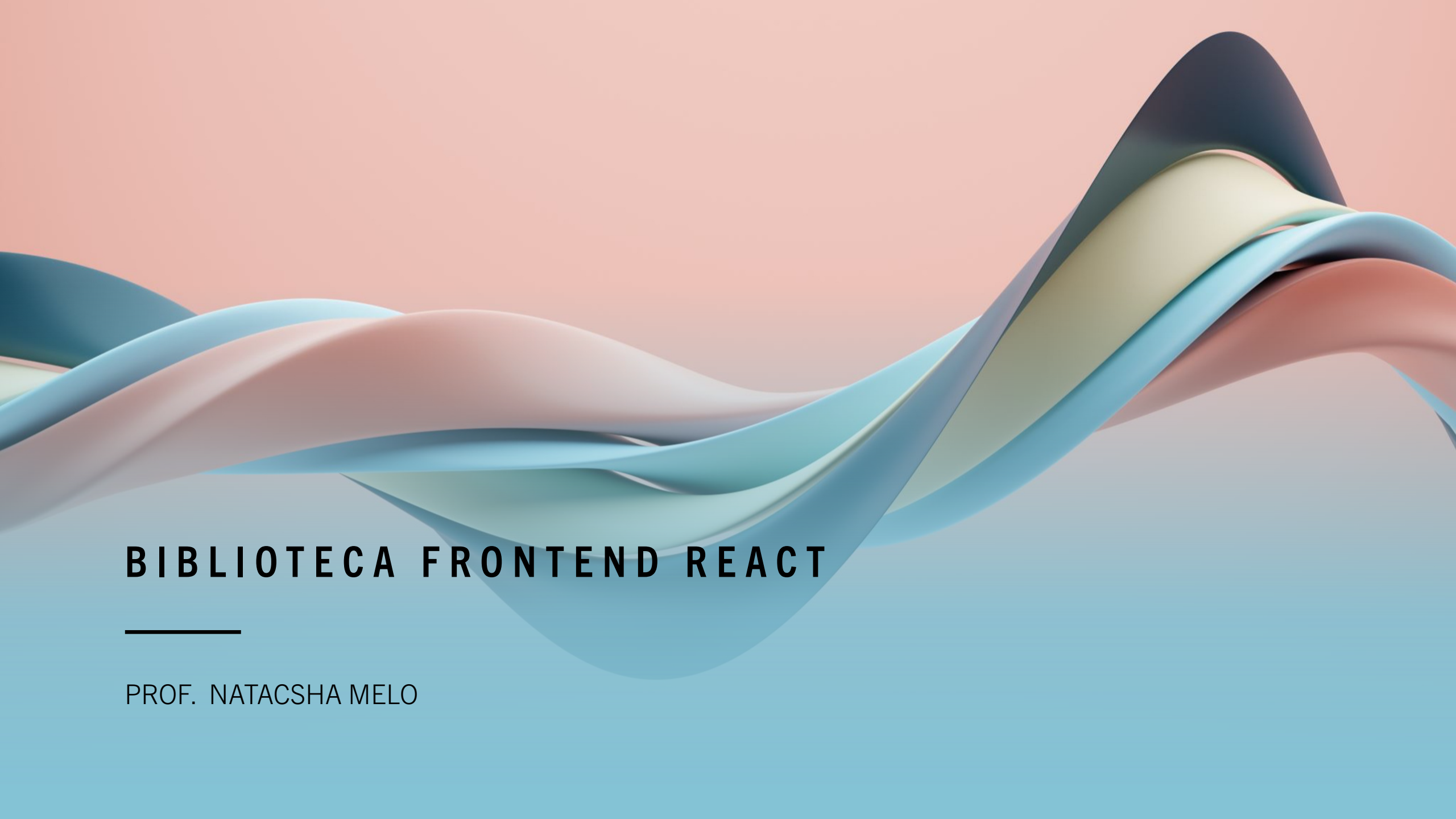

BIBLIOTECA FRONTEND REACT

PROF. NATACSHA MELO

PROPRIEDADES

- Também é possível passar dados para os componentes através das **propriedades**, de uma forma similar à linguagem HTML
- Props são escritas como atributos dentro das invocações de componentes e são passadas para dentro de componentes
- Primeiro, vamos remover todos os dados do nosso TableBody componente no src/Table.js.

```
const TableBody = () => {
    return <tbody />
}
```

- Vamos mover todos esses dados para uma matriz de objetos
- Como se estivéssemos usando uma API baseada em JSON
- Crie um array dentro do render()

PROPRIEDADES

- O arquivo src/App.js fica assim

```
class App extends Component {
    render() {
        const characters = [
            {
                name: 'Nat',
                email: 'nat@mail.com',
            },
            {
                name: 'João',
                email: 'joao@mail.com',
            },
            {
                name: 'José',
                email: 'jose@mail.com',
            },
            {
                name: 'Anna',
                email: 'anna@mail.com',
            },
        ]

        return (
            <div className="container">
                <Table />
            </div>
        )
    }
}
```

PROPRIEDADES

- Vamos passar os dados para o componente filho (Table) com propriedades
- Similar ao uso de data- atributes
- Podemos chamar a propriedade de qualquer nome. (*Não pode ser palavra reservada!*)
- Utilizaremos o characterData para enviar os dados
- O arquivo src/App.js fica assim:

```
return (
    <div className="container">
        <Table characterData={characters} />
    </div>
)
```

PROPRIEDADES

- Vamos passar os dados para o componente filho (Table) com propriedades
- Similar ao uso de data- atributes
- Podemos chamar a propriedade de qualquer nome. (*Não pode ser palavra reservada!*)
- Utilizaremos o characterData para enviar os dados
- O arquivo src/App.js fica assim:

```
return (
    <div className="container">
        <Table characterData={characters} />
    </div>
)
```

Utiliza-se chaves, pois é uma expressão JavaScript

PROPRIEDADES

- Vamos acessar os dados dentro do componente no arquivo src/Table.js

```
class Table extends Component {
    render() {
        const { characterData } = this.props

        return (
            <table>
                <TableHeader />
                <TableBody characterData={characterData} />
            </table>
        )
    }
}
```

PROPRIEDADES

- Agora os dados estão no DOM Virtual
- Para acessar no DOM Real vamos utilizar o this.props
- Utilizaremos o this.props.characterData
- Vamos passar os dados para o Tablebody por props
- Utilizaremos o map para retornar uma linha da tabela para cada objeto no array
- No map está contido as rows

```
const TableBody = (props) => {
     const rows = props.characterData.map((row, index) => {
          return (
               <tr key={index}>
               <td>{row.name}</td>
               <td>{row.email}</td>
               </tr>
          )
     })

return <tbody>{rows}</tbody>

}
```

PROPRIEDADES

- Sempre que for usar listas no React utilize Keys
- Props são uma maneira eficaz de passar dados existentes para um componente React
- O componente não pode alterar as props (são somente leitura)
- Para manipular os dados utilizados o Estado

ESTADO

- No nosso exemplo da Tabela estamos armazenando os dados em uma matriz e passando os dados por uma props
- Com props temos um fluxo de dados unidirecional
- Com Estado podemos atualizar dados privados de um componente
- Vamos criar um objeto state no src/App.js

```
class App extends Component {
    state = {}
}
```

ESTADO

O objeto conterá propriedades para tudo que deseja armazenar no estado

Utilizaremos characters

```
class App extends Component {
    state = {
        characters: [],
    }
}
```

ESTADO

Vamos mover toda a matriz de objetos que criamos para o state.characters no src/App.js

Pronto! Nossos dados estão oficialmente contidos no estado.

```
class App extends Component {
    state = {
        characters: [
            {
                name: 'Nat',
                email: 'nat@mail.com',
            },
            {
                name: 'João',
                email: 'joao@mail.com',
            },
            {
                name: 'José',
                email: 'jose@mail.com',
            },
            {
                name: 'Anna',
                email: 'anna@mail.com',
            },
        ],
    }
}
```

ESTADO

- Vamos criar o método removeCharacter para remover um item da tabela na classe App
- Para recuperar o estado usaremos this.state.characters
- Para atualizar o estado usaremos this.setState()
- Filtraremos o array com base em um index que passamos e retornaremos o novo array.

```
removeCharacter = (index) => {
    const { characters } = this.state

    this.setState({
        characters: characters.filter((character, i) =>
        {
            return i !== index
        }),
    })
}
```

ESTADO

- Vamos criar o método **removeCharacter** para remover um item da tabela na classe **App**
- Para recuperar o estado usaremos **this.state.characters**
- Para atualizar o estado usaremos **this.setState()**
- Filtraremos o **array** com base em um **index** que passamos e retornaremos o novo array.

```
removeCharacter = (index) => {
    const { characters } = this.state

    this.setState({
        characters: characters.filter((character, i) =>
        {
            return i !== index
        }),
    })
}
```

Você deve usar **this.setState()** para modificar uma matriz de dados. Aplicar um novo valor a **this.state.property** não funcionará.

ESTADO

- Vamos renderizar o botão em cada linha para chamar a função
- Passaremos a função removeCharacter como uma prop para Table no arquivo src/App.js

```
render() {
    const { characters } = this.state

    return (
        <div className="container">
        <Table characterData={characters} removeCharacter={this.removeCharacter} />
        </div>
    )
}
```

ESTADO

- Assim como passamos a TableBody para a Table faremos com o removeCharacter
- Substitua a Classe Table pelo código abaixo

```
const Table = (props) => {
     const { characterData, removeCharacter } = props

     return (
          <table>
          <TableHeader />
          <TableBody characterData={characterData} removeCharacter={removeCharacter} />
          </table>
     )
}
```

MANIPULANDO EVENTOS

- Vamos utilizar o índice que definimos no método removeCharacter()
- No componente TableBody passaremos o índice
- Vamos criar um botão com um onClick para passá-lo

```
<tr key={index}>
    <td>{row.name}</td>
    <td>{row.email}</td>
    <td>
        <button onClick={() => props.removeCharacter(index)}>Delete</button>
    </td>
</tr>
```