Aula 08 - Oi Devs

1. Sugestões de temas (30, 15, 10)
   a. Curiosidade do frontend
   b. Mulher na tecnologia
   c. Cultura pop
   d. Ecossistema OI

Aula 07 - Oi Devs

- [x] ~~Stack: React, JavaScript (Componentes Funcionais com ES6)~~
- [x] ~~CRA~~
- [x] ~~Modelagem baseada em componentes~~
- [x] ~~JSX (doc oficial)~~
- [x] ~~ES6~~
- [x] ~~React Router~~
- [x] ~~Props~~
- [x] ~~Arrow Function~~
- [x] ~~Proptypes~~
- [x] ~~React State~~
- [x] ~~State Lifting~~
- [x] ~~Renderização Condicional~~
- [x] ~~bind() (class components)~~
- [x] ~~Renderização de Listas~~
- [x] ~~Função de Callback~~

Eventos

Bem como os eventos HTML da DOM, React pode realizar ações baseadas em eventos do usuário
React tem os mesmos eventos do HTML: click, change, mouseover, etc.

Os manipuladores de eventos determinam qual ação deve ser executada sempre que um evento é acionado.
Isso pode ser um clique num botão ou uma alteração no input de um texto.

Essencialmente, os manipuladores de eventos fazem com que os usuários interajam a aplicação React.

O tratamento de eventos com elementos React é semelhante ao tratamento de eventos em elementos DOM.

onFocus

É chamado quando o usuário clica em um input de texto.

```
function Example() {
  return (
    <input
      onFocus={(e) => {
        console.log('Focos no input');
      }}
      placeholder="onFocus é acionado quando você clica nesta entrada."
    />
  )
}
```

onBlur

É chamado quando o usuário clica fora de um input de texto focado.

```
function Example() {
  return (
    <input
      onBlur={(e) => {
```

```
        console.log('Disparado porque esta entrada perdeu o foco');
      }}
      placeholder="onBlur é acionado quando você clica nesta entrada e clica fora dela."
    />
  )
}
```

onChange

```
import React from 'react';

function App() {

  function handleChange(event) {
    console.log(event.target.value);
  }

  return (
    <input name="firstName" onChange={handleChange} />
  );
}

export default App;
```

onClick

O manipulador de eventos **onClick** do React nos permite chamar uma função quando um usuário clica em um elemento, como um botão, por exemplo.

Os nomes dos eventos devem ser escritos em **camelCase,** então o evento **onclick** deve ser escrito como **onClick** em uma aplicação React.

deve ser escrito como **onClick** em uma aplicação React.

Além disso, os manipuladores de eventos do React aparecem dentro de **chaves {}**.

Um exemplo de um evento escrito em HTML:

```
<button onclick="sayHello()">
  Say Hello
<button>
```

Em React:

```
<button onClick={sayHello}>
  Say Hello
<button>
```

Outra diferença chave é que nós devemos chamar explicitamente a função **preventDefault**
no React, enquanto que em HTML, simplesmente retornaria **false** para evitar que um link abra em uma nova página "acidentalmente".

O exemplo seguinte mostra como evitar que um link abra uma nova página por padrão:

```
<a href="#" onclick="console.log('The link was clicked.'); return false">
  Click me
</a>
```

Esse mesmo código em React seria:

```
function ActionLink() {
  const handleClick = (e) => {
    e.preventDefault();
    console.log('The link was clicked.');
  }

```

```
  return (
    <button onClick={handleClick}>
      Click me
    </button>
  );
}
```

Já no caso abaixo, quando o preventDefault é chamado quando estamos submetendo o formulário ele previne que o browser faça o reload/refresh.

Vamos adicionar uma permissão aos usuários: **admin** e **user**.
Vamos criar uma nova rota quando o usuário for **admin** que vá para uma nova página que exiba uma lista.
Vamos tentar com e sem o **preventDefault**.

Formulários

```
import React from 'react';

const INITIAL_LIST = [
  'Learn React',
  'Learn Firebase',
  'Learn GraphQL',
];

const ListWithAddItem = () => {
  const [value, setValue] = React.useState('');
  const [list, setList] = React.useState(INITIAL_LIST);

  const handleChange = (event) => {
    setValue(event.target.value);
  };

```

```
  const handleSubmit = (event) => {
    if (value) {
      setList(list.concat(value));
    }

    setValue('');

    event.preventDefault();
  };

  return (
    <div>
      <ul>
        {list.map(item => (
          <li key={item}>{item}</li>
        ))}
      </ul>

      <form onSubmit={handleSubmit}>
        <input type="text" value={value} onChange={handleChang
e} />
        <button type="submit">Add Item</button>
      </form>
    </div>
  );
};

export default ListWithAddItem;
```

```jsx
import React from 'react';

const INITIAL_LIST = [
  'Learn React',
  'Learn Firebase',
  'Learn GraphQL',
];

const ListWithAddItem = () => {
  const [value, setValue] = React.useState('');
  const [list, setList] = React.useState(INITIAL_LIST);

  const handleChange = (event) => {
    setValue(event.target.value);
  };

  const handleSubmit = (event) => {
    if (value) {
      setList(list.concat(value));
    }

    setValue('');

    event.preventDefault();
  };

  return (
    <div>
      <ul>
        {list.map(item => (
          <li key={item}>{item}</li>
        ))}
      </ul>

      <form onSubmit={handleSubmit}>
        <input type="text" value={value} onChange={handleChange} />
        <button type="submit">Add Item</button>
      </form>
    </div>
  );
};

export default ListWithAddItem;
```

```jsx
import React from "react";
import { useState } from "react";
function ChoreForm({ addChoreLog }) {
  const [choreDesc, setChoreDesc] = useState();
```

```
  const [name, setName] = useState();
  const [date, setDate] = useState();
  const handleSubmit = (e) => {
    addChoreLog([choreDesc, name, date]);
    e.preventDefault();
  };
  return (
    <form
      onSubmit={(e) => {
        handleSubmit(e);
      }}
    >
      <label>Chore description:</label>
      <br />
      <input
        name="choreDesc"
        type="text"
        value={choreDesc}
        onChange={(e) => setChoreDesc(e.target.value)}
      />
      <br />
      <label>Name:</label>
      <br />
      <input
        name="name"
        type="text"
        value={name}
        onChange={(e) => setName(e.target.value)}
      />
      <br />
      <label>Date:</label>
```

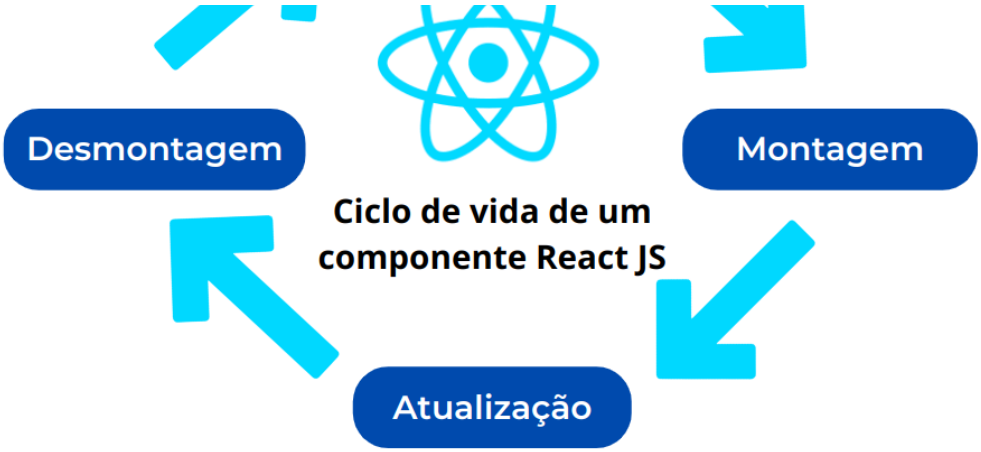
```
      <br />
      <input
        name="date"
        type="date"
        value={date}
        onChange={(e) => setDate(e.target.value)}
      />
      <br />
      <input type="submit" value="Add Log" />
    </form>
  );
}
export default ChoreForm;
```

```
          value={choreDesc}
          onChange={e => setChoreDesc(e.target.value)}
        />
        <br/>
        <label>Name:</label>
        <br />
        <input
          name='name'
          type='text'
          value={name}
          onChange={e => setName(e.target.value)}
        />
        <br />
        <label>Date:</label>
        <br />
        <input
          name='date'
          type='date'
          value={date}
          onChange={e => setDate(e.target.value)}
        />
        <br/>
        <input
          type='submit'
          value='Add Log'
        />
      </form>
    )
}

export default ChoreForm;
```

Métodos do Ciclo de Vida

Descrevendo um componente da forma mais básica possível, podemos dizer que é um pedaço de código que é renderizado para o usuário, que pode ser atualizado caso haja alguma interação, e que pode deixar de ser renderizado.



Ciclo de vida de um componente

O useEffect é um Hook que permite executar efeitos colaterais em seus componentes.
Alguns desses efeitos colaterais podem ser: busca de dados em uma API, atualizar diretamente o VirtualDOM e controlar temporizadores (setTimeout e setInterval).

O useEffect é utilizado para fazer o ciclo de vida de um componente em substituição às funções componentWillMount, componentDidMount, componentDidUpdate e componentWillUnmount.

As funções de ciclo de vida (componentWillMount, componentDidMount, componentDidUpdate e componentWillUnmount) não são mais utilizadas desde a atualização do React para a versão 16.8.

Sintaxe

```
useEffect(<função>, <array de dependência>)
```

Note que o useEffect recebe como primeiro parâmetro uma função que será executada assim que o componente for renderizado.

O segundo parâmetro recebido pelo useEffect será um array de dependência, podendo ser utilizado de forma opcional.

Inicialização

Esta é a primeira fase de um componente quando uma página é acessada, em que a aplicação insere seu State.
Esse dado já fica acessível durante a renderização da página.

```
import { useState } from "react";

export default function App() {
  const [contador, setContador] = useState(0);

  return (
    <div className="App">
      <h1>Contador: {contador}</h1>
      <button onClick={() => setContador(contador + 1)}>+</button>
    </div>
  );
}
```

Já temos o valor do state **contador** instanciado.

Montagem

Esta é a primeira fase do ciclo de vida de um componente. Aqui veremos que o useEffect é executado uma única vez quando o componente é montado, após a primeira renderização.
Esse recurso é normalmente utilizado para fazer requisições na API quando a página é recém carregada.

```
import { useState, useEffect } from "react";

export default function App() {
  const [usuarios, setUsuarios] = useState([{}]);



  useEffect(() => {
    // executa um bloco JavaScript após a renderização do comp
```

```
    // executa um bloco JavaScript após a renderização do componente
    console.log("executei")
  }, []);

  return (
    <div className="App">
      <h1>Contador: {contador}</h1>
      <button onClick={() => setContador(contador + 1)}>+</button>
    </div>
  );
}
```

O useEffect será executado após a primeira renderização do componente.

Atualização

Esta é a terceira fase pela qual passa nosso componente. Após a fase de montagem onde o componente foi criado, a fase de atualização entra em cena. É aqui que quando o estado do componente muda ocorre a re-renderização.

```
import { useState, useEffect } from "react";

export default function App() {
  const [contador, setContador] = useState(0);

  useEffect(() => {
    console.log("executará o useEffect toda a vez que o [contador] mudar");
  }, [contador]);

  return (
    <div className="App">
```

```
      <h1>Contador: {contador}</h1>
      <button onClick={() => setContador(contador + 1)}>+</button>
    </div>
  );
}
```

A sintaxe aqui é similar da montagem, porém agora ao invés de um array vazio, colocaremos um array de dependência (contador) para que além da montagem do componente, a cada atualização do valor da dependência o useEffect chamará a função novamente e re-renderizá-las.

Desmontagem

Esta é a última fase do ciclo de vida de um componente e é responsável por fazer a limpeza do Virtual DOM. Quando precisamos desmontar um componente é ela que utilizaremos.

```
import { useState, useEffect } from "react";

export default function App() {
  const [contador, setContador] = useState(0);

  useEffect(() => {
    return () => console.log("Aqui é quando o componente será desmontado!");
  }, []);

  return (
    <div className="App">
      <h1>Contador: {contador}</h1>
      <button onClick={() => setContador(contador + 1)}>+</button>
```

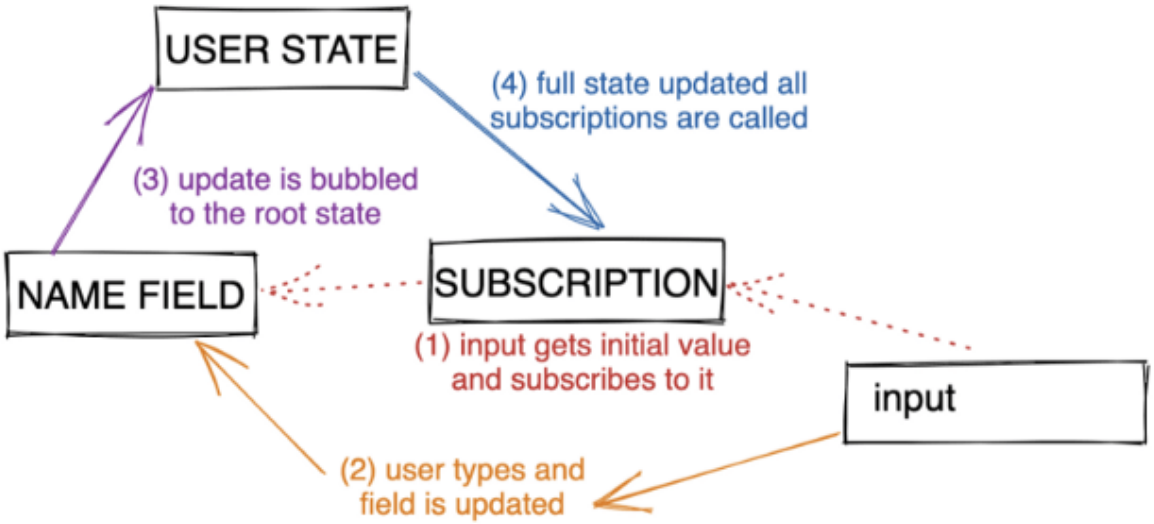
```
    </div>
  );
}
```

Quando passamos uma função para o return do useEffect, ele executa a função quando desmontar o componente.

Bindings

```
const UserName = ({ name, onChange }) => {
  return <input onChange={onChange} value={name} />;
};

const App = () => {
  const [user, setUser] = useState({ name: "" });
  return <UserName
    name={name}
    onChange={(e) => setUser({ name: e.target.value})}
  />;
};
```



Composição de Componentes (children)

Alguns componentes não tem como saber quem serão seus elementos filhos. Isso é muito comum para componentes como o `SideBar` ou `Dialog` que representam "caixas" genéricas.

Recomendamos que esses componentes utilizem a prop especial `children` para passar os elementos filhos diretos para sua respectiva saída:

```
function ComponenteFilho(props) {
  return (
    <div style={{color: props.color}}>
      {props.children}
    </div>
  );
}
```

```
function ComponentePai() {
  return (
    <ComponenteFilho color="blue">

      <h1 className="Dialog-title">
        Bem-vindo
      </h1>
      <p className="Dialog-message">
        Obrigado por visitar a nossa espaçonave!
      </p>

    </ComponenteFilho>
  );
}
```