




React JS

Douglas Nassif Roma Junior

 /douglasjunior

 /in/douglasjunior

 douglasjunior.me

 nassifroma@gmail.com

Agenda

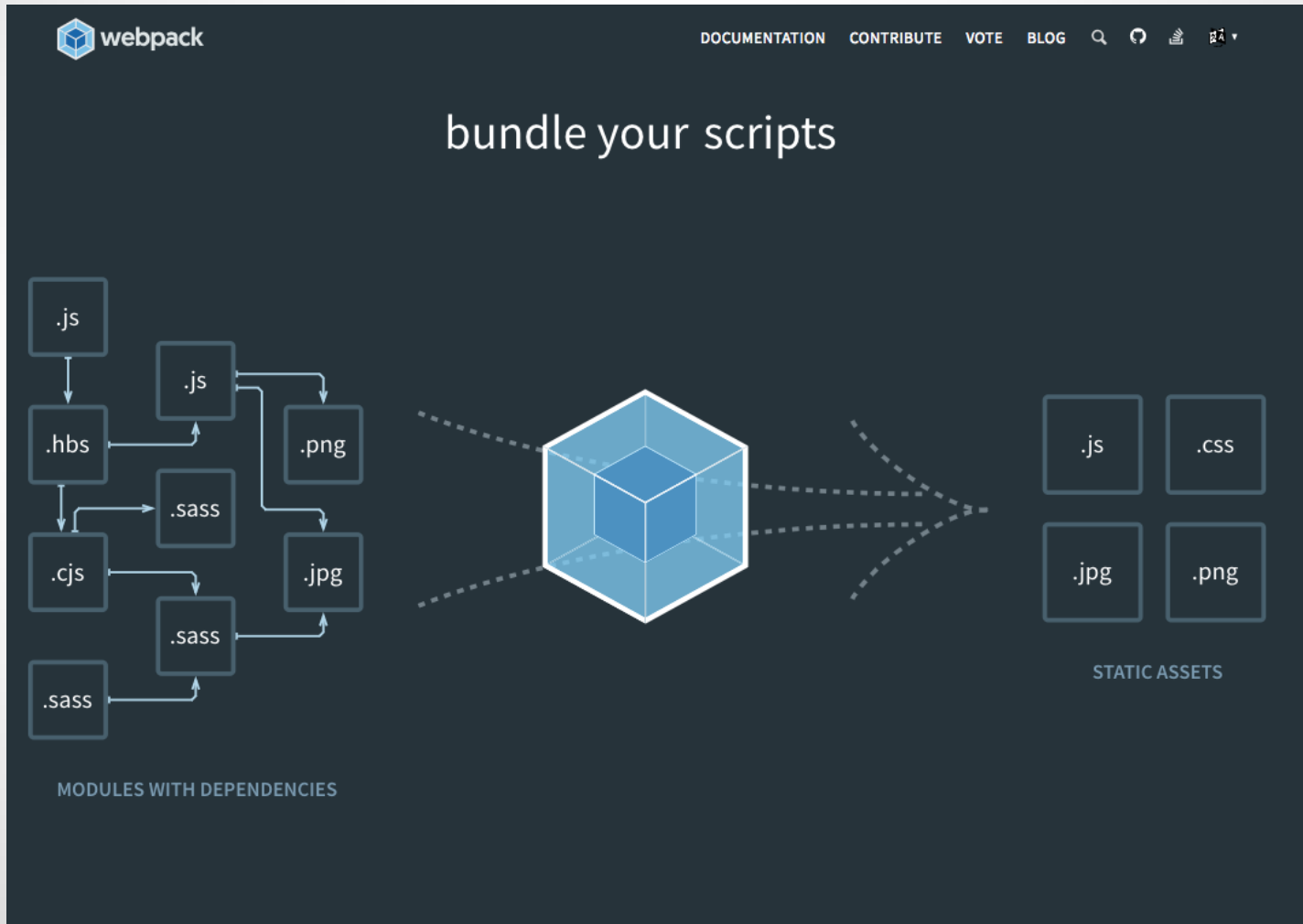
- webpack
 - Introdução ao webpack
 - Instalação e uso básico
 - Configuração (Loaders, Plugins)
- React JS
 - Introdução ao React JS
 - Componentes (funcional, classe, hooks)
 - ECMAScript 6 (2015) e JSX
 - Estado e Ciclo de Vida
 - Coleções de Componentes
- React Router
 - Introdução ao React Router
 - Parâmetros de URL
 - Rotas privadas
 - Links customizados
 - Prevenindo transições
 - Rotas desconhecidas (404)
 - Transições animadas
- Tópicos avançados
 - Chamadas para APIs
 - Renderizando tabelas
- Referências

webpack

Introdução ao webpack

- `webpack` (com “w” minúsculo) é um empacotador de código para projetos web, assim como o [browserify](#).
- O objetivo do `webpack` não é apenas unir todos os arquivos JS e CSS em um único pacote, mas também possibilitar a divisão do projeto em módulos reaproveitáveis e com isso auxiliando no trabalho em equipe.
- Entretanto, o `webpack` não é recomendado apenas para projetos grandes, tornando-se uma ferramenta muito útil também para projetos pequenos.

Introdução ao webpack



Instalando o webpack

- Essencialmente o `webpack` deve ser instalado como uma dependência de desenvolvimento do seu projeto.

```
$ npm install --dev webpack webpack-cli
```

- Em alguns casos também pode ser interessante instalar o `webpack` globalmente, para facilitar a execução de scripts e testes rápidos.

```
$ npm install -g webpack webpack-cli
```

Projeto sem webpack

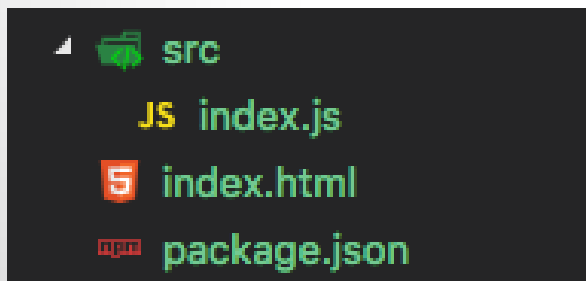
- Para entender o real papel do webpack, primeiro vamos criar uma estrutura tradicional de um projeto web.

```
$ mkdir meu-projeto-webpack
```

```
$ cd meu-projeto-webpack
```

```
$ npm init -y
```

- Em seguida crie a seguinte estrutura de arquivos:



Projeto sem webpack

- index.html

```
<html>

<head>
  <meta charset="utf-8" />
  <title>Introdução ao webpack</title>
  <script src="https://unpkg.com/lodash@4.16.6"></script>
</head>

<body>
  <script src="./src/index.js"></script>
</body>

</html>
```


Projeto sem webpack

- src/index.js

```
function component() {  
  var element = document.createElement('div');  
  // Lodash é utilizado como variável global por meio da  
  // declaração do <script> no index.html  
  element.innerHTML = _.join(['Olá', 'webpack', '!'], ' ');  
  return element;  
}  
  
document.body.appendChild(component());
```

Projeto sem webpack

- Resultado:



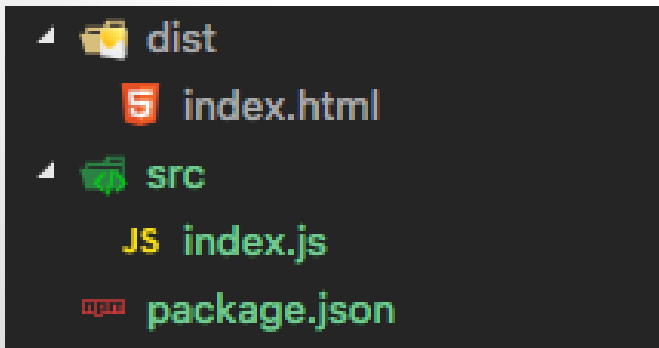
Uso básico do webpack

- Para converter este projeto para utilizar o webpack, é preciso instalar as dependências necessárias:

```
$ npm install -D webpack webpack-cli
```

```
$ npm install lodash
```

- Atualizar a estrutura do projeto para:



Uso básico do webpack

- dist/index.html (novo)

```
<html>

<head>
  <meta charset="utf-8" />
  <title>Introdução ao webpack</title>
</head>

<body>
  <script src="bundle.js"></script>
</body>

</html>
```

Uso básico do webpack

- src/index.js (novo)

```
import _ from 'lodash';

function component() {
  var element = document.createElement('div');

  // Lodash agora é importado do node_modules
  element.innerHTML = _.join(['Olá', 'webpack', '!'], ' ');

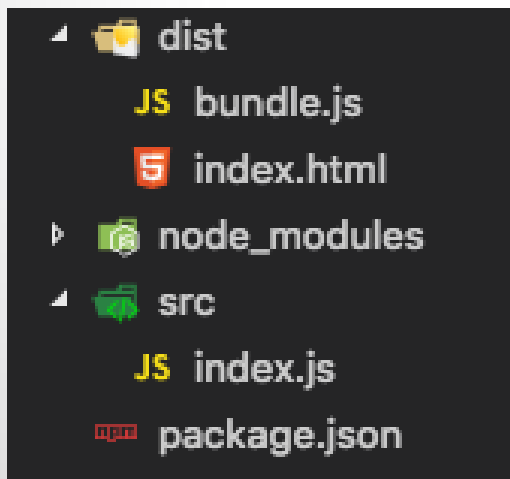
  return element;
}

document.body.appendChild(component());
```

Uso básico do webpack

- Por fim, para empacotar o projeto execute o comando:

```
$ npx webpack --mode production -o dist/bundle.js
```



Uso básico do webpack

- Resultado:



Configuração do webpack

- A maioria dos projetos pode precisar de uma configuração mais complexa, para isso o webpack suporta a criação de arquivos de configuração.
- Para utilizar o arquivo de configuração, basta criar um arquivo chamado `webpack.config.js` na raiz do projeto e acrescentar o seguinte conteúdo.

```
const path = require('path');

module.exports = {
  entry: './src/index.js',
  mode: 'production',
  output: {
    filename: 'bundle.js',
    path: path.resolve(__dirname, 'dist')
  }
};
```

```
$ npx webpack --config webpack.config.js
```


Loaders

- Loaders são auxiliares que permitem que o webpack saiba trabalhar com outros tipos de arquivos além do JavaScript. Uma vez que o loader é configurado, você está dizendo ao webpack o que deve ser feito quando um arquivo daquele formato for encontrado.
- Por exemplo, para possibilitar que o webpack saiba carregar arquivos do tipo CSS, é preciso instalar e configurar os seguintes módulos:

```
$ npm install -D style-loader css-loader
```

Loaders

- webpack.config.js

```
const path = require('path');

module.exports = {
  // ...
  module: {
    rules: [
      {
        test: /\.css$/,
        use: [
          'style-loader',
          'css-loader'
        ]
      }
    ]
  }
};
```

Loaders

- Uma vez que o loader de CSS foi configurado, é possível importar os arquivos de estilo utilizando o `import`.

src/index.js

```
import _ from 'lodash';
import './styles.css';

function component() {
  // ...
  element.classList.add('hello');
  // ...
}
```

src/styles.css

```
.hello {
  color: red;
}
```

Loaders

- Loaders também podem ser utilizados para carregar imagens.

```
$ npm install -D file-loader
```

webpack.config.js

```
const path = require('path');

module.exports = {
  // ...
  module: {
    rules: [
      // ... ,
      {
        test: /\. (png|svg|jpg|gif) $/,
        use: ['file-loader']
      }
    ]
  }
};
```

Loaders

- Uma vez que o arquivo de imagem for importado, será retornada uma `String` contendo o caminho para o arquivo.

`src/index.js`

```
// ...
import Icon from './icon.svg';

function component() {
  // ...

  var myIcon = new Image();
  myIcon.src = Icon;

  element.appendChild(myIcon);

  return element;
}

document.body.appendChild(component());
```

Plugins

- Outro recurso interessante do `webpack` é a possibilidade de adição de `plugins`. Isso permite que você possa adicionar funcionalidades extras que o `webpack` não atende por si só.
- Por exemplo, podemos adicionar o `html-webpack-plugin` para gerenciar a criação do `index.html` à partir de um template.

```
$ npm install -D html-webpack-plugin
```

Plugins

- Configurando o `html-webpack-plugin`:

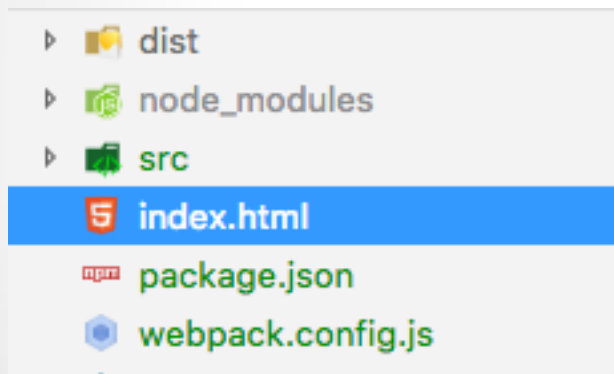
`webpack.config.js`

```
const HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {
  // ...,
  plugins: [
    new HtmlWebpackPlugin({
      template: './index.html'
    })
  ],
};
```

Plugins

- Então, o arquivo de template `index.html` agora deve ficar na raiz do projeto.



```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <title>Projeto webpack!</title>
</head>

<body>
</body>

</html>
```


React JS

Introdução ao React JS

[Docs](#)[Tutorial](#)[Blog](#)[Comunidade](#)[🔍 Buscar docs](#)[v16.8.6](#)[Languages](#)[GitHub](#)

React

Uma biblioteca JavaScript para criar interfaces de usuário

[Comece a Usar](#)[Faça o Tutorial >](#)

Declarativo

React faz com que a criação de UIs interativas seja uma tarefa fácil. Crie views simples para cada estado na sua aplicação, e o React irá

Baseado em componentes

Crie componentes encapsulados que gerenciam seu próprio estado e então, combine-os para criar UIs complexas.

Aprenda uma vez, use em qualquer lugar

Não fazemos suposições sobre as outras tecnologias da sua stack, assim você pode

Introdução ao React JS

- **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 JS

- **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 JS

- **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**.

```
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>
    );
  }
}
```

Introdução ao React JS

- Para utilizar o ReactJS na web, você precisa apenas importar as bibliotecas React e React-DOM.

```
<script crossorigin src="https://unpkg.com/react@16.2.0/umd/react.development.js"></script>  
<script crossorigin src="https://unpkg.com/react-dom@16.2.0/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>
```

Componentes

- Os componentes permitem que você divida a 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”) e retornam elementos descrevendo o que deve aparecer na tela.

LabelComponent
ItemComponent

NETFLIX

Navegar ▾

Kids

TODA SEMANA

UM NOVO EPISÓDIO

NOVOS EPISÓDIOS

Populares na Netflix



SectionComponent

Em alta



ORIGINAIS NETFLIX



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 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.**
- À partir da versão 16, função de criação de classes foi movida para um pacote separado, então é precisamos importar:

```
<script crossorigin src="https://unpkg.com/create-react-class@15.6.2/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);
  }
});
```

ECMAScript 2015 e JSX

- Visando aproveitar todo o poder das versões mais recentes do JavaScript, o React dispõe de um *plugin* para o **Babel** que permite o uso do ECMAScript 2015 (ES6) e JSX.

```
// ECMAScript 5
function WelcomeComponent(props) {
  return React.createElement('h1', null, 'Hello, ' + props.name);
}

// ECMAScript 2015 com JSX
const WelcomeComponent = (props) => (
  <h1>Hello, {props.name}</h1>
)
```

- Para isso, é recomendado o uso de ferramentas como “webpack” ou “Browserify” para auxiliar na “transpilação” e empacotamento do código.
- Felizmente, existe uma ferramenta chamada “create-react-app” que faz todo o trabalho necessário com um único comando.

Create-React-App

- Ferramenta de linha de comando para auxiliar na criação e configuração de um projeto “webpack” com ReactJS.
- Adicionalmente, `create-react-app` traz um conjunto de configurações e módulos auxiliares que facilitam o processo de desenvolvimento e implantação.
- Para instalar o `create-react-app` utilize:

```
$ npm install -g create-react-app
```

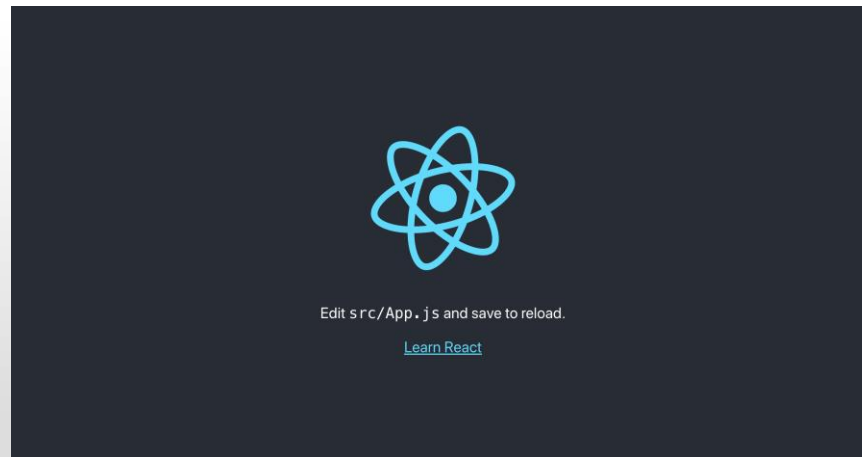
Create-React-App

- Para criar um novo projeto, execute o comando:

```
$ create-react-app meu-projeto-reactjs
```
- Entre na pasta do projeto e execute e inicie o webpack-dev-server em modo de desenvolvimento:

```
$ cd meu-projeto-reactjs
```

```
$ npm start
```



Estado e Ciclo de Vida

- 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.

Estado e Ciclo de Vida

- Recebendo data e hora via “props”.

```
import React from 'react';
import ReactDOM from 'react-dom';

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

function contar() {
  ReactDOM.render(<Relogio date={new Date()} />,
    document.getElementById('root'));
}

setInterval(contar, 1000);
```


Estado e Ciclo de Vida

- Controlando seu próprio estado.

```
import React, { Component } from 'react';
import ReactDOM from 'react-dom';

class Relogio extends Component {

  state = {
    date: new Date()
  }

  componentDidMount() {
    setInterval(this.contar.bind(this),
      1000);
  }

  contar() {
    this.setState({ date: new Date() });
  }

  // continua
```

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

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

Estado e Ciclo de Vida

- Componentes de classe também possuem eventos que podem ser utilizados para detectar momentos do ciclo de vida dos componentes.
- **Montagem:** Eventos disparados quando o componente é criado e anexado ao DOM.
 - `constructor()`
 - ~~`componentWillMount()`~~
 - `render()`
 - `componentDidMount()`

Estado e Ciclo de Vida

- **Atualização:** Uma atualização pode ocorrer por uma mudança nas `props` ou no estado. Estes eventos são disparados quando o componente é re-renderizado.
 - ~~`componentWillReceiveProps()`~~
 - `shouldComponentUpdate()`
 - ~~`componentWillUpdate()`~~
 - `render()`
 - `componentDidUpdate()`
- **Desmontagem:** Evento chamado quando o componente será destruído.
 - `componentWillUnmount()`

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 4)
 - [Ant Design](#) (Alibaba/Aliexpress)
 - [Office UI Fabric](#) (Microsoft)
 - [Material UI](#) (implementa o Material Design)
 - [Blueprint](#)
 - [Atlas Kit](#) (Atlassian)

Coleções de Componentes

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

```
$ npm install --save bootstrap reactstrap
```

- E então, importar o Bootstrap 4 globalmente no `index.js`:

```
import 'bootstrap/dist/css/bootstrap.css';
```

Coleções de Componentes

```
import { Alert } from 'reactstrap';  
...  
<div className="App">  
  <Alert color="primary">  
    This is a primary alert — check it out!  
  </Alert>  
  <Alert color="secondary">  
    This is a secondary alert — check it out!  
  </Alert>  
  <Alert color="success">  
    This is a success alert — check it out!  
  </Alert>  
  <Alert color="danger">  
    This is a danger alert — check it out!  
  </Alert>  
  <Alert color="warning">  
    This is a warning alert — check it out!  
  </Alert>  
</div>
```



Edit `src/App.js` and save to reload.

[Learn React](#)

This is a primary alert — check it out!

This is a secondary alert — check it out!

This is a success alert — check it out!

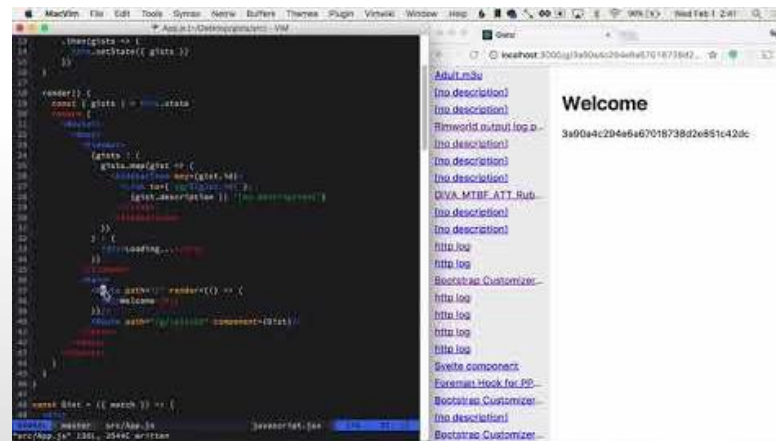
This is a danger alert — check it out!

This is a warning alert — check it out!

React Router

Introdução ao React Router

- React Router é uma coleção de componentes de navegação que compõem declarativamente com sua aplicação.
- Se você quer ter URLs navegáveis para seu aplicativo Web ou uma maneira componentizada para navegar no React Native, o React Router funciona onde quer que o React JS esteja renderizando.



Introdução ao React Router

- Para instalar o React Router para Web, basta executar:

```
$ npm install react-router-dom
```

```
import {  
  HashRouter as Router,  
  Route,  
  Link,  
} from 'react-router-dom';  
  
import Home from './pages/Home';  
import Tasks from './pages/Tasks';  
import About from './pages/About';
```

```
<Router>  
  <ul>  
    <li><Link to="/">Home</Link></li>  
    <li><Link to="/tasks">Tarefas</Link></li>  
    <li><Link to="/about">Sobre</Link></li>  
  </ul>  
  <Route exact path="/" component={Home} />  
  <Route path="/tasks" component={Tasks} />  
  <Route path="/about" component={About} />  
</Router>
```

Parâmetros de URL

- React Router permite que parâmetros sejam passados entre as rotas, usando parâmetros de URL.
- Para usar parâmetros de URL, basta usar o caractere ":" seguido do nome do parâmetro.

```
const Hero = ({ match }) => (  
  <div>  
    <h3>Hero: {match.params.heroId}</h3>  
  </div>  
)
```

```
<Router>  
  <div>  
    <h2>Marvel</h2>  
    <ul>  
      <li><Link to="/ironman">Iron Man</Link></li>  
      <li><Link to="/strange">Strange</Link></li>  
      <li><Link to="/captain">Captain</Link></li>  
      <li><Link to="/spide-man">Spider Man</Link></li>  
    </ul>  
    <Route path("/:heroId" component={Hero} />  
  </div>  
</Router>
```

Rotas Privadas

- Usando o componente `Redirect` é possível redirecionar o usuário para uma rota determinada.
- Esta abordagem é útil, por exemplo, quando alguns dos componentes exigem autenticação do usuário.

```
const PrivateRoute = ({ component: Component, ...others }) => {  
  return (  
    <Route {...others} render={props => (  
      fakeAuth.isAuthenticated  
        ? <Component {...props} />  
        : <Redirect to={{  
            pathname: '/login',  
            state: { from: props.location }  
          }} />  
      )} />  
    );  
  }  
}
```

```
<PrivateRoute path="/protected" component={Tasks} />
```

Prevenindo Transações

- Em alguns casos, como preenchimento de formulários, pode ser interessante prevenir que o usuário navegue para outra rota da aplicação, sem antes salvar o trabalho em progresso.
- Exemplo de uso do componente `Prompt`:

```
<Prompt
  when={isBlocking}
  message={location => (
    `Tem certeza que deseja navegar para ${location.pathname}?`
  )}
/>
```

Rotas Desconhecidas (404)

- Em uma aplicação Web pode ser interessante exibir uma mensagem amigável para um usuário. Com React Router, basta utilizar o componente `Switch`.
- Com o `Switch`, o React Router vai renderizar a primeira rota que combinar com a URL.

```
<Switch>
  <Route path="/" exact component={Home} />
  <Redirect from="/old-match" to="/will-match" />
  <Route path="/will-match" component={WillMatch} />
  <Route component={NoMatch} /> { /* página 404 */ }
</Switch>
```

Transições Animadas

- Usando um módulo adicional ao React, é possível criar animações de transição com o React Router. Basicamente, este módulo adicionado irá adicionar/remover classes CSS em seus componentes, criando os efeitos desejados.

```
$ npm install --save react-transition-group
```

```
import { TransitionGroup, CSSTransition }
  from 'react-transition-group'
import './AnimationExample.css';

<TransitionGroup>
  <CSSTransition
    key={location.key}
    classNames="fade"
    timeout={{ exit: 300, enter: 300 }}
  >
    <Route
      location={location}
      key={location.key}
      path="/sua/url/aqui"
      component={Comp}
    />
  </CSSTransition>
</TransitionGroup>
```

```
.fade-enter {
  opacity: 0;
  z-index: 1;
}

.fade-enter.fade-enter-active {
  opacity: 1;
  transition: opacity 300ms ease-in;
}

.fade-exit {
  opacity: 1;
}

.fade-exit.fade-exit-active {
  opacity: 0;
  transition: opacity 300ms ease-in;
}
```

Chamadas para APIs

- Em uma aplicação Web real certamente será necessário fazer o **consumo de APIs**, ou trocar qualquer tipo de informações com um serviço de *backend*.
- Como foi dito no início, não exige (e não possui), nenhuma forma específica com que isso deve acontecer.
- Isso quer dizer que você pode utilizar sua biblioteca ou forma preferida para que isso aconteça, como:
 - [Fetch API](#)
 - [Axios](#)
 - [Superagent](#)
 - [Request](#)
 - E até [jQuery](#)

Chamadas para APIs

- Para nosso exemplo, vamos utilizar o `axios`:

```
$ npm install --save axios
```

- Crie um novo componente para representar a página de listagem de tarefas:

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

export default class ListaTarefas extends Component {
  render() {
    return (
      <div>
        Lista de tarefas
      </div>
    );
  }
}
```


Chamadas para APIs

- Assim que o componente for criado, então podemos realizar a chamada da API.
- Observe que o resultado da requisição será armazenado no `state` para que possa ser renderizado.

```
import axios from 'axios';

...
componentDidMount() {
  axios.get('https://jsonplaceholder.typicode.com/todos')
    .then(response => {
      const { data } = response;
      this.setState({ todos: data })
    })
    .catch(err => {
      console.warn(err)
    })
}
...
```

Chamadas para APIs

- Renderizando o resultado da requisição como JSON em formato texto.

```
...  
render() {  
  const { todos } = this.state;  
  return (  
    <div>  
      Lista de tarefas  
      <pre>{JSON.stringify(todos, null, 2)}</pre>  
    </div>  
  );  
}  
...
```

Renderizando tabelas

- Uma vez que a lista de tarefas foi carregada no state, podemos então aproveitar os poderes do **JavaScript** para renderizar uma **tabela legível**.

```
...
renderRow = (todo, index) => {
  return (
    <tr>
      <td>{todo.id}</td>
      <td>{todo.userId}</td>
      <td>{todo.title}</td>
      <td>
        {todo.completed ? 'Sim' : 'Não'}
      </td>
    </tr>
  )
}
...

...
render() {
  const { todos } = this.state;
  return (
    <div>
      <h1>Lista de tarefas</h1>
      <Table>
        <thead>
          <tr>
            <th>#</th>
            <th>Usuário</th>
            <th>Título</th>
            <th>Concluída</th>
          </tr>
        </thead>
        <tbody>
          {todos.map(this.renderRow)}
        </tbody>
      </Table>
    </div>
  );
}
...
```

60

```

...
renderRow = (todo, index) => {
  return (
    <tr>
      <td>{todo.id}</td>
      <td>{todo.userId}</td>
      <td>{todo.title}</td>
      <td>
        {todo.completed ? 'Sim' : 'Não'}
      </td>
    </tr>
  )
}
...

```

```

...
render() {
  const { todos } = this.state;
  return (
    <div>
      <h1>Lista de tarefas</h1>
      <Table>
        <thead>
          <tr>
            <th>#</th>
            <th>Usuário</th>
            <th>Título</th>
            <th>Concluída</th>
          </tr>
        </thead>
        <tbody>
          {todos.map(this.renderRow)}
        </tbody>
      </Table>
    </div>
  );
}
...

```

Referências


- webpack - <https://webpack.js.org/>
- webpack guides - <https://webpack.js.org/guides/>
- Lodash - <https://lodash.com/>
- React JS - <http://reactjs.org>
- JSX - <https://reactjs.org/docs/introducing-jsx.html>
- Documentação React - <https://reactjs.org/docs/getting-started.html>
- Create-react-app - <https://github.com/facebookincubator/create-react-app>
- Podcast - <https://hipsters.tech/react-o-framework-onipresente-hipsters-66/>
- React Router - <https://reacttraining.com/react-router/>
- React Router DOM - <https://reacttraining.com/react-router/web/guides/philosophy>
- React Transition Group - <https://github.com/reactjs/react-transition-group>

Obrigado!

Douglas Nassif Roma Junior

 /douglasjunior

 /in/douglasjunior

 douglasjunior.me

 nassifrroma@gmail.com