React I

CAPÍTULO 1 – AMBIENTE

PROF. RAPHAEL GOMIDE

React I

CAPÍTULO 1 – AMBIENTE

PROF. RAPHAEL GOMIDE

Nesta aula

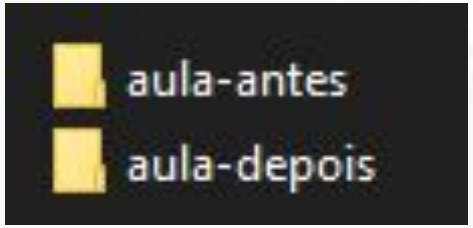
- Orientações sobre este Módulo.
- Considerações sobre o ambiente.
- Montagem do ambiente com Windows.

Orientações sobre este módulo

- **Estude o Módulo 0 caso ainda não se sinta confortável com HTML, CSS e/ou JavaScript!**
- Módulo com conteúdo 90% prático e 10% teórico.
- A apostila serve de **referência**. O conteúdo principal está nas **videoaulas**.
- Divisão das aulas em capítulos.
- Cada aula dura, em regra, entre 10 e 15 minutos.
- Formato das aulas, para cada capítulo:
  - Aula inicial – teoria.
  - Demais aulas – prática.
- Assista novamente à aula inicial após realizar a prática.

Orientações sobre este módulo

- Sugestão do professor, para cada aula:
  1. Assista à aula prática uma vez.
  2. Assista à mesma aula novamente, **codificando**.
- **Todo o código-fonte** das aulas será disponibilizado pelo professor no "Fórum de Avisos do Professor".
- Cada app a ser desenvolvido possui um roteiro para acompanhamento.
- Formato padrão dos arquivos de cada capítulo:



Considerações sobre o ambiente

- Será configurado um ambiente de desenvolvimento com o **Windows 10**.
- Em **Linux** e **MacOS**, é bem semelhante. Busque apoio nos fóruns, se necessário.
- Serão abordados os seguintes tópicos:
  - Instalação e teste do Node.js, npm e npx.
  - Instalação e teste do Yarn.
  - Instalação de algumas bibliotecas úteis, como o rimraf e o serve.
  - Instalação e configuração do VSCode Portable.
  - Criação de um projeto React com a biblioteca create-react-app.
  - Utilização de um projeto base, fornecido pelo professor.
    - Projeto “limpo” e com integração ao Tailwind CSS.
    - Sugestão de extensões.

Montagem do ambiente Windows

- Acompanhe o professor:
  - Aula prática.

Próxima aula

- Introdução ao React
- Criação do projeto **react-hello**.

React I

CAPÍTULO 2 – INTRODUÇÃO AO REACT

PROF. RAPHAEL GOMIDE

React I

CAPÍTULO 2 – INTRODUÇÃO AO REACT

PROF. RAPHAEL GOMIDE

Nesta aula

- Introdução ao React.
- Criação do projeto **react-hello**.

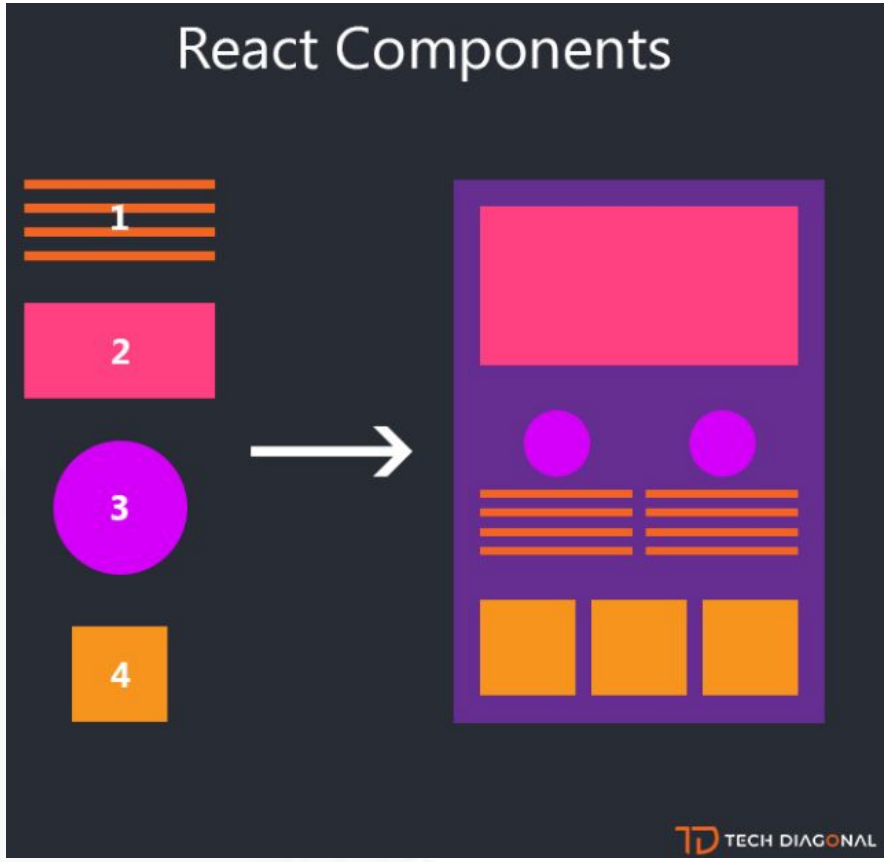
O que é o React?

- *“A JavaScript **library** for building user interfaces”.*

- Declarativo:
  - Componentes reativos com JSX.
  - **Mais foco** no **estado** do app e **regras de negócio**;
  - **Menos foco** em manipulação do **DOM manual**.
  - Manipulação do DOM performática (Virtual DOM).

- Baseado em componentes:
  - Alto grau de **reutilização de código**.

O que é o React?

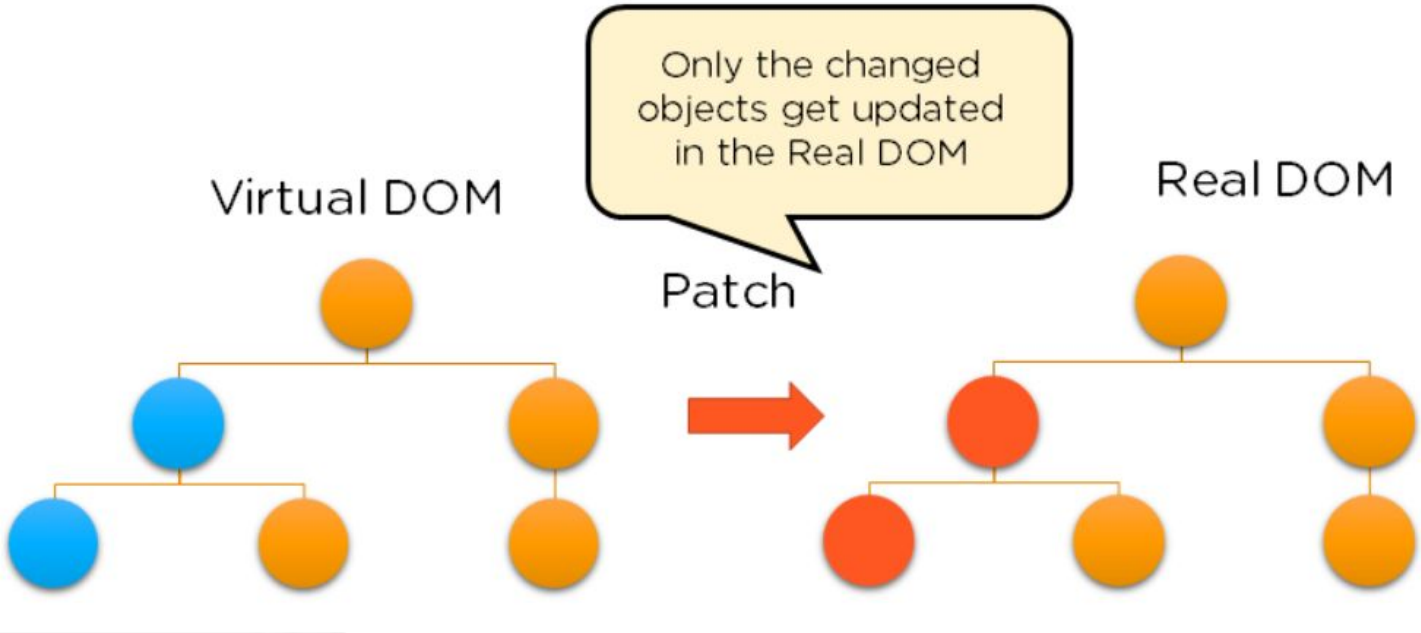


Fonte

Virtual DOM

- Manipulação performática do DOM.
- A manipulação manual do DOM é considerada uma operação **cara (lenta)**.
- O React só modifica o DOM nos locais que foram realmente alterados.
- Esse processo é mais conhecido como Reconciliation.

Virtual DOM



Fonte

Termos importantes

- Componente:
  - **Função** que agrupa determinado comportamento.
  - Em regra, realiza processamento e retorna dados renderizados (HTML + CSS).
  - Se bem escritos, podem ser reaproveitados.
  - Uma aplicação React é geralmente composta por diversos **componentes**.

- JSX:
  - JavaScript XML.
  - Utilizada pelo React para a escrita da renderização de **componentes**.

Termos importantes

- Sintaxe de importação e exportação:
  - Palavras-chave **import** e **export**.
  - Faz parte do JavaScript Moderno (ES6+).
  - Utilizadas para definirmos o que utilizar (**import**) e o que fornecer (**export**) durante a escrita do código-fonte.

- props:
  - **Propriedades** de componentes. Consideradas “somente leitura”.
  - Semelhante aos atributos de tags HTML.
  - Existe uma **prop** especial – *children*.
  - Utilizados para o **envio** e **recebimento** de dados entre **componentes**.

Termos importantes

- Closure:
  - **Função** implementada dentro do **escopo** de outra **função**.
  - Permite o **acesso** ao **escopo externo**.
  - Muito utilizado pelo React na criação de **componentes**, através da criação de funções para lidar com ocorrência de eventos (cliques, digitação, etc.).

- State:
  - Representa o **estado** de componentes.
  - **Estado**: dado que pode ser modificado com o tempo.
  - Exemplos: dados de formulários, Back End, etc.

Termos importantes

- Hooks:
  - Estrutura do React que pode ser vinculada (hooked) a componentes.
  - Internamente, são implementadas como **funções**.
  - Atuam diretamente na reatividade do React.
  - **Convenção**: começar com o prefixo “**use**”.
  - Principais hooks: **useState** e **useEffect**.
  - É possível criar nossos próprios hooks.

Termos importantes

- useState:
  - Hook para lidar com **estado**.
  - Devolve um valor e função modificadora (setter).
  - Sempre que o setter é invocado, uma nova renderização do componente é agendada e pode ocorrer a qualquer momento.
  - Isso é feito de forma assíncrona.

Termos importantes

- useEffect:
  - Hook para lidar com “efeitos colaterais” (**side effects**).
  - Palavra-chave: **sincronização**.
  - Estado do app x estado do mundo.
  - Permite, por exemplo:
    - Manipular o DOM manualmente.
    - Monitorar eventos do navegador.
    - Trabalhar com o Back End.

Termos no código

- Link no CodeSandBox.

```jsx
// Importação do hook useState
import { useEffect, useState } from "react";

/**
 * Declaração do componente
 * O export indica que ele
 * poderá ser utilizado externamente
 */
export default function App() {
  // CSS in JS
  const { containerStyle } = styles;

  // Exemplo de utilização de useEffect
  useEffect(() => {
    document.title = "react-counter";
  }, []);

  // JSX
  return (
    <div style={containerStyle}>
      <h1>react-counter</h1>

      {/* Utilização de outro componente */}
      <Counter />
    </div>
  );
}
```

Termos no código

- Link no CodeSandBox.

```
29  // Outro componente
30  function Counter() {
31    // Exemplo de utilização de useState
32    const [count, setCount] = useState(0);
33
34    // Closure
35    function handleDecrement() {
36      setCount((currentCount) => currentCount - 1);
37    }
38
39    // Closure
40    function handleIncrement() {
41      setCount(count + 1);
42    }
43
44    // JSX com props
45    return (
46      <div>
47        <button onClick={handleDecrement}>-</button>
48        <input type="text" value={count} readOnly />
49        <button onClick={handleIncrement}>+</button>
50      </div>
51    );
52  }
```

Prática

- Acompanhe o professor:
  - Criação do projeto **react-hello**.

Próxima aula

- Construção de componentes mais robustos.
- Criação do projeto **react-countries**.

React I

CAPÍTULO 3 – CONSTRUÇÃO DE COMPONENTES MAIS ROBUSTOS

PROF. RAPHAEL GOMIDE

React I

CAPÍTULO 3 – CONSTRUÇÃO DE COMPONENTES MAIS ROBUSTOS

PROF. RAPHAEL GOMIDE

Nesta aula

- Técnicas para a criação de componentes.
  - Renderização de arrays.
  - O problema do **prop drilling**.
  - Técnica ***Composition***.

- Criação do projeto **react-countries**.

Renderização de arrays

- Projeto de referência.
- Array:

```
const CARS = [
  {
    id: "c1",
    name: "Fiesta"
  },
  {
    id: "c2",
    name: "Versa"
  },
  {
    id: "c3",
    name: "Nivus"
  }
];
```

- **Má** prática – renderizar manualmente.

```
<ul>
  <li>{CARS[0].name}</li>
  <li>{CARS[1].name}</li>
  <li>{CARS[2].name}</li>
</ul>
```

Renderização de arrays

- Projeto de referência.
- Array:

```
const CARS = [
  {
    id: "c1",
    name: "Fiesta"
  },
  {
    id: "c2",
    name: "Versa"
  },
  {
    id: "c3",
    name: "Nivus"
  }
];
```

- **Boa** prática – renderizar com **array.map**.

```
<ul>
  {CARS.map((car) => {
    return <li key={car.id}>{car.name}</li>;
  })}
</ul>
```

Renderização de arrays

- Atenção à prop **key**.

```
<ul>
  {CARS.map((car) => {
    return <li key={car.id}>{car.name}</li>;
  })}
</ul>
```

- Ela “ajuda” o React a renderizar os dados mais corretamente.
- Recomenda-se a utilização de **identificadores únicos**.
- **Não** é recomendado que seja utilizado o **índice do array**.
- Caso a **key** não seja fornecida, o React emite o seguinte alerta no console do navegador:

```
Warning: Each child in a list should have a unique "key" prop.
```

O problema do *prop drilling*

- Em apps com muitos componentes, pode ocorrer este problema.
- Consiste na passagem de dados via props em *componentes intermediários.*
- Esses componentes intermediários não utilizam a prop, servem apenas para transportar a prop para componentes filhos.
- Uma das formas de se resolver esse problema é adotar uma técnica conhecida como **Composition**:
  - Os componentes *containers* recebem *filhos* através da prop *children*.
  - A declaração dos filhos ocorre no mesmo arquivo da declaração do componente pai, evitando assim o problema do *prop drilling.*

O problema do *prop drilling*

- Projeto de referência.
- App.js:

```
3   export default function App() {
4     const fatherName = "Phil Dunphy";
5     const motherName = "Claire Dunphy";
6     const sonName = "Luke Dunphy";
7     const daughterName = "Haley Dunphy";
8
9     return (
10      <div>
11        {/* Prop drilling com "son" */}
12        <Father name={fatherName} son={sonName} />
13
14        {/* Composition, evitando o prop drilling */}
15        <Mother name={motherName}>
16          <Daughter>{daughterName}</Daughter>
17        </Mother>
18      </div>
19    );
20  }
```

```
22  function Father({ name, son }) {
23    return (
24      <ul>
25        <li>Pai: {name}</li>
26        <li>
27          <Son son={son} />
28        </li>
29      </ul>
30    );
31  }
32
33  function Son({ son }) {
34    return <p>Filho: {son}</p>;
35  }
```

O problema do *prop drilling*

- Projeto de referência.
- App.js:

```
3   export default function App() {
4     const fatherName = "Phil Dunphy";
5     const motherName = "Claire Dunphy";
6     const sonName = "Luke Dunphy";
7     const daughterName = "Haley Dunphy";
8
9     return (
10      <div>
11        {/* Prop drilling com "son" */}
12        <Father name={fatherName} son={sonName} />
13
14        {/* Composition, evitando o prop drilling */}
15        <Mother name={motherName}>
16          <Daughter>{daughterName}</Daughter>
17        </Mother>
18      </div>
19    );
20  }
```

```
37  function Daughter({ children }) {
38    return <p>Filha: {children}</p>;
39  }
40
41  function Mother({ name, children }) {
42    return (
43      <ul>
44        <li>Mãe: {name}</li>
45        <li>{children}</li>
46      </ul>
47    );
48  }
```

Prática

- Acompanhe o professor:
  - Criação do projeto **react-countries**.

Próxima aula

- Projeto **react-flash-cards**.

React I

CAPÍTULO 4 – LIFTING STATE UP

PROF. RAPHAEL GOMIDE

React I

CAPÍTULO 4 – LIFTING STATE UP

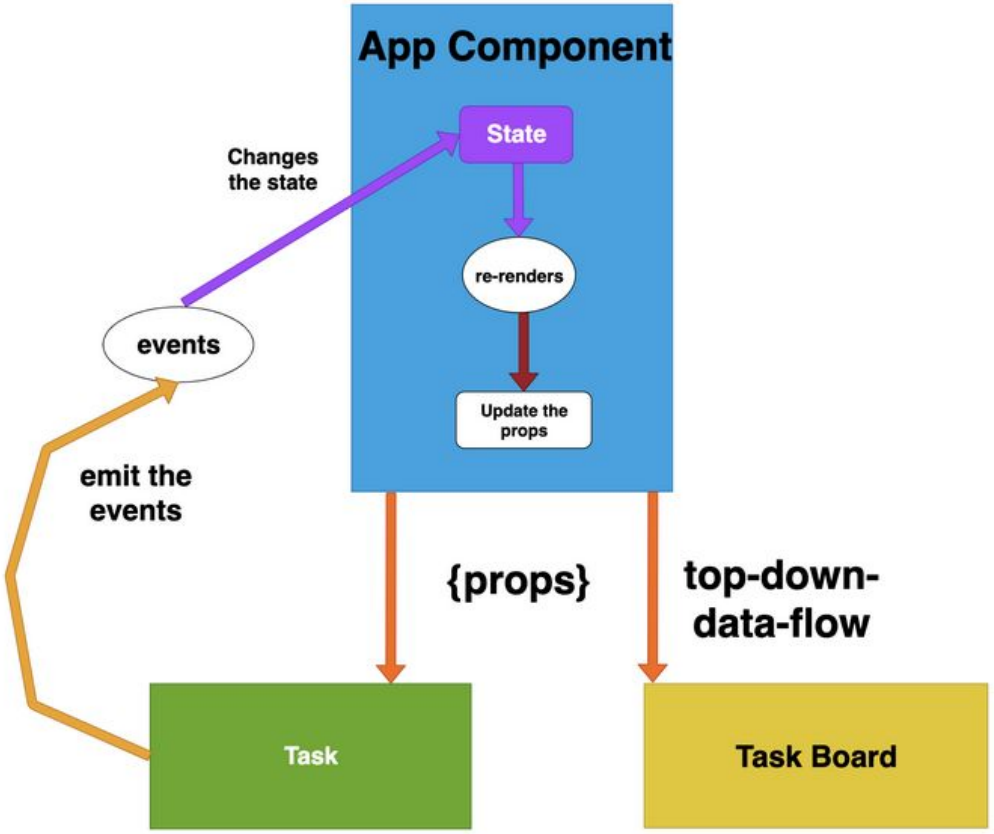
PROF. RAPHAEL GOMIDE

Nesta aula

- Apresentação da técnica “lifting state up”.
- Criação do projeto **react-flash-cards**.

Lifting state up

- Técnica que permite o compartilhamento de estado entre componentes “irmãos”, por exemplo.
- A técnica consiste em “subir” o estado para o componente “pai” comum a todos os que precisam do valor do estado.
- Mais informações aqui.



Fonte

Lifting state up

- Projeto de referência.
- Componente <Counter1 /> com estado local.

```
39  function Counter1() {
40    const [count, setCount] = useState(0);
41
42    function increment() {
43      setCount((currentCount) => currentCount + 1);
44    }
45
46    return (
47      <div>
48        <span>Contador "local" : {count}</span>
49        {"  "}
50        <button onClick={increment}>+</button>
51      </div>
52    );
53  }
```

- Componente <Counter2 />, que utiliza o estado de <App />

```
55  function Counter2({ value = 0, onIncrement = null }) {
56    function increment() {
57      if (onIncrement) {
58        onIncrement();
59      }
60    }
61
62    return (
63      <div>
64        <span>Contador "global": {value}</span>
65        {"  "}
66        <button onClick={increment}>+</button>
67      </div>
68    );
69  }
```

Projeto react-flash-cards

- Serão criadas 3 versões deste projeto.
- Versão 1:
  - Técnicas semelhantes às do projeto **react-countries**.
  - Utilização de “lifting state up”.
- Versão 2:
  - Utilização da Versão 1 como base.
  - Inclusão de **integração** com o **Back End**.
  - Inclusão de CRUD (Create Retrieve Update Delete).
- Versão 3:
  - Utilização da Versão 2 como base.
  - Inclusão de ***deploy*** nos serviços Glitch (**Back End**) e Netlify (**Front End**).

Prática

- Acompanhe o professor:
  - Criação do projeto **react-flash-cards-v1**.

Próxima aula

- Projeto **react-flash-cards-v2**.

React I

CAPÍTULO 5 – INTEGRAÇÃO COM O BACK END

PROF. RAPHAEL GOMIDE

React I

CAPÍTULO 5 – INTEGRAÇÃO COM O BACK END

PROF. RAPHAEL GOMIDE

Nesta aula

- Integração do React com o Back End.
- Criação do projeto **react-flash-cards-v2**.

Integração do React com o Back End

- Em regra, um Back End consiste em uma API que recebe e devolve dados no formato JSON.
- O sistema que trata esses dados pode ser construído com diversas tecnologias (Java, .NET, Node.js, PHP, Python, etc.)
- O Back End pode ser considerado como “estado do mundo” (*state of the world).*
- Sendo assim, devemos sincronizar os dados com useEffect.
- É interessante também dar um *feedback* visual ao usuário enquanto os dados estão sendo carregados.

Integração do React com o Back End

- Projeto de referência.
- Em **useEffect**, há uma função para a obtenção dos dados com o comando **fetch**.
- O App inicia sem dados e com **loading = true**
- Após obter os dados, definimos os mesmos e **loading = false**.
- O valor de **loading** é utilizado para definir o que é renderizado.

```
const [loading, setLoading] = useState(true);
const [todos, setTodos] = useState([]);

useEffect(() => {
  async function getTodos() {
    const resource = await fetch(
      "https://jsonplaceholder.typicode.com/todos"
    );
    const json = await resource.json();
    setTodos(json);

    // Forçando a visualização do texto "Carregando..."
    setTimeout(() => {
      setLoading(false);
    }, 500);
  }

  getTodos();
});

if (loading) {
  return <p>Carregando...</p>;
}

return (
  <div>
    <h1>Todo's</h1>
    // restante do código
```

Prática

- Acompanhe o professor:
  - Criação do projeto **react-flash-cards-v2**.

Próxima aula

- Projeto **react-flash-cards-v3**.

React I

CAPÍTULO 6 – DEPLOY DE APPS

PROF. RAPHAEL GOMIDE

React I

CAPÍTULO 6 – DEPLOY DE APPS

PROF. RAPHAEL GOMIDE

Nesta aula

- Introdução ao Glitch.
- Introdução ao Netlify.
- Criação do projeto **react-flash-cards-v3**.

Introdução ao Glitch

- Glitch.
- Oferece hospedagem gratuita (limitada) para Back End.
- Não exige cartão de crédito.
- Permite o upload de arquivos.

Introdução ao Netlify

- Netlify.
- Oferece hospedagem gratuita (limitada) para Front End.
- Não exige cartão de crédito.
- Possui um CLI (Command Line Interface) compatível com Node.js.

Prática

- Acompanhe o professor:
  - Criação do projeto **react-flash-cards-v3**.
  - Hospedagem do Back End no Glitch.
  - Hospedagem do Front End na Netlify.