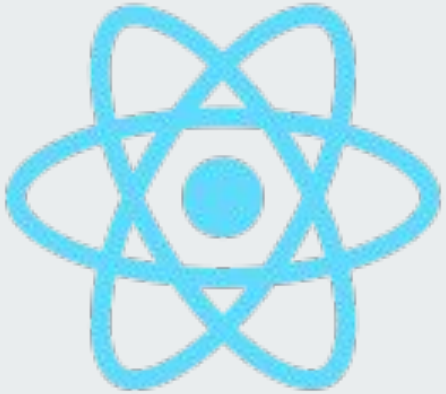


Minicurso: React

Gabriel C. Ullmann

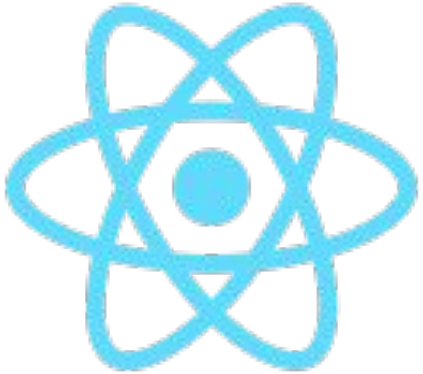
O que vamos fazer?

1. Aprender alguns conceitos básicos do React
2. Criar uma página estática com componentes React
3. Interagir com os componentes utilizando TypeScript

O que NÃO vamos fazer?

1. Trabalhar com HTML/CSS de forma aprofundada
2. Persistir dados ou integrar com bancos de dados
3. Fazer um tutorial de TypeScript
4. Fazer apps mobile

O que é React?



- Biblioteca JavaScript
- Interfaces de usuário (UI)
- Facebook (2011)
- Open-source (https://github.com/facebook/react)
- Aplicações do tipo SPA e mobile (React Native)

Alguns conceitos principais do React

- Declarativo
- Compatível
- Componentizado
- Nova forma de pensar (MVC, arquitetura)

Fonte: https://reactjs.org/docs/design-principles.html

Antes de começar...

Conceitos Importantes

1. Componente

- Elemento ou conjunto de elementos de interface
- Ex: botões, listas, títulos, botões, navbars, imagens
- É um módulo que pode ser reutilizado em várias páginas HTML
- Evita repetição de código

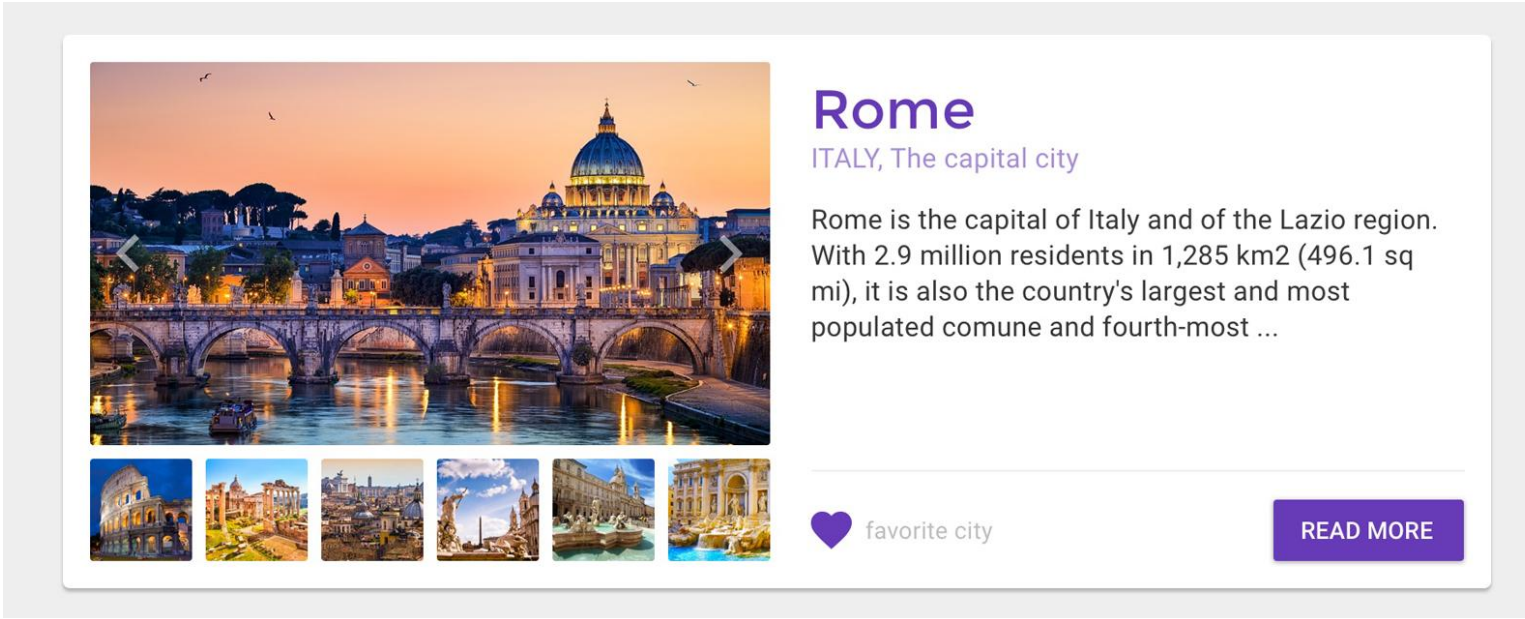


Figura 1 - Um componente do tipo “card” com imagens, textos e botões
https://www.uplabs.com/posts/horizontal-material-card

2. Render

- Renderizar = desenhar
- Função que cria e atualiza componentes
- Programador não manipula diretamente os elementos da página
- Manipulação do DOM - Document Object Model
- Não precisa usar: element.appendChild(element2);

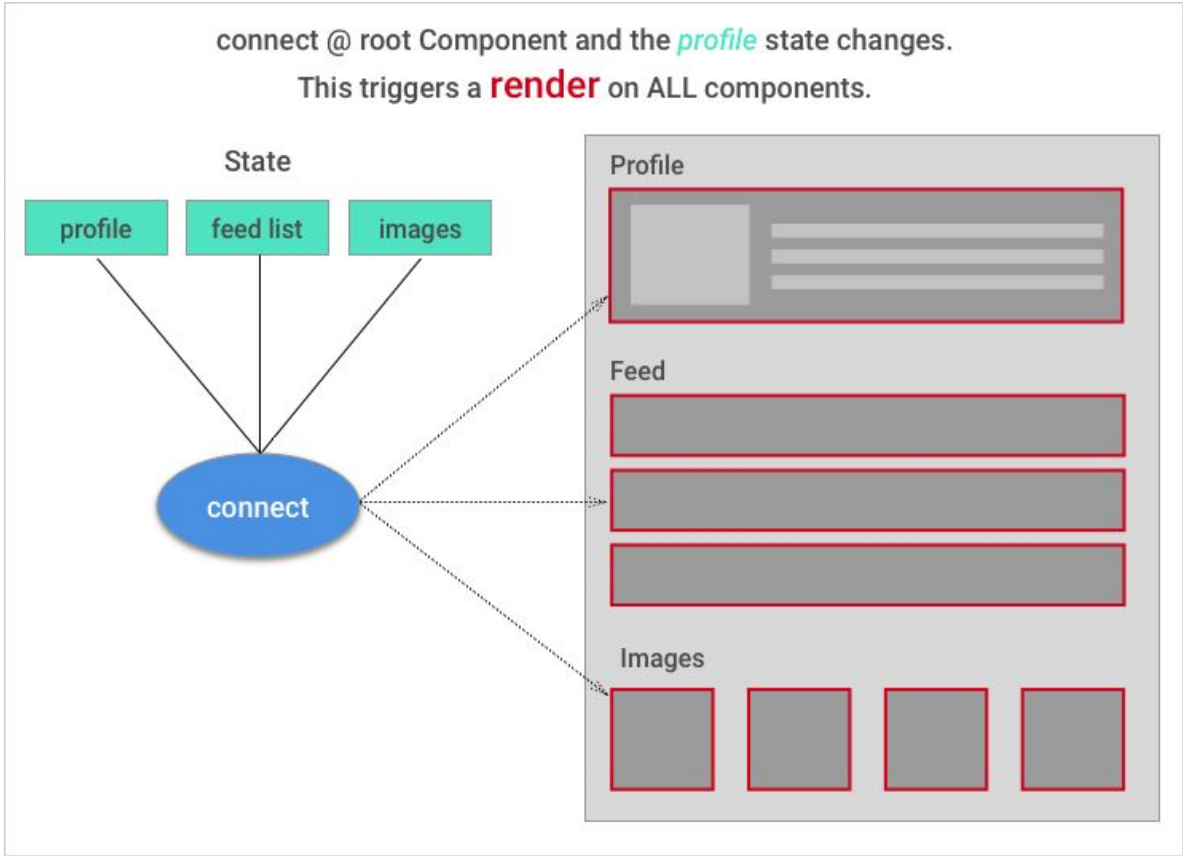


Figura 2 - Componentes e renderização
https://medium.com/dailyjs/quick-redux-tips-for-connecting-your-react-components-e08da72f5b3

3. JSX (JavaScript XML)

- Permite escrever e interpretar HTML diretamente dentro do JS
- É a forma mais comum de desenvolver com React
- Não é suportado nativamente (envolve configuração)
- Não precisa usar: document.createElement(“div”)

```jsx
function ComponentExample(props) {
  return <h1>{props.text}</h1>;
}

function App() {
  return (
    <div>
      <ComponentExample text="Let's see an example, shall we?" />
    </div>
  );
}

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

Figura 3 - Exemplo de código JSX
https://devdigression.com/2018/01/28/props-to-jsx/

4. TypeScript

- Superset do JavaScript
- Microsoft (2012)
- Open-source (https://github.com/Microsoft/TypeScript)
- Padrão ECMAScript 2015+

1. Primeiro componente

- **Title.js:** componente
- **index.js:** página principal
- Padrão: nomes de componentes com letra maiúscula
- Código dos componentes: src/components
- Estilização e imagens: src/assets

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

class Title extends Component {
 render() {
   return (
     <h1>Minicurso React</h1>
   );
 }
}

export default Title;
```

Title.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import Title from './components/Title';
import './assets/index.css';

//elemento raiz do render em public/index.html
ReactDOM.render(<Title />,
document.getElementById('root'));
```

index.js

2. Criando um card

- Criar arquivo **Card.js** em src/components
- Importar e utilizar juntamente com o **Title** em **index.js**
- Padrão: nomes de atributos JSX devem ser escritos em CamelCase

```
import React, { Component } from 'react';
import forest from '../assets/forest.jpg';

class Card extends Component {
    render() {
        return <div className="card">
        <img className="card-image" src={forest}/>
        <div className="card-title">Título</div>
        <div className="card-text">Texto</div>
    </div>
    }
}
export default Card;
```

Card.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import Title from './components/Title';
import Card from './components/Card';
import './assets/index.css';

//será que funciona com 2 componentes?
ReactDOM.render(<Title /> <Card />,
document.getElementById('root'));
```

index.js

```
//é preciso ter um elemento raiz!
//colocar tudo dentro de uma <div>
ReactDOM.render(<div>
   <Title />
   <Card />
</div>, document.getElementById('root'));
```

index.js

3. Criando um CardList

- Criar arquivo **CardList.js** em src/components
- Importar **Card** em **CardList** para utilizar os dois juntos

```
import React, { Component } from 'react';
import Card from './Card'

class CardList extends Component {
      render() {
            <div>Lista</div>
       }
}

export default CardList;
```

CardList.js

```
//crie uma função construtora dentro da classe
constructor(props) {
        super(props);
        this.state = {
            name: "Lista",
            cities: ["Santa Rosa", "Ijuí", "Horizontina",
"Três de Maio"]
        }
    }
```

CardList.js

Props vs. States

- **Props:** parâmetros que o componente-filho recebe de um componente-pai (através do **super**)
- **States:** parâmetros do próprio componente (“atributos” da classe)

```
//percorrer o array de cidades e criar 1 card por cidade
render() {
    var items = this.state.cities.map((city, index) => {
        return <Card key={index} title={city}></Card>
    });
    return items;
}
```

CardList.js

Função Array.map(fn)

Parâmetros: uma função qualquer (fn)

1. **Pega item** do array
2. **Executa a função de entrada** passando o item como parâmetro
3. **Retorna um array** com os itens **“processados”**

Neste caso, além do item utilizamos também o seu índice

```
//ATRIBUTOS
//key: obrigatório, controle interno feito pelo React
//cardnum: nosso controle
//title: nome da cidade, que será o título do card

<Card key={index} cardnum={index} title={city}></Card>

//Padrão: nomes de atributos personalizados NÃO devem ser
escritos em CamelCase
```

CardList.js

```
//criar um construtor para Card, que recebe as props
constructor(props) {
    super(props);
    this.state = {
        name: "Card"
    }
}
```

Card.js

```
//tornar título dinâmico e criar índice para o Card
<div className="card" cardnum={this.props.cardnum}>
    <img className="card-image" src={forest}/>
    <div className="card-title">{this.props.title}</div>
    <div className="card-text">Lorem ipsum...</div>
</div>
```

Card.js

4. Adicionando interação

- Criar função de que printa uma prop/state em **Card**
- Executar essa função ao clicar no **Card**

```
//criar função e fazer binding com "this"
constructor(props) {
    super(props);
    this.state = {
        name: "Card"
    }
    this.printName = this.printName.bind(this);
}

printName(e) {
    console.log("Nome: " + this.state.name);
    console.log("Número: " + this.props.cardnum);
}
```

Card.js

```
//a função será disparada ao clicar
render() {
    console.log("render");
    return <div className="card" cardnum={this.props.cardnum}
      onClick={fn => this.printName(fn)}>
       //resto do código...
    </div>
}
```

Card.js

5. Criando um accordion

- **Accordion:** componente que encolhe/expande
- Criar função de teste em **Card**
- Executar essa função ao clicar no **Card**

```
//função para expandir/encolher cards
//classes já estão definidas em assets/index.css
collapse(e) {
    var element = e.currentTarget;
    if (element.classList.contains("collapsed")) {
        element.classList.remove("collapsed");
    } else {
        element.classList.add("collapsed");
    }
}
```

Card.js

```
//incluir no construtor
this.collapse = this.collapse.bind(this);

//alterar no render conforme abaixo
/*antes*/ <div onClick={fn => this.printName(fn)} ...
/*depois*/ <div onClick={fn => this.collapse(fn)} ...
```

Card.js

6. Verificando avisos (warnings)

- Erros e avisos são mostrados pelo React no console
- Avisos muitas vezes contém dicas de padrões e melhores práticas

```
//Padrão: incluir alt em imagens
<div className="card" cardnum={this.props.cardnum}>
    <img className="card-image" src={forest} alt="Floresta"/>
    <div className="card-title">{this.props.title}</div>
    <div className="card-text">Lorem ipsum...</div>
</div>
```

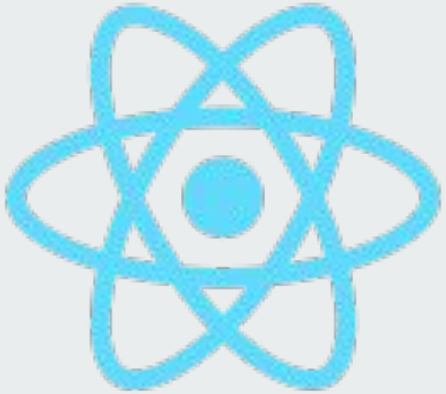
Card.js

Feito!

- Para saber mais:
  - Outro projeto: https://github.com/nkinesis/awesometask-client
  - Tutorial oficial do React: https://reactjs.org/tutorial/tutorial.html
  - Documentação do TypeScript: https://www.typescriptlang.org/docs/
  - React Native: https://facebook.github.io/react-native/

Para saber ainda mais...

- Integração com BD usando Sequelize: http://docs.sequelizejs.com/
- React com Redux (PT-BR): https://www.devmedia.com.br/react-redux/
- Webpack: https://webpack.js.org/
- Babel: https://babeljs.io/
- Node.js: https://nodejs.org/en/docs/



Obrigado!

http://bluend.xyz