



Desenvolvimento para Dispositivos Móveis

React

Ferramentas

Android Studio

NodeJS

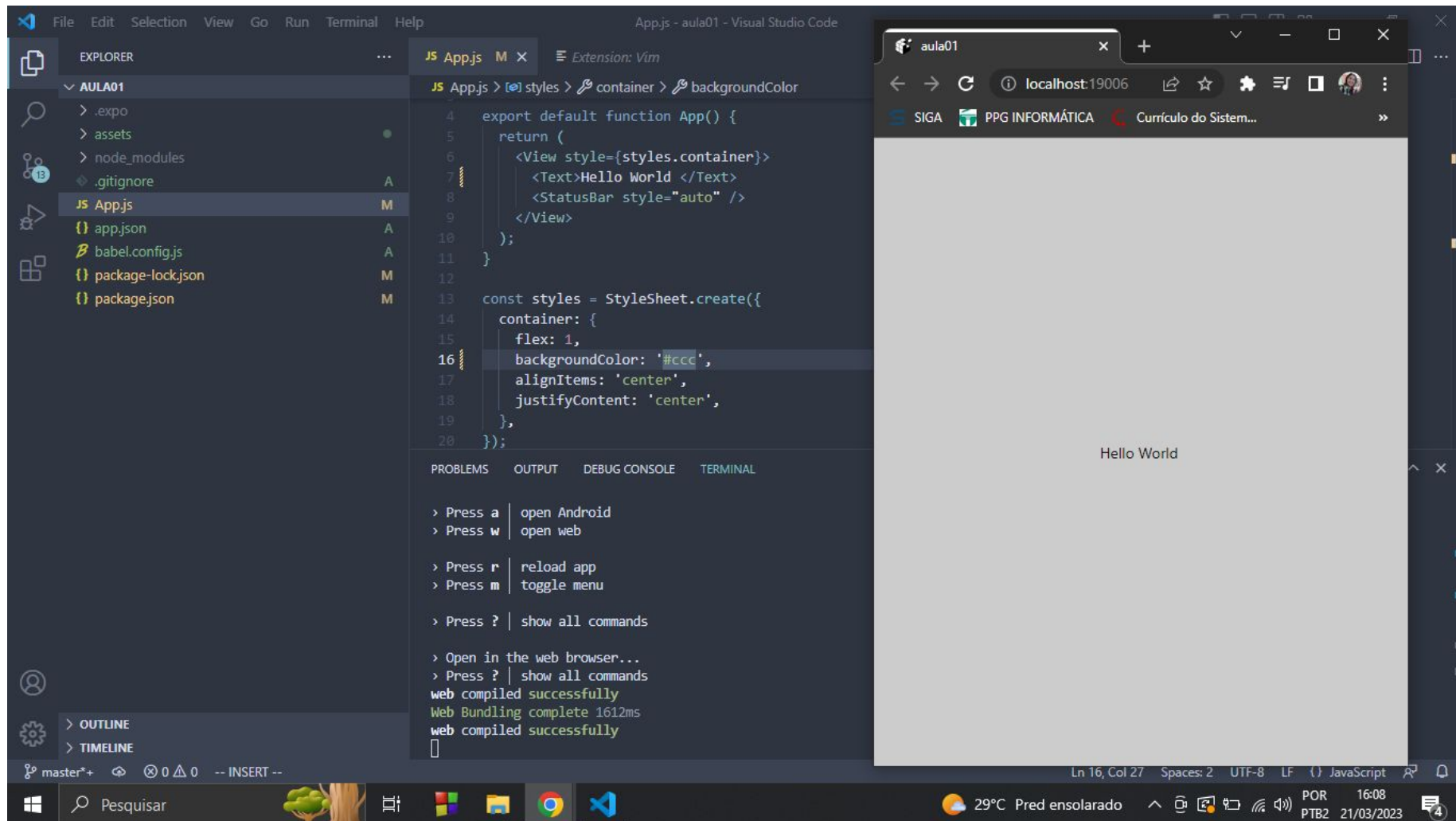
NPM ou Yarn

Expo-cli



Ambiente

interface web



Ambiente

npm start

```
> aula01@1.0.0 start
> expo start
```

```
Starting project at C:\Users\User\Documents\laravel\aula01
Starting Metro Bundler
```



```
> Metro waiting on exp://192.168.0.36:19000
> Scan the QR code above with Expo Go (Android) or the Camera app (iOS)
```

```
> Press a | open Android
> Press w | open web
```

```
> Press j | open debugger
> Press r | reload app
> Press m | toggle menu
```

```
> Press ? | show all commands
```

```
Logs for your project will appear below. Press Ctrl+C to exit.
```

Para hoje

React

Props

JSX e Babel

- extensão de sintaxe JS

- instalação básica

Componentes Funcionais

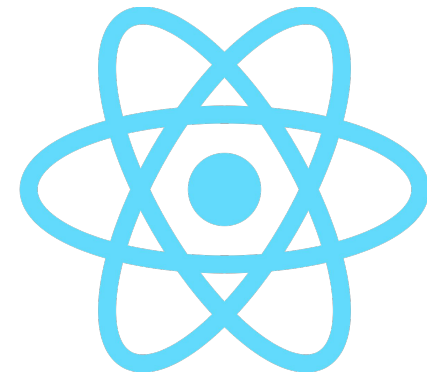
Componentes de Classe

Estado

- Ciclo de vida dos Componentes

Hooks

React X React Native





React

Biblioteca JavaScript para construir interfaces de usuário.

Age no front-end, com adaptações em app mobile

Características base:

- Declarative
- Component-based
- Learn Once, write anywhere



React

React pode ser renderizado em servidores utilizando **Node** e apps Mobile utilizando **React Native**.

Abordagens diferentes!



React

Para compreender o React em sua essência, vamos criar um projeto bastante básico com a biblioteca.

- Crie uma nova pasta chamada **teste-react**;
- Crie um novo arquivo com html básico;
- Adicione uma div com **id="react"**;

React

Adicione as CDNs do react e react-dom e um arquivo JS chamado src/teste-react.js

reactjs.org/docs/add-react-to-a-website.html

```
<!-- ... other HTML ... -->

<!-- Load React. -->
<!-- Note: when deploying, replace "development.js" with "production.min.js". -->
<script src="https://unpkg.com/react@18/umd/react.development.js" crossorigin></script>
<script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js" crossorigin></script>

<!-- Load our React component. -->
<script src="like_button.js"></script>

</body>
```

React

Ao final, seu arquivo HTML deve ficar similar ao seguinte:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Teste React</title>
</head>
<body>
  <div id="react"></div>

  <script src="https://unpkg.com/react@17/umd/react.development.js" crossorigin></script>
  <script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js" crossorigin></script>
  <script src="src/teste-react.js"></script>
</body>
</html>
```

React

No arquivo src/teste-react.js, adicione o seguinte:

```
'use strict';

const el = React.createElement;

function MeuElemento() {
  return(
    el('h1', {}, 'ola mundo em react') //3 argumentos: html, atributos, conteudo
  )
}

const domContainer = document.querySelector('#react');
ReactDOM.render(el(MeuElemento, null, null), domContainer);
```



React

Executando esta página

ola mundo em react



React

Elementos criados pelo método `React.createElement` recebem 3 argumentos:

- Função que produz o elemento (html);
- Atributos que o elemento receberá;
- Conteúdo da tag;

React

É possível utilizar o segundo argumento para passar propriedades (**props**) para o elemento.

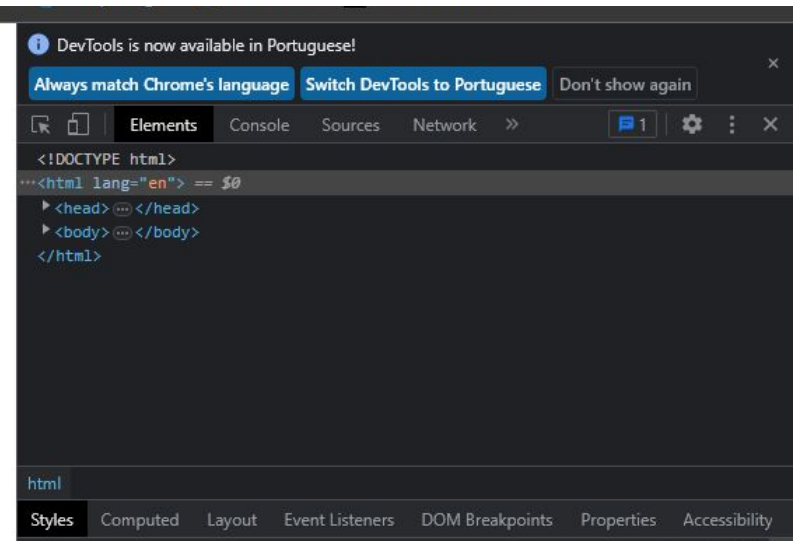
props são valores recebidos por um componente e que não podem ser alterados.

```
function MeuElemento(props) {  
  return(  
    el('h1', {}, props.content) //3 argumentos: html, atributos, conteudo  
  )  
}  
  
const domContainer = document.querySelector('#react');  
ReactDOM.render(el(MeuElemento, {content: 'agora com conteudo'}, null), domContainer);
```

React

props

agora com conteudo





JSX

Escrever componentes através do método `React.createElement` funciona, mas dá muito trabalho.

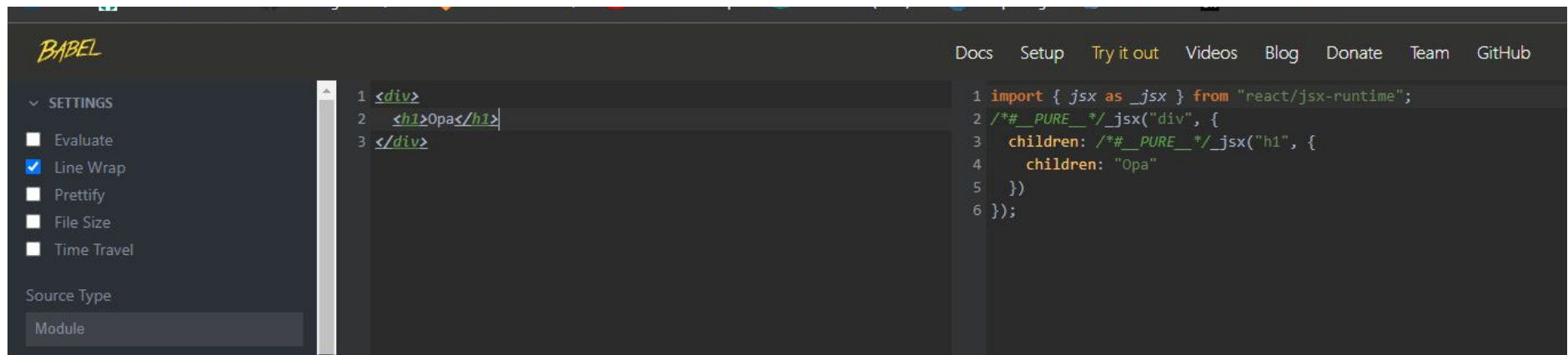
Seria muito mais interessante se pudéssemos escrever código HTML diretamente dentro do nosso JS. Para que isso ocorra, podemos utilizar algo chamado `jsx`.

A biblioteca Babel auxilia nesse processo.

<https://babeljs.io/repl>

JSX

Babel integra versões de JS e permite utilizar recursos específicos através de uma transformação em plain JS.



The screenshot shows the Babel REPL interface. On the left, there is a 'SETTINGS' panel with options: 'Evaluate' (unchecked), 'Line Wrap' (checked), 'Prettify' (unchecked), 'File Size' (unchecked), and 'Time Travel' (unchecked). Below this is a 'Source Type' dropdown menu set to 'Module'. The main editor area is split into two panes. The left pane contains JSX code:

```
1 <div>
2   <h1>Opa</h1>
3 </div>
```

 The right pane shows the transformed plain JavaScript code:

```
1 import { jsx as _jsx } from "react/jsx-runtime";
2 /*#__PURE__*/_jsx("div", {
3   children: /*#__PURE__*/_jsx("h1", {
4     children: "Opa"
5   })
6 });
```

JSX

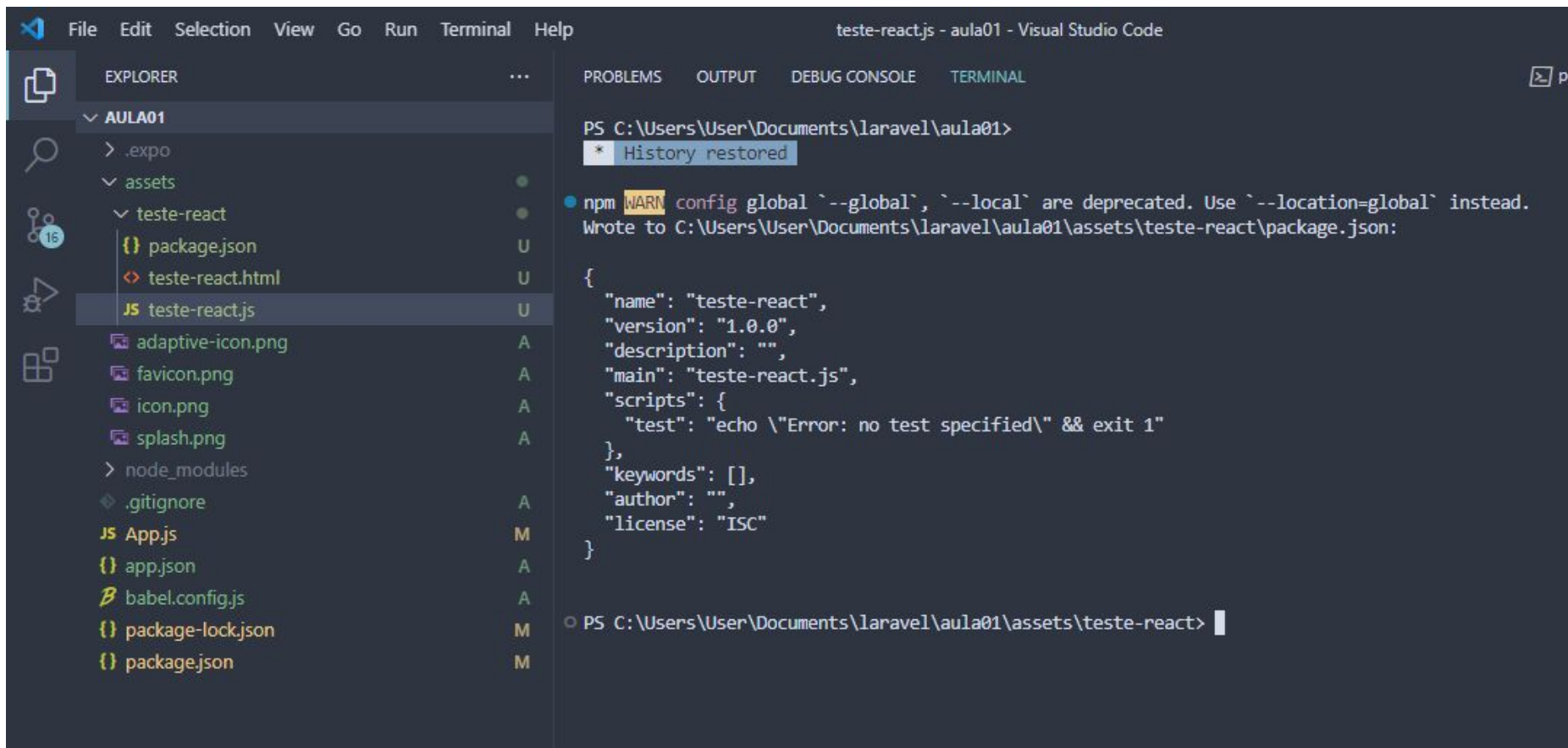
Poderíamos apenas adicionar a CDN dessa biblioteca, mas, como estamos trabalhando em arquivo local, sem servidor, teríamos problemas com CORS(Cross Origin Resource Sharing).

Então, instalaremos a biblioteca Babel para utilizando o **npm**

```
npm init -y # projeto npm
npm install babel-cli babel-preset-react-app@3
npx babel --watch src --out-dir . --presets react-app/prod
```

JSX

npm init -y



```
File Edit Selection View Go Run Terminal Help
teste-react.js - aula01 - Visual Studio Code

EXPLORER
AULA01
├── .expo
├── assets
├── teste-react
│   ├── package.json
│   ├── teste-react.html
│   └── JS teste-react.js
├── adaptive-icon.png
├── favicon.png
├── icon.png
├── splash.png
├── node_modules
├── .gitignore
├── JS App.js
├── app.json
├── babel.config.js
├── package-lock.json
└── package.json

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\User\Documents\laravel\aula01>
* History restored

● npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.
Wrote to C:\Users\User\Documents\laravel\aula01\assets\teste-react\package.json:

{
  "name": "teste-react",
  "version": "1.0.0",
  "description": "",
  "main": "teste-react.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}

○ PS C:\Users\User\Documents\laravel\aula01\assets\teste-react>
```

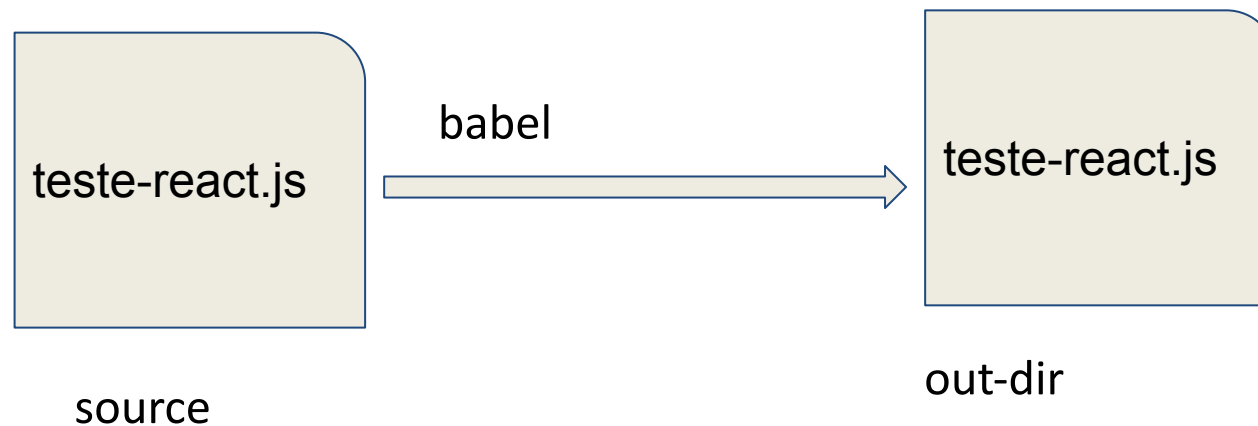
JSX

```
npm install babel-cli babel-preset-react-app@3
```

Instala o pacote Babel 3 react app

```
npx babel --watch teste-react.js --out-dir ./bbl --presets  
react-app/prod
```

Encontra o pacote e executa, na nuvem



JSX

Visual Studio Code interface showing a React project setup. The Explorer sidebar on the left displays the file structure, including the 'teste-react.js' file. The main editor area shows the code for 'MeuElemento' using JSX. The terminal at the bottom shows the command to run the application with Babel.

```
1 'use strict';
2
3 var el = React.createElement;
4
5 function MeuElemento() {
6   // incluir props no argumento para utilizar
7   return(
8     <h6>Teste de Conteudo</h6>
9   )
10 }
11
12 var domContainer = document.querySelector('#react');
13 ReactDOM.render(el(MeuElemento, null, null), domContainer);
14 //ReactDOM.render(el(MeuElemento, {content: 'agora com conteudo'}, null), domContainer);
15
16 /*
```

Terminal output:

```
teste-react.js -> teste-react.js
teste-react.js -> teste-react.js
teste-react.js -> teste-react.js
teste-react.js -> teste-react.js
teste-react.js -> teste-react.js
teste-react.js -> teste-react.js
teste-react.js -> teste-react.js
teste-react.js -> teste-react.js
Deseja finalizar o arquivo em lotes (S/N)? S
PS C:\Users\User\Documents\laravel\aula01\assets\teste-react> npx babel --watch teste-react.js --out-dir ./bbl --presets react-app/prod
```

npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.

```
teste-react.js -> bbl\teste-react.js
teste-react.js -> bbl\teste-react.js
```

JSX

```
4
5 function MeuElemento() {
6   // incluir props no argumento para utilizar
7   return(
8     <h6>Teste de Conteudo</h6>
9   )
10 }
```

```
4
5 function MeuElemento() {
6   // incluir props no argumento para utilizar
7   return React.createElement(
8     'h6',
9     null,
10    'Teste de Conteudo'
11  );
12 }
```

JSX

O comando anterior gera um novo arquivo JS. Logo, aponte o HTML para carregar o novo JS gerado.

Agora, podemos utilizar a sintaxe da extensão JSX. Isso permite a escrita de código HTML que será automaticamente convertido para chamadas do método `React.createElement`:

```
'use strict';

const el = React.createElement;

function MeuElemento(props) {
  return(
    <h1>{props.content}</h1>
  )
}

const domContainer = document.querySelector('#react');
ReactDOM.render(<MeuElemento content="conteúdo com JSX" />, domContainer);
```



JSX

Qual é mais eficiente?

- Arquivo JS compilado (babel)
- CDNs

Flag **watch**

Componentes

Componentes Funcionais

Vimos a criação de um elemento por função:

```
4  
5 function MeuElemento() {  
6   // incluir props no argumento para utilizar  
7   return React.createElement(  
8     'h6',  
9     null,  
10    'Teste de Conteudo'  
11  );  
12  }
```

Componentes de Classe

Componentes

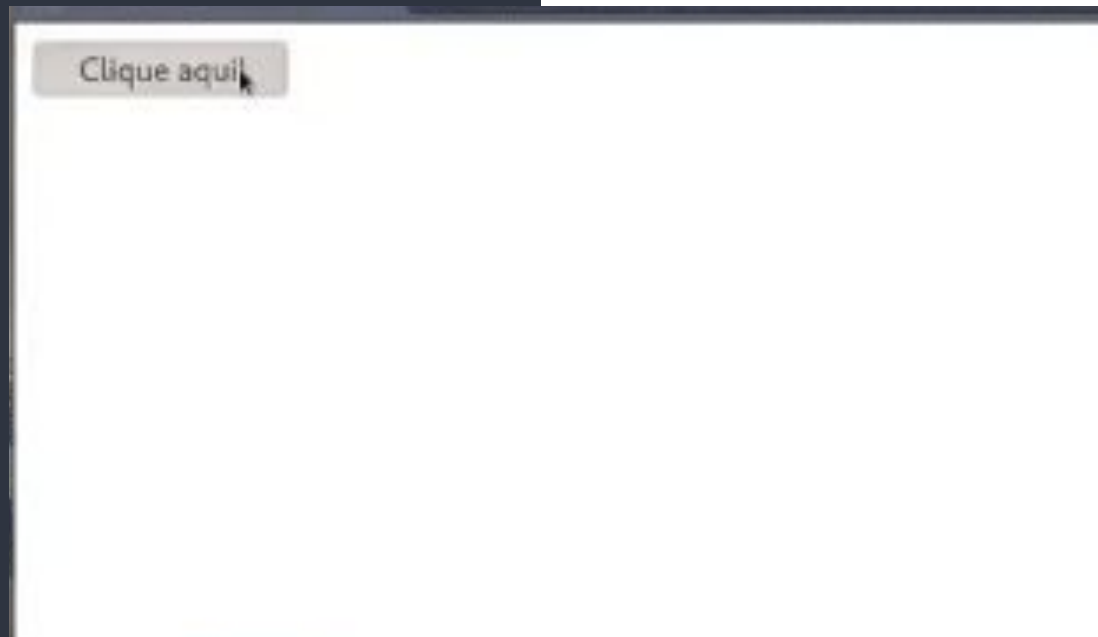
Componentes de Classe

```
12  class MeuBotao extends React.Component {  
13      constructor(props){  
14          super(props);  
15      }  
16  
17      render () {  
18          return(  
19              <button>  
20                  Clique aqui!  
21              </button>  
22          )  
23      }  
24  }  
25
```

Componentes

Componentes de Classe

```
12 class MeuBotao extends React.Component {  
13   constructor(props){  
14     super(props);  
15   }  
16  
17   render () {  
18     return(  
19       <button>  
20         Clique aqui!  
21       </button>  
22     )  
23   }  
24 }  
25
```





Componentes

Geralmente, componentes funcionais são escritos com arrow functions.

```
function MeuElemento(props){  
  const MeuElemento = (props) => {  
    return(  
      <h1> {props.content} </h1>  
    )  
  }  
}
```

Diferenças principais: **Funcional x Classe**



Estados

Estados permitem que se construa variáveis que, quando alteradas, fazem com que o componente seja redesenhado.

Estado é uma informação que fica em observação. Se ele for alterado, o componente é recarregado.

Criaremos na classe Botão um **estado** e **evento**

Estados

```
class MeuBotao extends React.Component {  
  constructor(props){  
    super(props);  
    this.state = {clicked: false};  
  }
```

```
  render () {  
    if(this.state.clicked === true)  
      return('Clicou!');
```

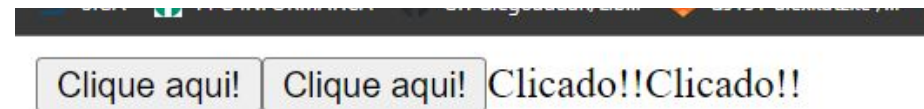
```
  return {  
    <button onClick={() =>{  
      this.setState({clicked: true})  
      console.log('clicou');  
    }}> Clique aqui!  
    </button>  
  }  
}
```

Estados

onClick é o onclick do html, precisamos do cammel case no jsx.

Tentar exemplo com vários botões: cada um mantém seu estado.

```
ReactDOM.render(  
  <div>  
    <MeuBotao />  
    <MeuBotao />  
    <MeuBotao />  
  </div>, domContainer);
```





Estados - Hooks

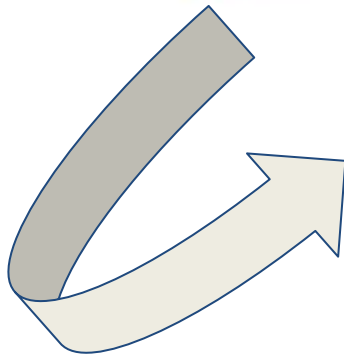
- Props não podem ser alterados. Estados sim.
- Hooks são utilizados para estados (e outras coisas) em componentes funcionais.
 - Problemas em construir estados por classes
- Hooks já são o padrão das novas aplicações React.

Estados - Hooks

Hook de estado em uma função

```
const el = React.createElement;

function MeuElemento(props) {
  return(
    <h1>{props.content}</h1>
  )
}
```



```
// componente funcional com use state
const MeuElemento = (props) => {
  const [clicked, setClicked] = React.useState(false);
  if (clicked)
    return("Clicado com Hook");

  return (
    <button onClick={() => {
      setClicked(true);
    }} >
      Clique aqui!
    </button>
  )
}
```

Estados - Hooks

Hook de estado em uma função

```
JS App.js M  teste-react.html U  JS teste-react.js U  tes
assets > teste-react > JS teste-react2.js > _classCallCheck
59     };
60   }
61   });
62
63   return MeuBotao;
64 }(React.Component);
65
66 var domContainer = document.querySelector('#react');
67 ReactDOM.render(React.createElement(
68   "div",
69   null,
70   React.createElement(MeuElemento, null),
71   React.createElement(MeuElemento, null),
72   React.createElement(MeuBotao, null),
73   React.createElement(MeuBotao, null)
74 ), domContainer);
75
```

Teste React

Arquivo | C:...

SIGA PPG INFORMÁTICA GIT diegoaddan/Lib...

Clicado com Hook Clicado!!Clicado!!



Estados - Hooks

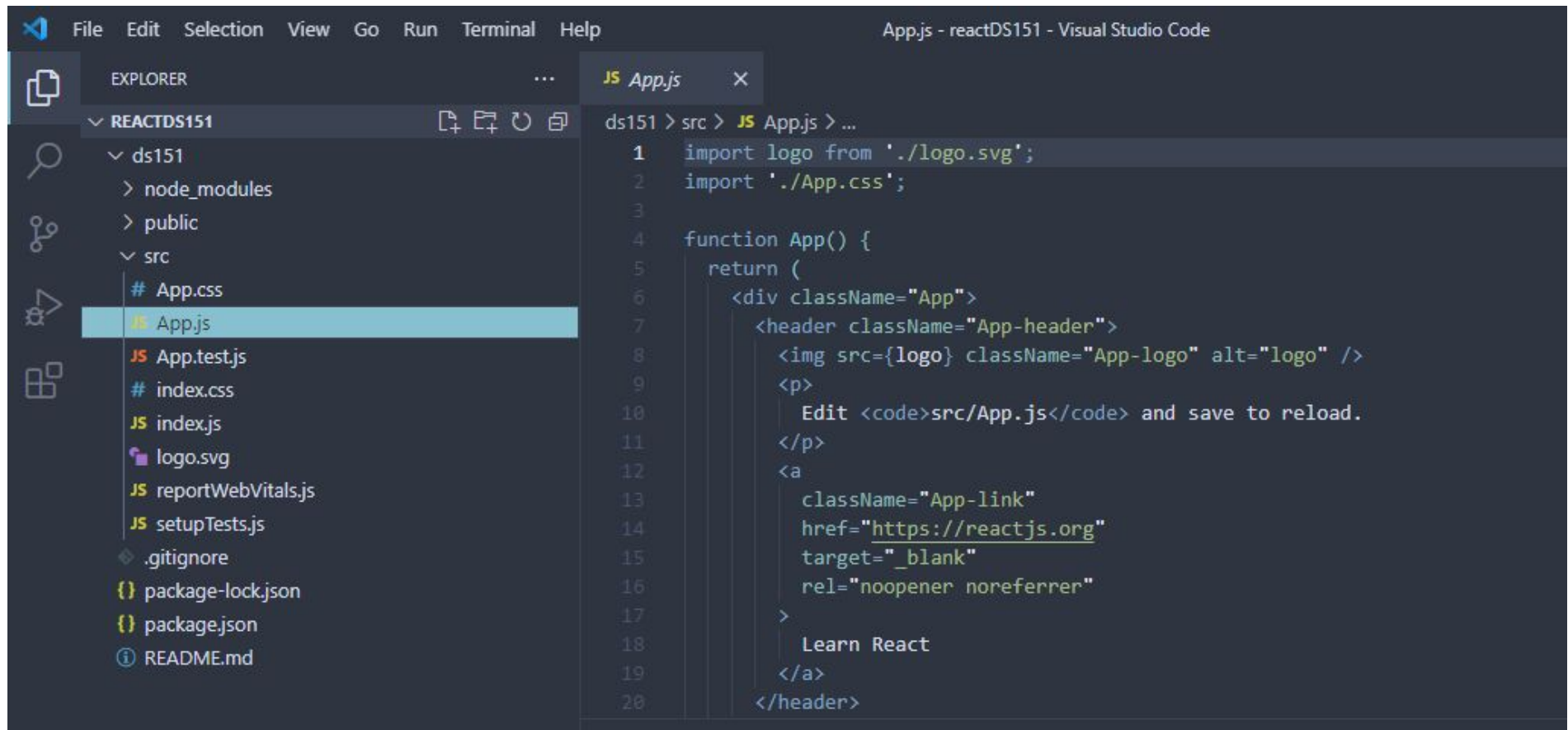
Estados e Hooks tornam dinâmicos o ciclo de vida de nossos componentes.

Existem muitos Hooks além de controle de estado.

Integrados a funções ou classes permitem recursos lógicos complexos e não apenas UI

Criando uma aplicação React

`npx create-react-app nomeApp`



The screenshot shows the Visual Studio Code interface with a project named 'reactDS151'. The Explorer sidebar on the left displays the file structure:

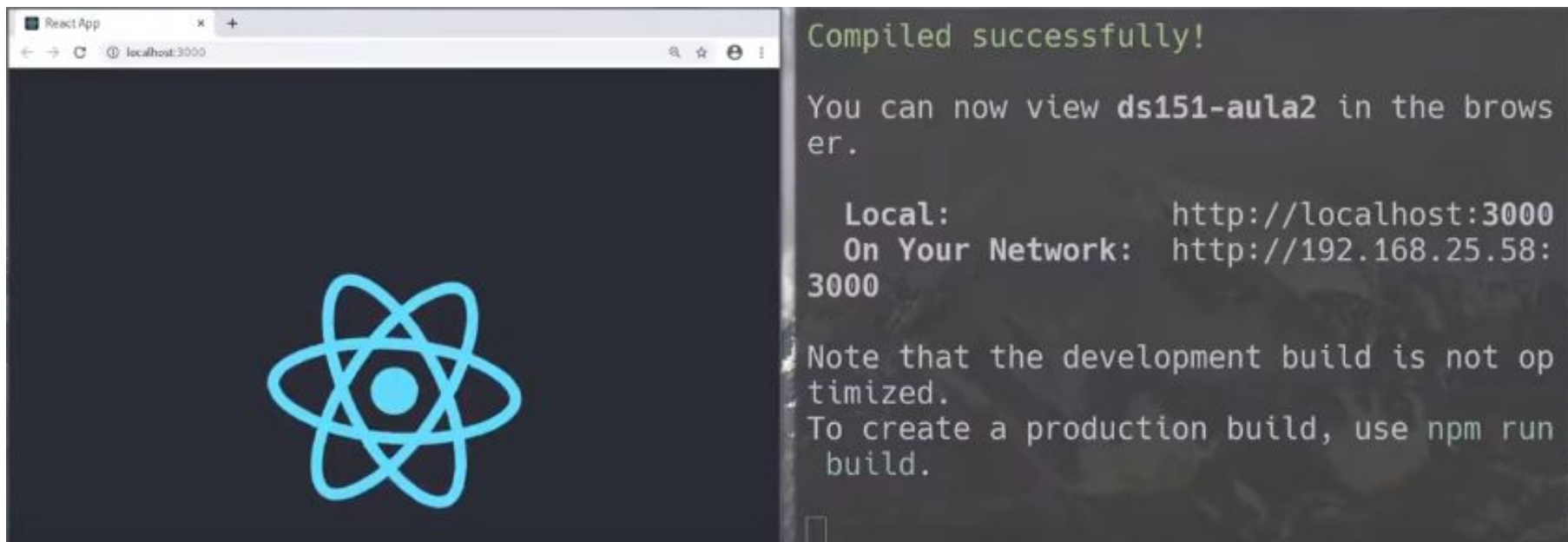
- REACTDS151
 - ds151
 - node_modules
 - public
 - src
 - App.css
 - App.js (selected)
 - App.test.js
 - index.css
 - index.js
 - logo.svg
 - reportWebVitals.js
 - setupTests.js
 - .gitignore
 - package-lock.json
 - package.json
 - README.md

The main editor displays the content of `App.js`:

```
1 import logo from './logo.svg';
2 import './App.css';
3
4 function App() {
5   return (
6     <div className="App">
7       <header className="App-header">
8         <img src={logo} className="App-logo" alt="logo" />
9         <p>
10           Edit <code>src/App.js</code> and save to reload.
11         </p>
12         <a
13           className="App-link"
14           href="https://reactjs.org"
15           target="_blank"
16           rel="noopener noreferrer"
17         >
18           Learn React
19         </a>
20       </header>
```

Criando uma aplicação React

npm start





Criando uma aplicação React

Dentro desta aplicação é possível criar nossos componentes através de arquivos JS próprios.

Ex: Criar um novo arquivo **MeuBotao.js** colocar o código do componente funcional e os imports:


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

```
/// ...
```

```
export default MeuBotao;
```

Criando uma aplicação React

Ex



The screenshot shows a VS Code editor with a dark theme. The Explorer sidebar on the left displays a project structure for 'DS151-AULA2'. It includes folders 'node_modules', 'public', and 'src'. The 'src' folder contains files: 'App.css', 'App.js', 'App.test.js', 'index.css', 'index.js', 'logo.svg', 'MeuBotao.js', 'reportWebVitals.js', 'setupTests.js', and a '.gitignore' file. The 'MeuBotao.js' file is selected and open in the editor. The editor has three tabs: 'MeuBotao.js', 'reportWebVitals.js', and 'App.js'. The code in 'MeuBotao.js' is as follows:

```
src > JS MeuBotao.js > [0] MeuBotao
1  import React, { useState } from 'react';
2
3  // function MeuElemento(props){
4  const MeuBotao = (props) => {
5    const [clicked, setClicked] = React.useState(false);
6    if (clicked)
7      return ("Clicado com Hook");
8
9    return <>
10      <button onClick={() => {
11        setClicked(true);
12      }} >
13        Clique aqui!
14      </button>
15    </>
16  }
```



React Native

Importa todos os conceitos do React mas com recursos adicionais

Os componentes do Native saberão como se comportar no ambiente mobile.

Overview de um projeto React Native
`expo init ds151 --npm`