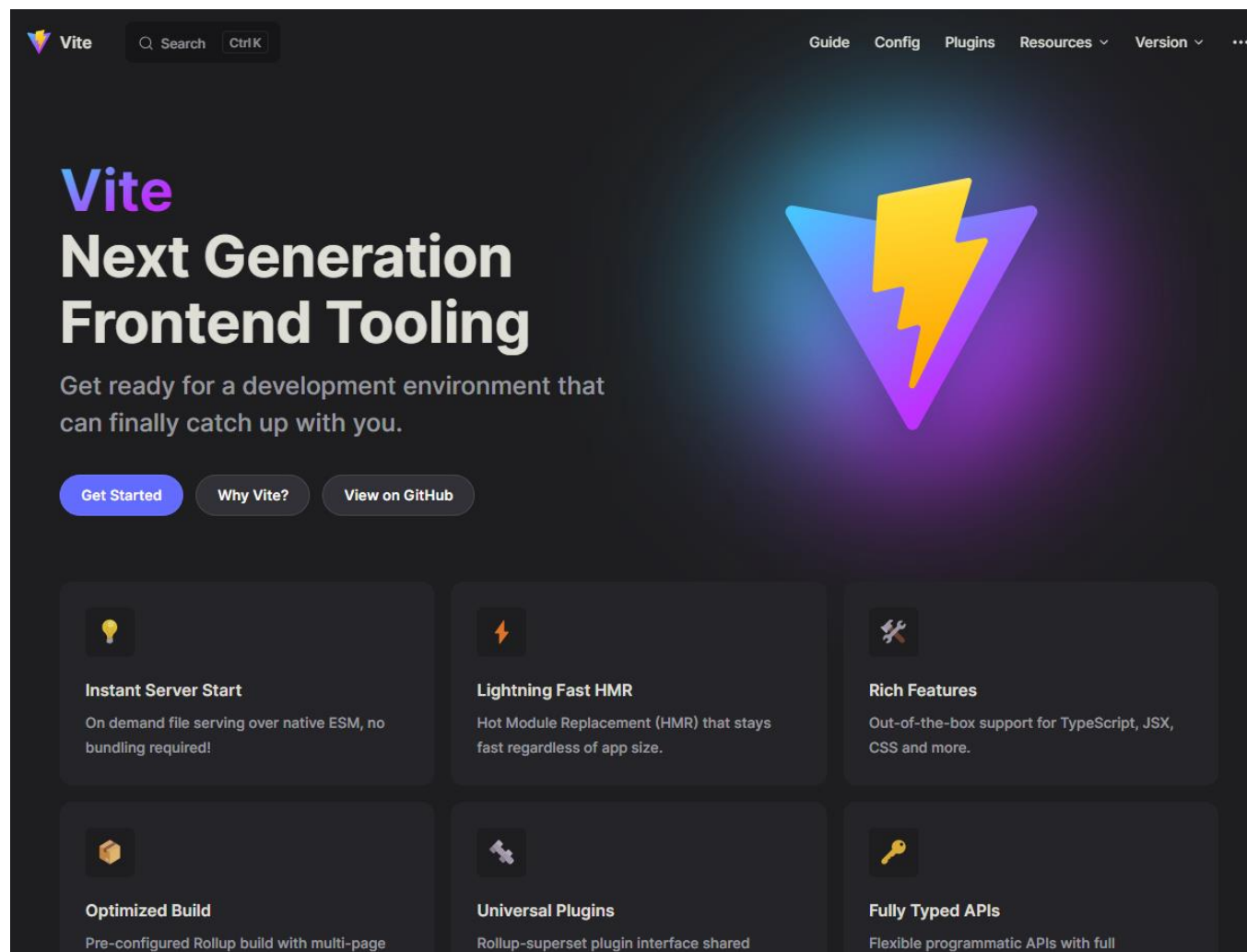
Tópico	Conteúdo
Bundlers e Frameworks	<ul style="list-style-type: none">- Introdução ao ambiente (webpack, Vite, Next, Gatsby, Remix)
React JS	<ul style="list-style-type: none">- Introdução ao ReactJS<ul style="list-style-type: none">- Componentes- Componente funcional- Componente de classe- Hooks- ECMAScript 2015 e JSX- Criando projetos com bundler- Estado, propriedades e ciclo de vida- Coleções de Componentes

BUNDLERS E FRAMEWORKS

WEBPACK



VITE



The screenshot shows the Vite website homepage with a dark theme. At the top, there's a navigation bar with the Vite logo, a search bar, and links for Guide, Config, Plugins, Resources, Version, and a menu icon. The main heading reads 'Vite Next Generation Frontend Tooling'. Below this, a subheading says 'Get ready for a development environment that can finally catch up with you.' There are three buttons: 'Get Started', 'Why Vite?', and 'View on GitHub'. The page features six feature cards arranged in a 2x3 grid. Each card has an icon, a title, and a description. The features are: Instant Server Start (lightbulb icon), Lightning Fast HMR (lightning bolt icon), Rich Features (wrench icon), Optimized Build (cube icon), Universal Plugins (gears icon), and Fully Typed APIs (key icon).

Vite
Next Generation
Frontend Tooling

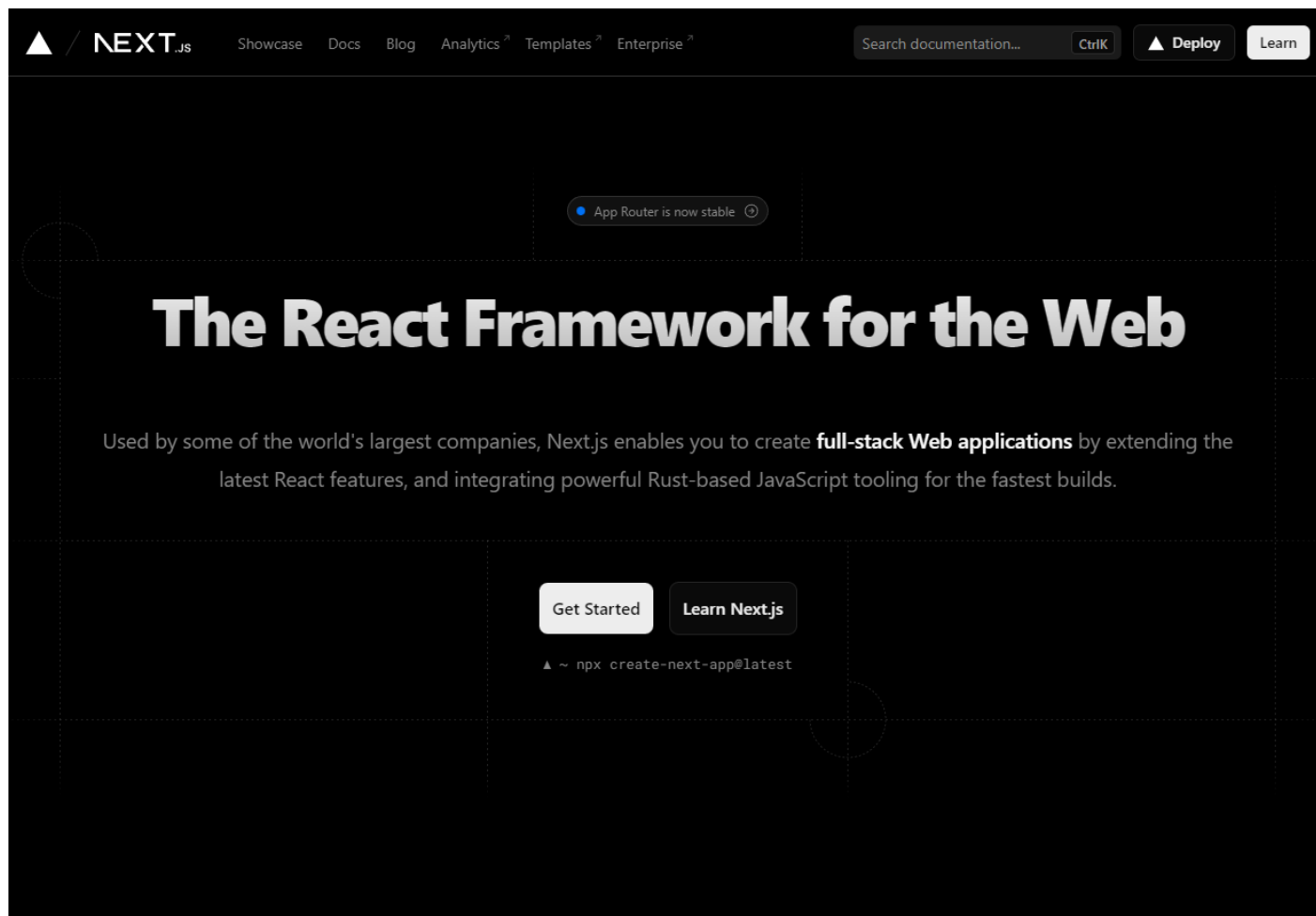
Get ready for a development environment that
can finally catch up with you.

[Get Started](#) [Why Vite?](#) [View on GitHub](#)

- Instant Server Start**
On demand file serving over native ESM, no bundling required!
- Lightning Fast HMR**
Hot Module Replacement (HMR) that stays fast regardless of app size.
- Rich Features**
Out-of-the-box support for TypeScript, JSX, CSS and more.
- Optimized Build**
Pre-configured Rollup build with multi-page
- Universal Plugins**
Rollup-superset plugin interface shared
- Fully Typed APIs**
Flexible programmatic APIs with full

<https://vitejs.dev/>

NEXT



GASTBY

SHIP IT FASTER

Introducing Netlify Connect: Connect Everything. Build Anything. [Learn more here](#)

[Support](#) [Log In](#)



Gatsby

[Why Gatsby](#) ▾

[Products](#) ▾

[Open Source](#) ▾

[Learn](#) ▾

[Company](#) ▾

[Pricing](#)



[Contact](#)

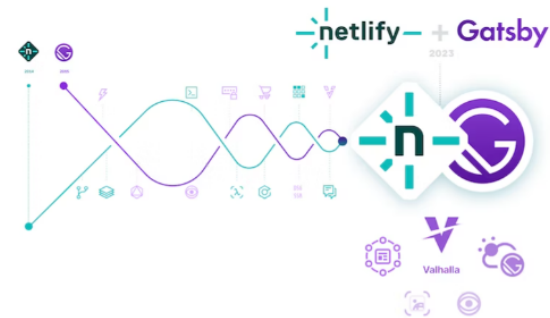
[Sign Up](#)

Gatsby has joined Netlify!

Together Gatsby and Netlify are accelerating growth and bringing composable architectures to the modern web.

[Watch the webinar](#)

[Request a demo](#)



It's in our DNA to be fast.

50%

Faster Page Load

20×

Faster Build Times

2×

SEO Boost



REMIX

Remix

[Blog](#) [Docs](#) [GitHub](#) [Remix Conf](#)

Focused on web standards and modern web app UX, you're simply going to build better websites

Remix is a full stack web framework that lets you focus on the user interface and work back through web standards to deliver a fast, slick, and resilient user experience. People are gonna love using your stuff.

[Get Started](#)

[Read the Docs](#)

```
export async function loader({ request }) {
  return getProjects();
}

export async function action({ request }) {
  const form = await request.formData();
  return createProject({ title: form.get("title") });
}

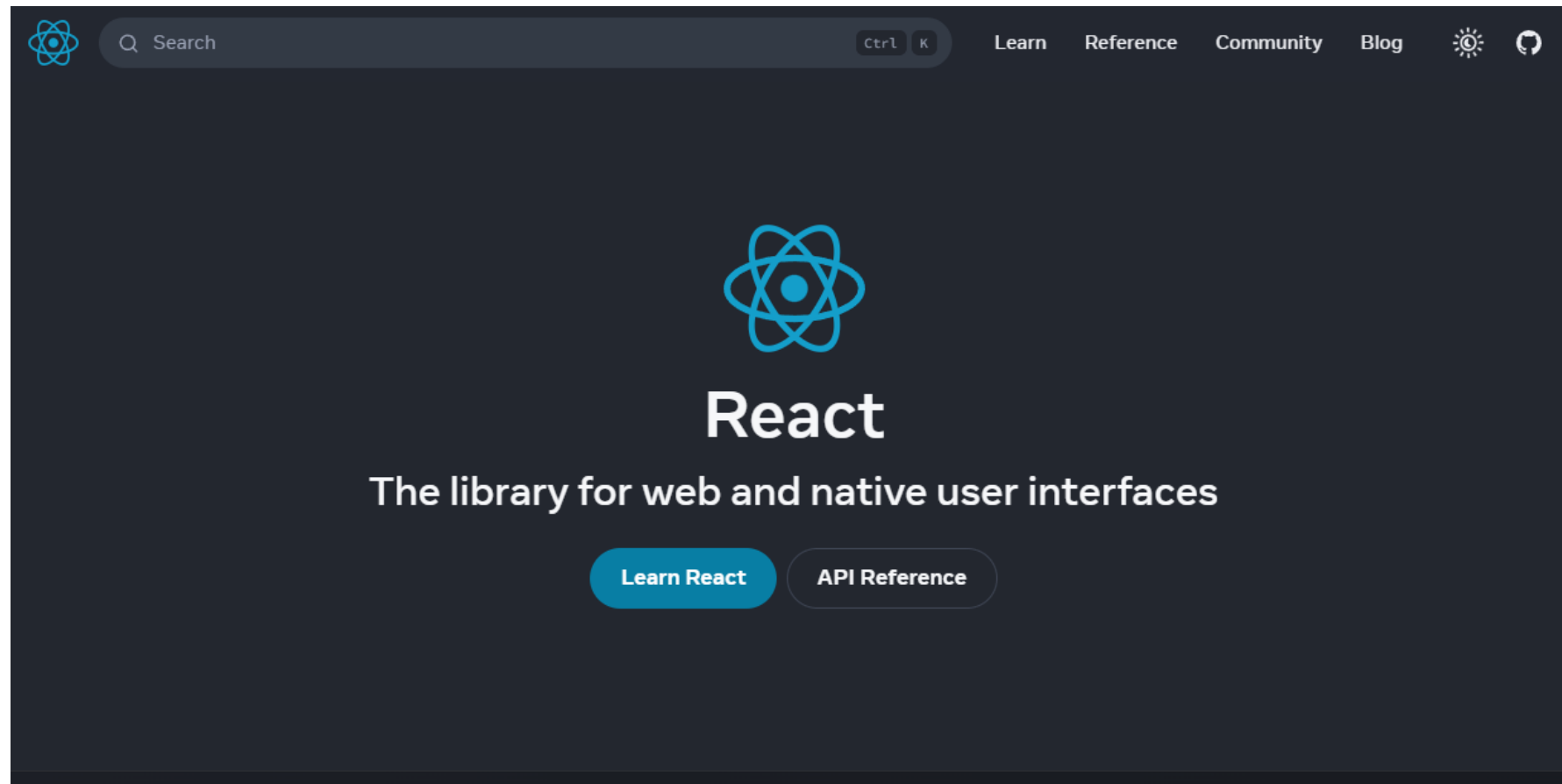
export default function Projects() {
  const projects = useLoaderData();
  const { state } = useNavigation();
  const busy = state === "submitting";

  return (
    <div>
      {projects.map((project) => (
        <Link to={project.slug}>{project.title}</Link>
      ))}

      <Form method="post">
        <input name="title" />
        <button type="submit" disabled={busy}>
          {busy ? "Creating..." : "Create New Project"}
        </button>
      </Form>
    </div>
  );
}
```


REACT-DOM

INTRODUÇÃO AO REACT-DOM

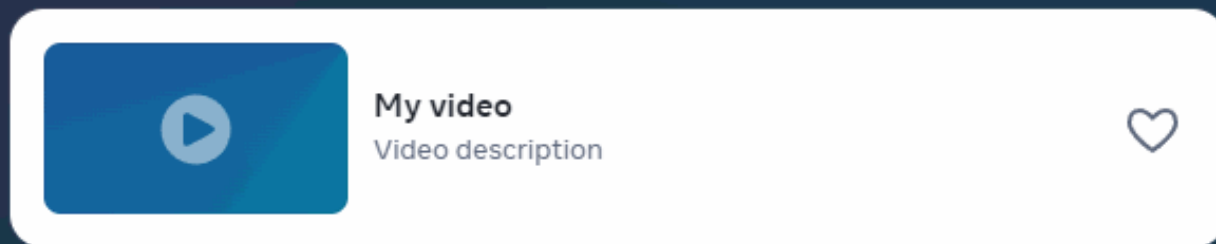


Create user interfaces from components

React lets you build user interfaces out of individual pieces called components. Create your own React components like `Thumbnail`, `LikeButton`, and `Video`. Then combine them into entire screens, pages, and apps.

Video.js

```
function Video({ video }) {  
  return (  
    <div>  
      <Thumbnail video={video} />  
      <a href={video.url}>  
        <h3>{video.title}</h3>  
        <p>{video.description}</p>  
      </a>  
      <LikeButton video={video} />  
    </div>  
  );  
}
```



INTRODUÇÃO AO REACT-DOM

■ Declarativo

- React facilita a criação de UIs interativas. Crie *views* simples para cada estado em seu aplicativo e o React irá atualizar e renderizar eficientemente apenas os componentes certos quando seus dados forem alterados.
- Views declarativas tornam seu código mais previsível e mais fácil de depurar.

```
class HelloMessage extends React.Component {  
  render() {  
    return (  
      <div>  
        Hello {this.props.name}  
      </div>  
    );  
  }  
}
```

INTRODUÇÃO AO REACT-DOM

- **Baseado em Componentes**
- Crie componentes encapsulados que gerenciem seu próprio estado, e então, use-os para compor UIs complexas.
- Uma vez que a lógica dos componentes está escrita em JavaScript, você pode facilmente passar dados através do seu aplicativo e manter o estado fora do DOM.

```
class Timer extends React.Component {  
  tick() {  
    this.setState((prevState) => ({  
      seconds: prevState.seconds + 1  
    }));  
  }  
  componentDidMount() {  
    setInterval(() => this.tick(), 1000);  
  }  
  render() {  
    return (  
      <div>  
        Seconds: {this.state.seconds}  
      </div>  
    );  
  }  
}
```

INTRODUÇÃO AO REACT-DOM

- **Aprenda uma vez, use em qualquer lugar**
- Não exige que você altere o conjunto de tecnologias utilizados em sua stack, evitando reescrever o código de sua aplicação.
- React também pode renderizar no servidor usando **Node** ou alimentar aplicativos móveis usando o **React Native**.
- React-DOM (Web), React-Native (Android, iOS, Windows), React XP (Mobile, Desktop, Web), React-Ink (Terminal)

```
import React, { Component } from 'react';
import { Text, View } from 'react-native';

class WhyReactNativeIsSoGreat extends Component {
  render() {
    return (
      <View>
        <Text>
          Se você gosta do React na web, você
          vai gostar do React Native.
        </Text>
        <Text>
          Você apenas usa componentes nativos
          como 'View' e 'Text', em vez de um
          componentes web como 'div' e 'span'.
        </Text>
      </View>
    );
  }
}
```

CRIANDO UM PROJETO NOVO

- Crie um diretório chamado `primeiro-projeto-web`
- Dentro do diretório crie um arquivo chamado `index.html`
- Neste arquivo, crie um *template* padrão para uma página HTML5, com `head` e `body`.

UTILIZANDO REACT SEM BUNDLER

- Para utilizar o ReactJS na web, você precisa apenas importar as bibliotecas React e React-DOM no `<head>` de sua página.

```
<script crossorigin src="https://unpkg.com/react@17.0.2/umd/react.development.js"></script>  
<script crossorigin src="https://unpkg.com/react-dom@17.0.2/umd/react-dom.development.js"></script>
```

- E então você já pode renderizar seu primeiro componente.

```
<body>  
  <div id="root"></div>  
  
  <script>  
    const myDiv = React.createElement('div', null, 'Olá React!');  
    ReactDOM.render(myDiv, document.getElementById('root'));  
  </script>  
</body>
```


INTRODUÇÃO AO REACT-DOM

- Código completo:

```
<!DOCTYPE html>

<head>
  <meta charset="UTF-8">
  <title>My React Website</title>

  <script crossorigin src="https://unpkg.com/react@17.0.2/umd/react.development.js"></script>
  <script crossorigin src="https://unpkg.com/react-dom@17.0.2/umd/react-dom.development.js"></script>
</head>

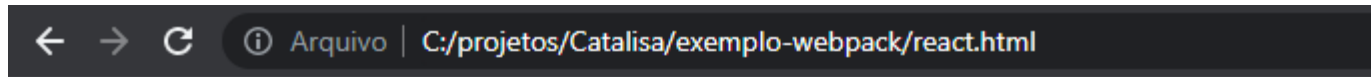
<body>
  <div id="root"></div>

  <script>
    const myDiv = React.createElement('div', null, 'Olá React!');
    ReactDOM.render(myDiv, document.getElementById('root'));
  </script>
</body>

</html>
```

INTRODUÇÃO AO REACT-DOM

- Resultado:



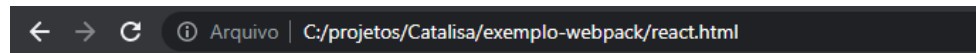
Olá React!

INTRODUÇÃO AO REACT-DOM

- Para visualizar a página, basta abrir o arquivo `index.html` no navegador, ou utilizar a ferramenta `serve` com o comando:

```
$ npx serve .
```

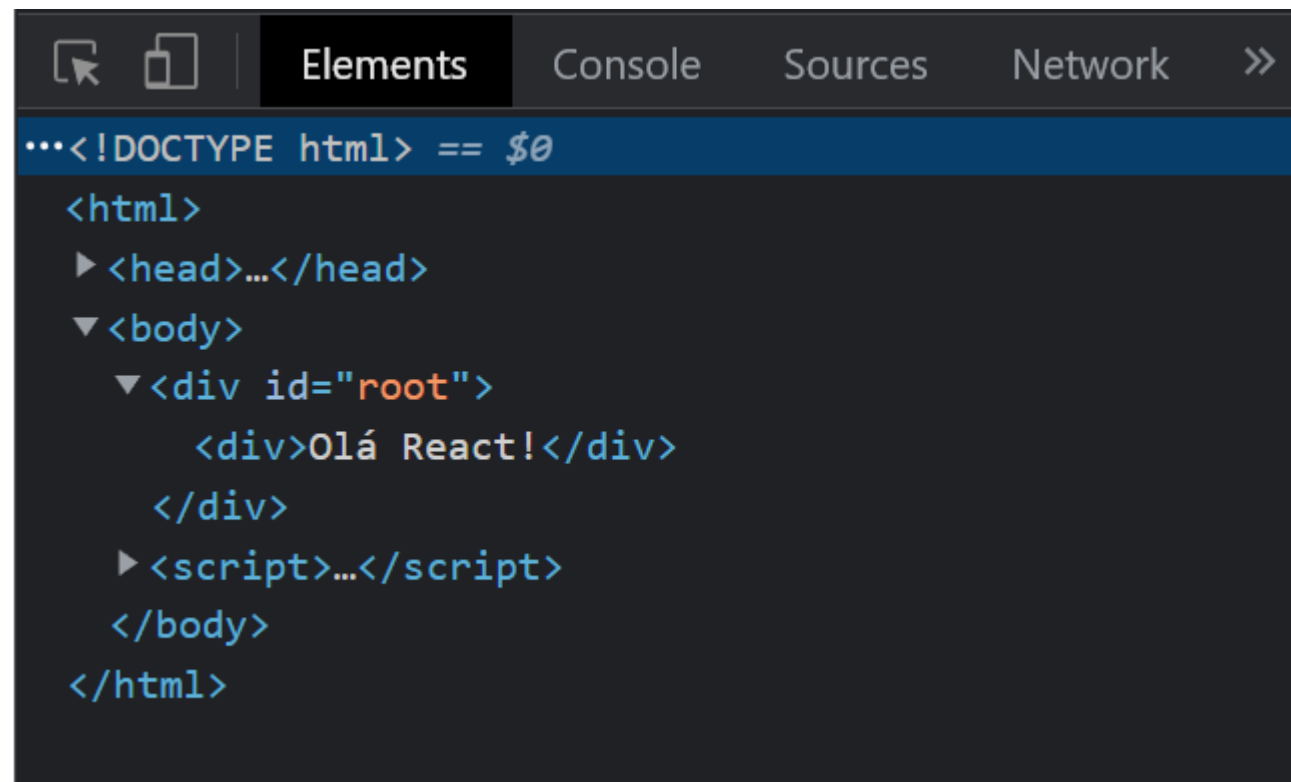
- Resultado:



Olá React!

INTRODUÇÃO AO REACT-DOM

- Código renderizado:

A screenshot of a web browser's developer tools, specifically the 'Elements' panel. The panel shows a tree view of the DOM. The root element is an HTML document, indicated by the '...' icon and the text '<!DOCTYPE html> == \$0'. Below this is the '<html>' element, which contains a '<head>...' element and a '<body>' element. The '<body>' element contains a '<div id="root">' element. This 'root' div contains a '<div>Olá React!</div>' element and a '<script>...</script>' element. The tree view is collapsed for the '<body>' and '<div id="root">' elements, with expand/collapse icons (triangles) next to them. The text is color-coded: HTML tags are in blue, attribute values are in orange, and text content is in white. The background of the developer tools is dark grey.

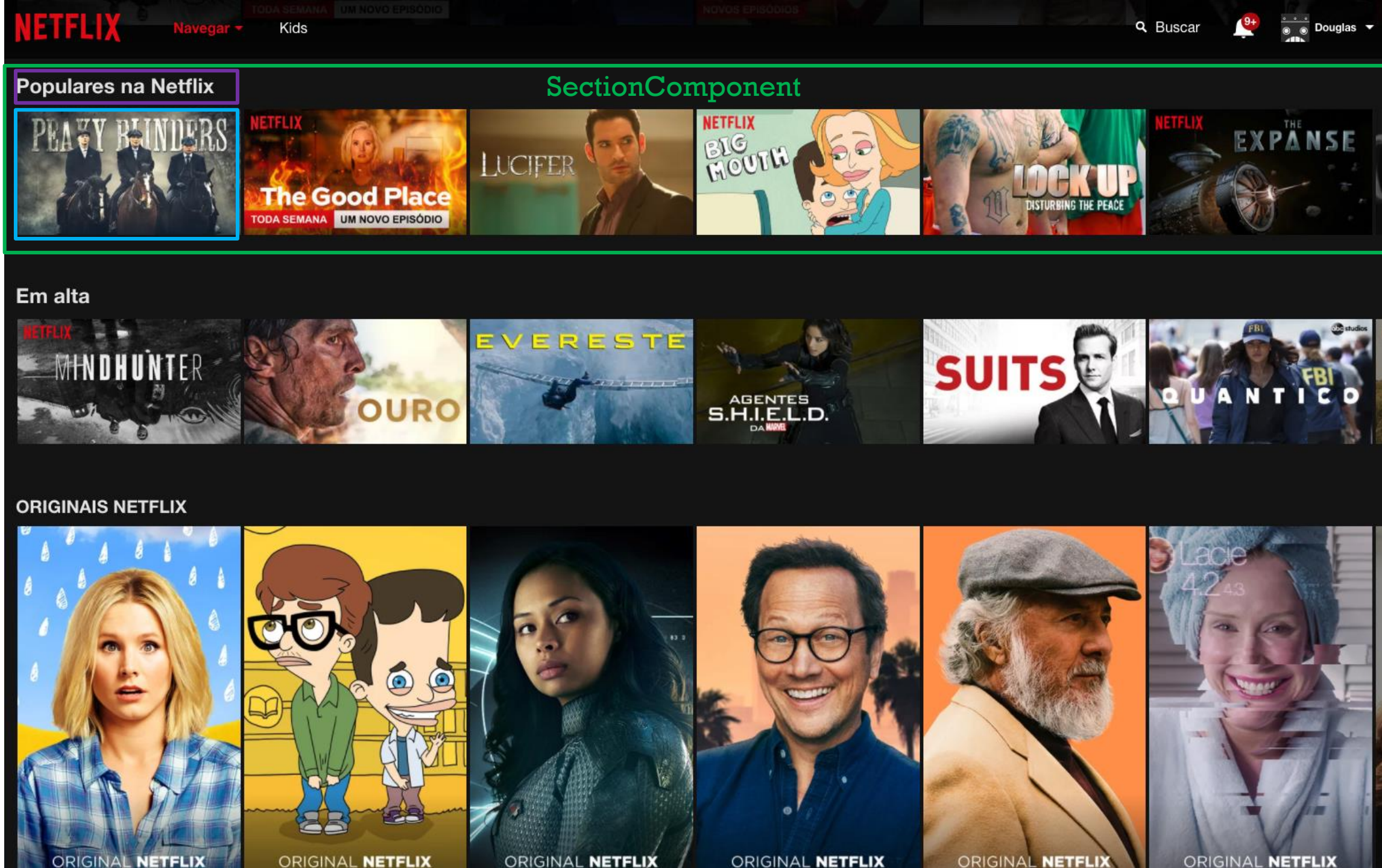
```
...<!DOCTYPE html> == $0
<html>
  ▶ <head>...</head>
  ▼ <body>
    ▼ <div id="root">
      <div>Olá React!</div>
    </div>
    ▶ <script>...</script>
  </body>
</html>
```

COMPONENTES

TUDO SÃO COMPONENTES

- Os componentes permitem que você divida a interface (UI) em partes independentes, reutilizáveis e pensar em cada uma isoladamente.
- Conceitualmente, os componentes são como funções JavaScript. Eles aceitam entradas arbitrárias (chamados de “props ou propriedades”) e retornam elementos descrevendo o que deve aparecer na tela.

LabelComponent
ItemComponent



COMPONENTES FUNCIONAIS

COMPONENTES FUNCIONAIS

- A maneira mais simples de se criar um Componente em ReactJS, é declarando uma função JavaScript.

```
function WelcomeComponent(props) {  
  return React.createElement('h1', null, 'Hello, ' + props.name);  
}
```

- Esta função é um componente válido pois ela recebe um parâmetro único chamado “props” e retorna um elemento React.
- É chamado de componente “funcional” pois, literalmente, é uma função JavaScript.

COMPONENTES FUNCIONAIS

- Código completo:

```
<script>
  function WelcomeComponent(props) {
    return React.createElement('h1', null, 'Hello, ' + props.name);
  }

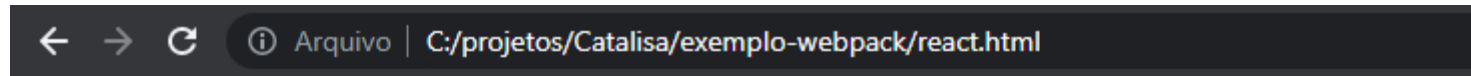
  const myComponent = React.createElement(
    WelcomeComponent,
    {
      name: 'Douglas'
    },
    null
  );

  const myDiv = React.createElement('div', null, 'Olá React!', myComponent);

  ReactDOM.render(myDiv, document.getElementById('root'));
</script>
```

COMPONENTES FUNCIONAIS

- Resultado:

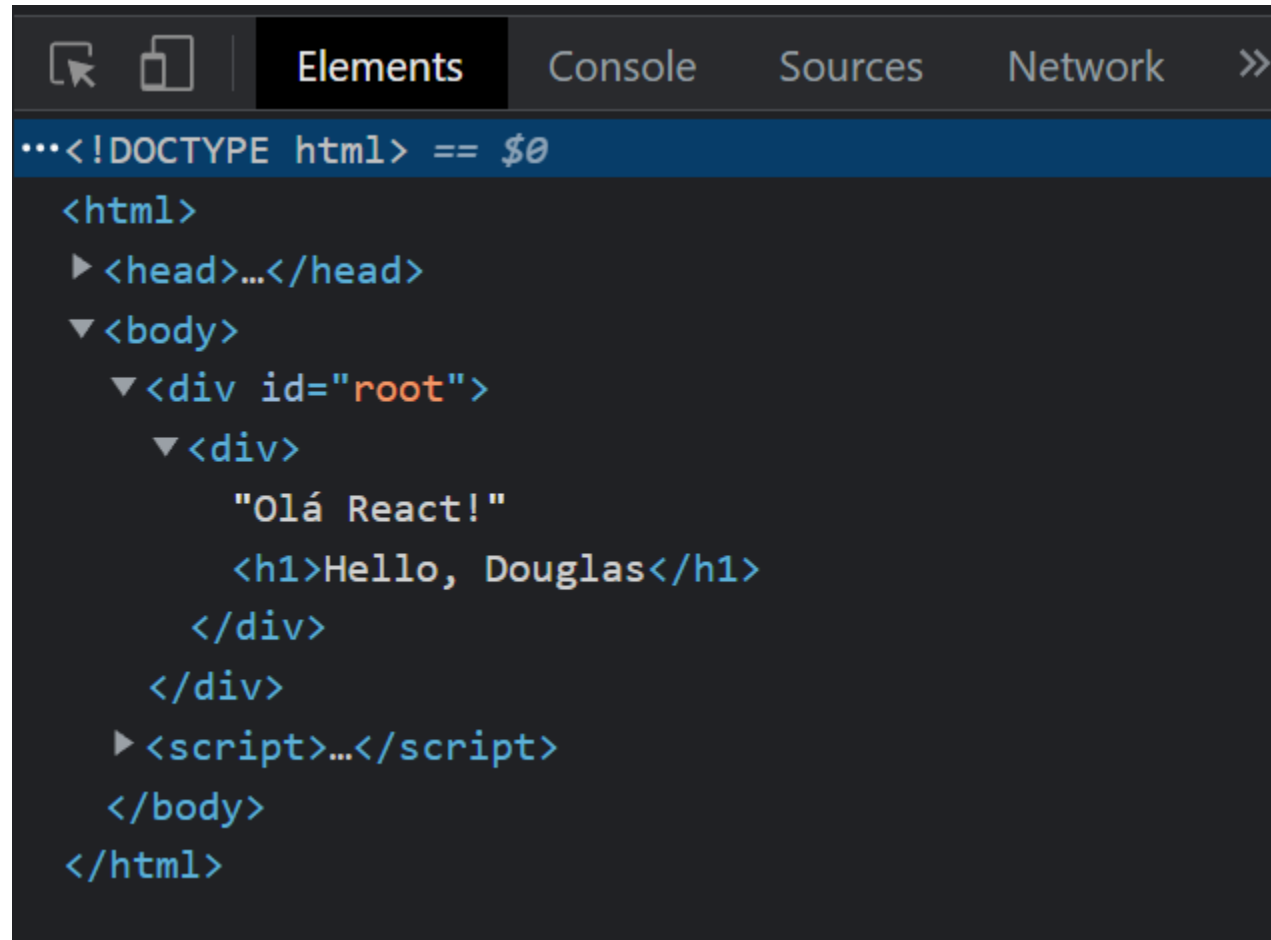


Olá React!

Hello, Douglas

COMPONENTES FUNCIONAIS

- Código renderizado:



The screenshot shows the 'Elements' panel of a web browser's developer tools. The DOM tree is expanded to show the following structure:

```
...<!DOCTYPE html> == $0
<html>
  <head>...</head>
  <body>
    <div id="root">
      <div>
        "Olá React!"
        <h1>Hello, Douglas</h1>
      </div>
    </div>
    <script>...</script>
  </body>
</html>
```

COMPONENTES DE CLASSES

COMPONENTES DE CLASSES

- Os componentes de classe também recebem valores através de “`props`” e podem renderizar um ou mais elementos React.
- Adicionalmente, **os componentes de classe são capazes de gerenciar seu próprio estado.**
- Se usado **sem bundler**:
 - À partir da versão 16, função de criação de classes foi movida para um pacote separado, então precisamos importar:

```
<script crossorigin src="https://unpkg.com/create-react-class@15.7.0/create-react-class.js"></script>
```

COMPONENTES DE CLASSES

- Os componentes de classe podem ser declarados assim:

```
const CounterComponent = createReactClass({
  getInitialState: function () {
    return {
      count: 0,
    };
  },
  componentDidMount: function () {
    const self = this;
    setInterval(function () {
      self.setState({ count: self.state.count + 1 })
    }, 1000);
  },
  render: function () {
    return React.createElement('p', null, this.state.count);
  }
});
```

COMPONENTES DE CLASSES

■ Código completo:

```
<script>
  function WelcomeComponent(props) {
    return React.createElement('h1', null, 'Hello, ' + props.name);
  }

  const myComponent = React.createElement(
    WelcomeComponent,
    {
      name: 'Douglas'
    },
    null
  );

  const CounterComponent = createReactClass({
    getInitialState: function () {
      return {
        count: 0,
      };
    },
    componentDidMount: function () {
      const self = this;
      setInterval(function () {
        self.setState({ count: self.state.count + 1 })
      }, 1000);
    },
    render: function () {
      return React.createElement('p', null, this.state.count);
    }
  });

  const myCounter = React.createElement(
    CounterComponent,
    null,
    null
  );

  const myDiv = React.createElement('div', null, 'Olá React!', myComponent, myCounter);

  ReactDOM.render(myDiv, document.getElementById('root'));
</script>
```

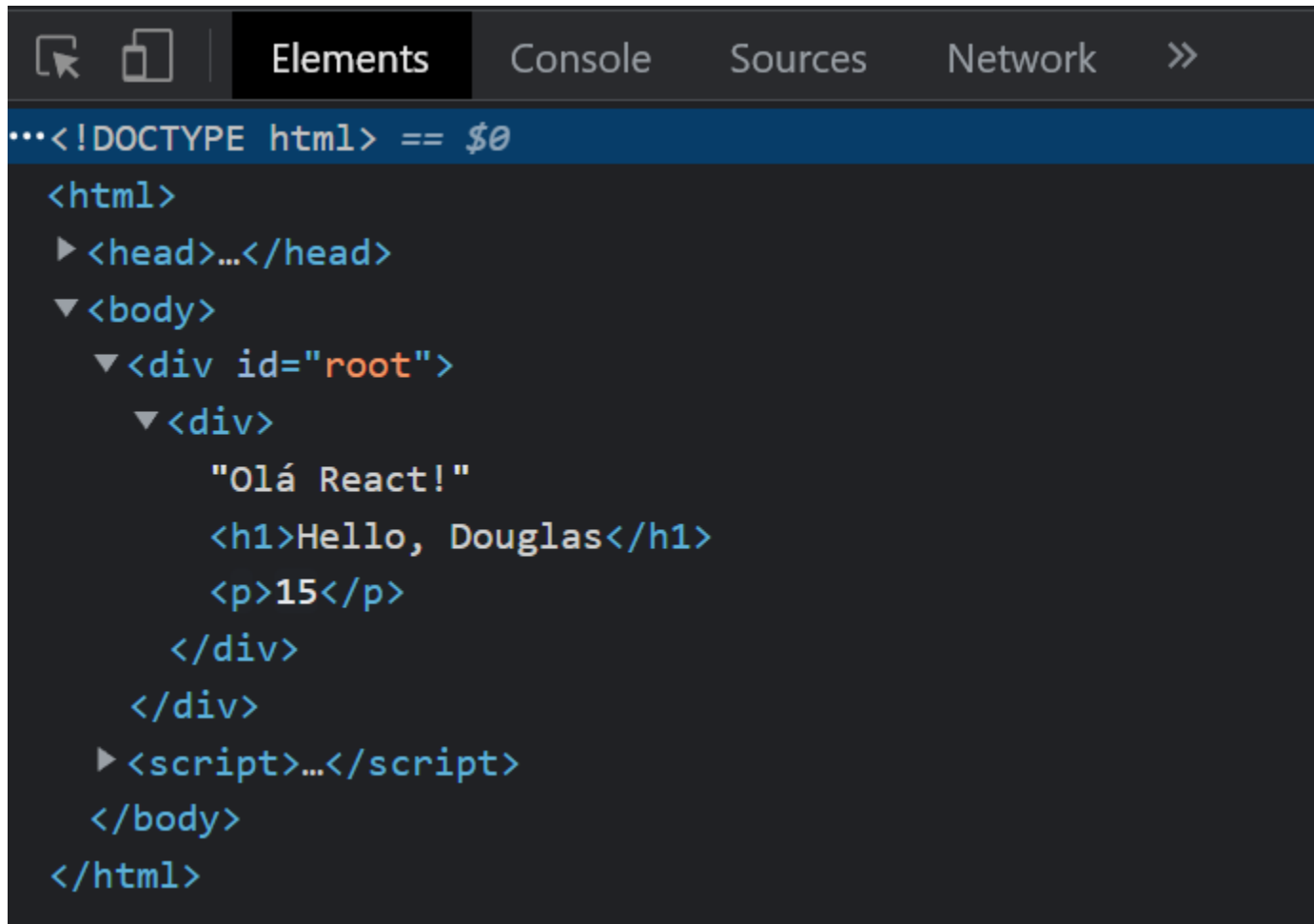

COMPONENTES DE CLASSES

- Resultado:



COMPONENTES DE CLASSES

- Código renderizado:



The screenshot shows the 'Elements' panel of a web browser's developer tools. The DOM tree is expanded to show the following structure:

```
...<!DOCTYPE html> == $0
<html>
  <head>...</head>
  <body>
    <div id="root">
      <div>
        "Olá React!"
        <h1>Hello, Douglas</h1>
        <p>15</p>
      </div>
    </div>
    <script>...</script>
  </body>
</html>
```

REACT HOOKS

HOOKS

- A partir do **React 16.8**, os **componentes funcionais** podem trabalhar com `hooks` para gerenciar estado, criar referências, receber eventos, dentre outras coisas.
- Até o **React 16.7**, esses recursos existiam apenas nos componentes de classe.
- **OPINIÃO:** *Ainda não existe nada oficial, mas parece que a tendência é que os componentes de classe poderão deixar de existir no futuro.*

HOOKS

- Exemplo de uso do useState.

```
// ...  
  
const StepComponent = () => {  
  const [step, setStep] = React.useState(0);  
  const incrementStep = () => {  
    setStep(s => s += 1);  
  };  
  const decrementStep = () => {  
    setStep(s => s -= 1);  
  };  
  return React.createElement('div', null,  
    React.createElement('button', { onClick: incrementStep }, '+'),  
    React.createElement('span', null, step),  
    React.createElement('button', { onClick: decrementStep }, '-'),  
  );  
};  
  
const myStep = React.createElement(  
  StepComponent,  
  null,  
  null  
);  
  
// ...
```

HOOKS

- Resultado



HOOKS

- Código renderizado:

```
...<html> == $0
  ▶ <head>...</head>
  ▼ <body>
    ▼ <div id="root">
      ▼ <div>
        "Olá React!"
        <h1>Hello, Douglas</h1>
        <p>83</p>
      ▼ <div>
        <button>+</button>
        <span>5</span>
        <button>-</button>
      </div>
    </div>
  </div>
  ▶ <script>...</script>
</body>
</html>
```

ECMAScript 2015 E JSX

ECMAScript 2015 E JSX

- Visando aproveitar todo o poder das versões mais recentes do JavaScript, *bundlers* possuem plugins que permitem o uso do ECMAScript 2015 (ES6) e JSX.

```
// ECMAScript 2009 (ES 5) sem JSX
function WelcomeComponent(props) {
  return React.createElement('h1', null, 'Hello, ' + props.name);
}
// ECMAScript 2015 (ES 6) com JSX
const WelcomeComponent = (props) => {
  return <h1>Hello, {props.name}</h1>
}
```

- Para isso, é recomendado o uso de ferramentas como “webpack” ou “vite” (como vimos anteriormente) para auxiliar na “transpilação” e empacotamento do código.
- Felizmente, assim como vimos no backend, existem ferramentas que auxiliam na criação de projetos, trazendo toda a configuração necessária.

CRIANDO PROJETOS COM BUNDLERS

CRIANDO PROJETOS

- **Create React App (webpack)**

```
$ npx create-react-app nome-do-projeto
```

- **Vite**

```
$ npm create vite@latest nome-do-projeto -- --template react
```

- **Next**

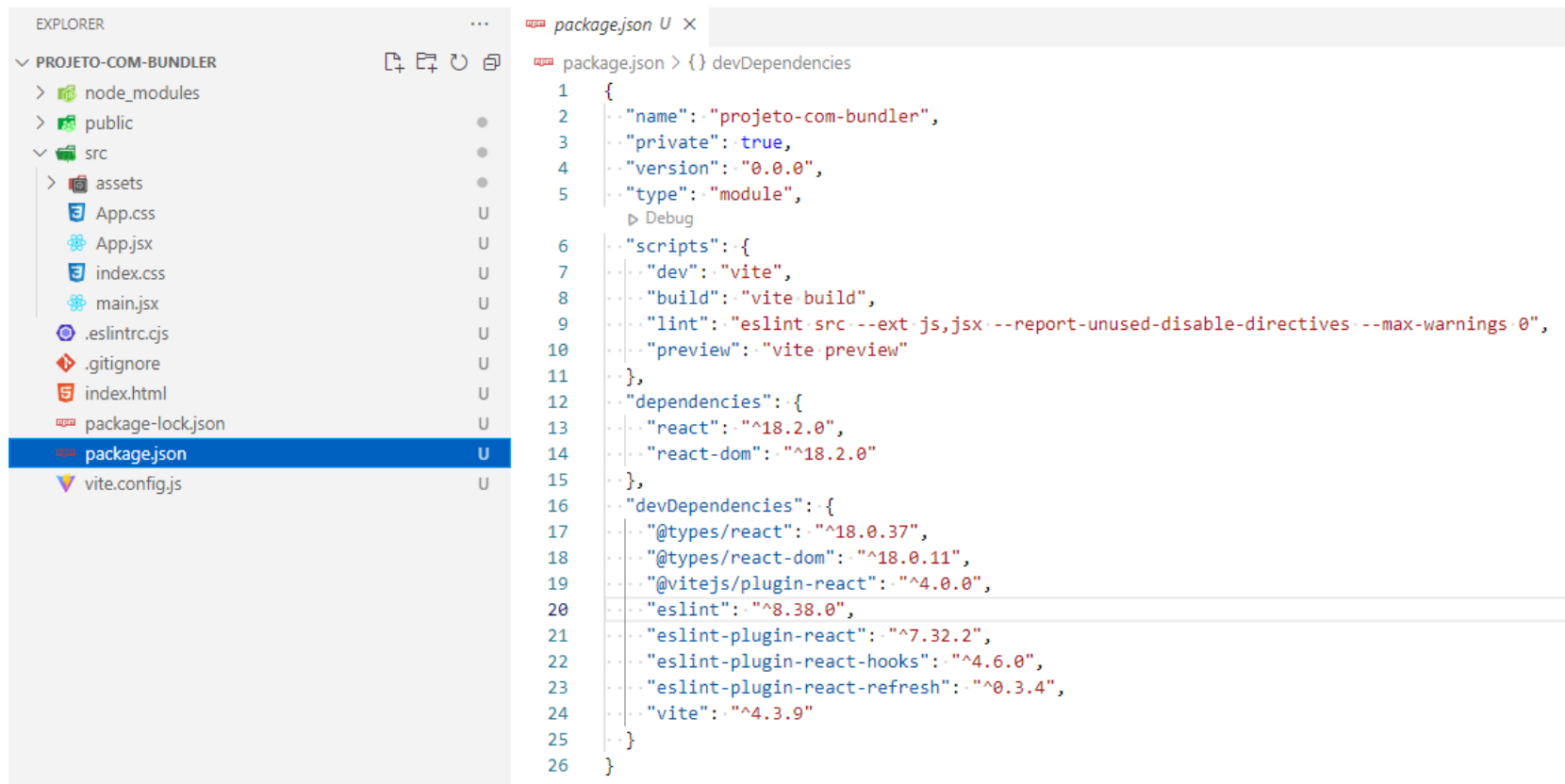
```
$ npx create-next-app@latest
```

- **Gatsby**

```
$ npx gatsby-cli new
```

CRIANDO PROJETOS

- Projeto criado com Vite usando template do React:



The screenshot shows a code editor with two panels. The left panel is the Explorer view, showing a file tree for a project named 'PROJETO-COM-BUNDLER'. The tree includes folders 'node_modules', 'public', and 'src'. The 'src' folder contains 'assets', 'App.css', 'App.jsx', 'index.css', and 'main.jsx'. Below these are files: '.eslintrc.cjs', '.gitignore', 'index.html', 'package-lock.json', 'package.json' (selected), and 'vite.config.js'. The right panel shows the content of 'package.json' with the following JSON structure:

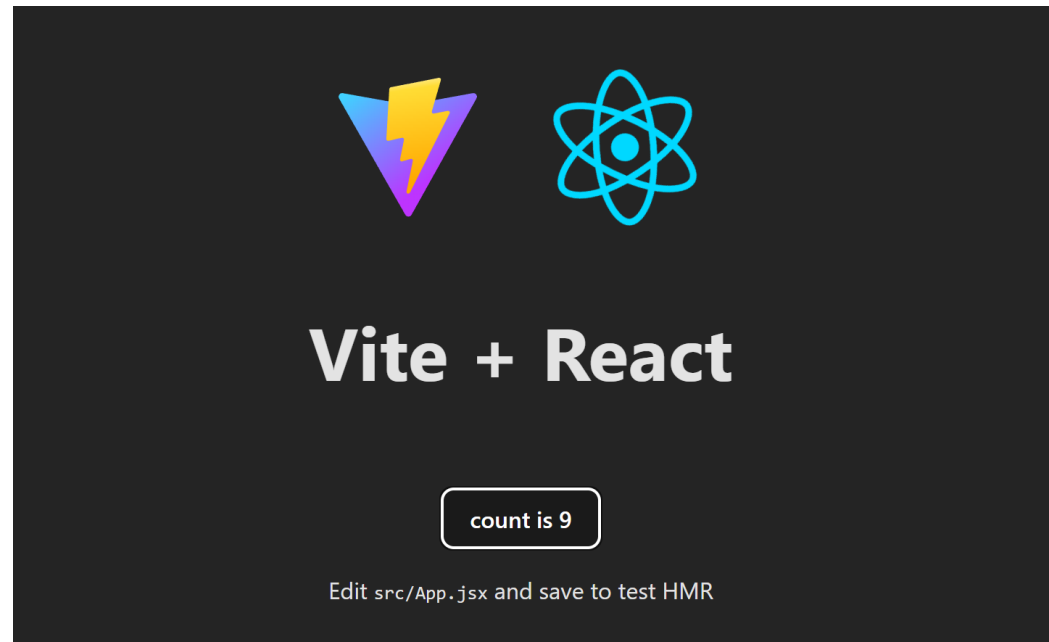
```
1 {
2   "name": "projeto-com-bundler",
3   "private": true,
4   "version": "0.0.0",
5   "type": "module",
6   "scripts": {
7     "dev": "vite",
8     "build": "vite build",
9     "lint": "eslint src --ext js,jsx --report-unused-disable-directives --max-warnings 0",
10    "preview": "vite preview"
11  },
12  "dependencies": {
13    "react": "^18.2.0",
14    "react-dom": "^18.2.0"
15  },
16  "devDependencies": {
17    "@types/react": "^18.0.37",
18    "@types/react-dom": "^18.0.11",
19    "@vitejs/plugin-react": "^4.0.0",
20    "eslint": "^8.38.0",
21    "eslint-plugin-react": "^7.32.2",
22    "eslint-plugin-react-hooks": "^4.6.0",
23    "eslint-plugin-react-refresh": "^0.3.4",
24    "vite": "^4.3.9"
25  }
26 }
```

CRIANDO PROJETOS

- Rodando o projeto:

```
$ npm run dev
```

- Resultado:



ESTADO, PROPIEDAD E CICLO DE VIDA

ESTADO E PROPRIEDADE

- Componentes funcionais vs Componentes de classe

```
function Contador(props) {  
  const [valor, setValor] =  
    React.useState(props.valorInicial);  
  return (  
    <div>  
      {valor}  
    </div>  
  );  
}
```

```
<Contador valorInicial={100} />
```

```
class Contador extends React.Component {  
  constructor(props) {  
    super(props)  
    this.state = {  
      valor: props.valorInicial  
    };  
  }  
  render() {  
    return (  
      <div>  
        Contando: {this.state.valor}  
      </div>  
    );  
  }  
}
```

```
<Contador valorInicial={100} />
```

ESTADO E PROPRIEDADE

- Estado é semelhante às propriedades, porém ele é privado (visível apenas dentro do componente) e totalmente controlado pelo componente de classe ou *hook*.
- Para entender a diferença, vamos criar um componente Relógio que recebe o tempo via “`props`” e em seguida alterá-lo para controlar seu próprio estado.

PROPRIEDADE

- Crie um arquivo e escreva seu componente:

```
import React from 'react';

function Relogio(props) {
  return (
    <div>
      <h1>Olá React!</h1>
      <h2>Hora certa: {props.date.toLocaleTimeString()}</h2>
    </div>
  );
}
```

- Usando:

```
<Relogio date={new Date()} />
```

ESTADO

- Controlando seu próprio estado.

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

const Relogio = () => {
  const [date, setDate] = useState(new Date());

  useEffect(() => {
    const contar = () => {
      setDate(new Date());
    }
    setInterval(contar, 1000);
  }, []);

  // continua
```

```
    return (
      <div>
        <h1>Olá React!</h1>
        <h2>
          Hora certa: {date.toLocaleTimeString()}.
        </h2>
      </div>
    );
  }

  export default Relogio;
```

CICLO DE VIDA E EFEITOS REATIVOS

- **Efeitos** tem ciclo de vida diferentes dos componentes.
- Enquanto componentes pode **montar**, **atualizar** ou **desmontar**, efeitos podem fazer apenas duas coisas:
 - **Iniciar algo**
 - **E, mais tarde, parar algo**
- Esse ciclo pode ocorrer apenas uma vez, quando o componente é **montado** e depois **desmontado**.
- Ou pode ocorrer várias vezes, em casos onde seu **efeito** depende de valores de **estados** ou **propriedades**.

CICLO DE VIDA E EFEITOS REATIVOS

- Efeitos na **montagem e desmontagem**:

```
function Contador() {  
  const [contador, setContador] = React.useState(0);  
  
  React.useEffect(() => {  
    // montagem  
    const interval = setInterval(() => {  
      setContador(cont => cont + 1);  
    }, 1000);  
  
    return () => {  
      // desmontagem  
      clearInterval(interval);  
    }  
  }, []);  
  
  return (  
    <div>  
      Contador: {contador}  
    </div>  
  );  
}
```

CICLO DE VIDA E EFEITOS REATIVOS

- Efeitos na atualização:

```
// ...  
  
React.useEffect(() => {  
  // sempre que "contador" receber um novo valor  
  console.log('Novo contador: ' + contador);  
  return () => {  
    // sempre "contador" deixar de ter o valor  
    console.log('Contador antigo: ' + contador);  
  }  
}, [contador]);  
  
// ...
```

COLEÇÕES DE COMPONENTES

COLEÇÕES DE COMPONENTES

- Com a popularização do React JS, é comum encontrar bibliotecas, frameworks e conjuntos de componentes já prontos para o uso com React.
- Quatro exemplos que devemos citar são:
 - ReactStrap (Bootstrap 5)
 - Ant Design (Alibaba/Aliexpress)
 - Fluent UI (Microsoft)
 - Material UI (implementa o Material Design do Google)
 - Atlas Kit (Atlassian)
 - Chakra UI

COLEÇÕES DE COMPONENTES

- Para instalar o **AntDesign**, basta executar:

```
$ npm install antd
```

- E então os componentes estão prontos para serem utilizados:

```
import { Button } from 'antd';
```

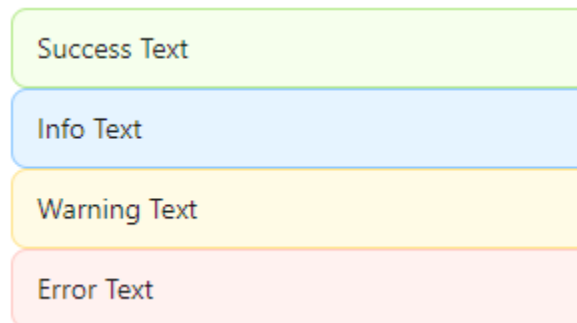
```
<Button onClick={() => { }}>  
  Clique aqui  
</Button>
```


COLEÇÕES DE COMPONENTES

Código:

```
import { Alert } from 'antd';  
  
// ...  
  
<div>  
  <Alert message="Success Text" type="success" />  
  <Alert message="Info Text" type="info" />  
  <Alert message="Warning Text" type="warning" />  
  <Alert message="Error Text" type="error" />  
</div>
```

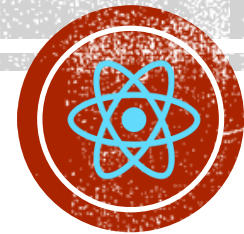
Resultado:



REFERÊNCIAS

- React JS - <http://react.dev>
- JSX - <https://react.dev/learn/writing-markup-with-jsx>
- Hook useState - <https://react.dev/reference/react/useState>
- Hook useEffect - <https://react.dev/reference/react/useEffect>
- webpack - <https://webpack.js.org/>
- Create-react-app - <https://create-react-app.dev/>
- Vite - <https://vitejs.dev/>
- Next - <https://nextjs.org/>
- Podcast - <https://hipsters.tech/react-o-framework-onipresente-hipsters-66/>

OBRIGADO!



Douglas Nassif Roma Junior

 /douglasjunior

 /in/douglasjunior

 douglas@smarppy.com