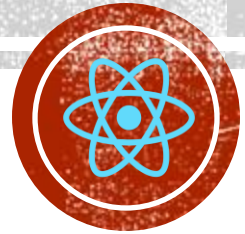
FORMULÁRIOS, VALIDAÇÃO E DATA-BIDING



Douglas Nassif Roma Junior

/douglasjunior

/in/douglasjunior

nassifrroma@gmail.com

AGENDA

Tópico	Conteúdo
Formulários, validação e data-binding	- Criando componentes reusáveis - Validação de formulários - Tabelas e busca - Requests com axios

CHAMADAS PARA APIS

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, o React 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 (aaaahhhhhhh nãããooooooo)

CHAMADAS PARA APIS

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

```
$ npm install axios
```

- Vamos criar um componente `Tasks` para requisitar uma lista de tarefas.

- A requisição deve ser feita com o `axios`, e o resultado armazenado no `state` do componente.

- O `state` contendo a lista de tarefas deve ser renderizado na tela em uma `<table />`.

CHAMADAS PARA APIS

pages/Tasks.jsx

```
import axios from 'axios';

// ...
    const [tasks, setTasks] = useState([]);

    useEffect(() => {
        axios.get('https://jsonplaceholder.typicode.com/todos')
            .then(response => {
                const { data } = response;
                setTasks(data);
            })
            .catch(err => {
                console.warn(err)
            })
    }, [])
// ...
```

RENDERIZANDO TABELAS

- Uma vez que a lista de tarefas foi carregada no `state`, podemos aproveitar dos recursos do **JavaScript** para renderizar uma **tabela legível** ao usuário.

pages/Tasks.jsx

```
// ...
    const renderTask = (task) => {
        return (
            <tr key={task.id}>
                <td>{task.id}</td>
                <td>{task.title}</td>
                <td>{task.completed ? '✓' : '✗'}</td>
            </tr>
        )
    }
// ...
```

```
// ...
    <table border="1">
        <thead>
            <tr>
                <th>ID</th>
                <th>Título</th>
                <th>Concluído</th>
            </tr>
        </thead>
        <tbody>
            {tasks.map(renderTask)}
        </tbody>
    </table>
// ...
```

FORMULÁRIOS E DATA-BINDING

FORMULÁRIOS E DATA-BINDING

- Outro recurso importante para uma aplicação Web é a criação de formulários.

- O **React** originalmente não possui nenhuma forma automática para a implementação do famoso "**two-way data binding**".

- Por isso, o processo consiste em capturar os eventos de "entrada de dados" dos componentes e então armazenar os valores desejados no `state`.

FORMULÁRIOS E DATA-BINDING

- Implementamos a função que irá lidar com o evento disparado pelo `input`, bem como a função que irá filtrar o `array` de tarefas.

pages/Tasks.jsx

```
// ...
  const [search, setSearch] = useState('');

  const handleSearch = (event) => {
    setSearch(event.target.value)
  }

  const filteredTasks = useMemo(() => {
    if (!search) return tasks;
    return tasks.filter(task => task.title.includes(search))
  }, [search, tasks]);
// ...
```

FORMULÁRIOS E DATA-BINDING

- E então, vamos criar um `input` de texto para filtrar as *Todos* carregadas no exemplo anterior.

pages/Tasks.jsx

```
import { Col, Form, Input, Row } from 'antd';

// ...
  <Row gutter={[24, 24]}>
    <Col span="23">
      <Form layout="vertical">
        <Form.Item label="Busca de tarefas">
          <Input placeholder="Buscar ..." value={search} onChange={handleSearch} />
        </Form.Item>
      </Form>
    </Col>
  </Row>
// ...
```

Continua...

COMPONENTES REUSÁVEIS

- Dentre as vantagens proporcionadas pelos componentes, a possibilidade de reaproveitamento de código está entre as principais.
- Utilizando o exemplo anterior, podemos criar componentes reusáveis para as linhas da tabela e também para o `input`.

components/TaskItem.jsx

```
const TaskItem = (props) => {
  const { task } = props;
  return (
    <tr>
      <td>{task.id}</td>
      <td>{task.title}</td>
      <td>
        {task.completed ? '✓' : '✗'}
      </td>
    </tr>
  );
};

export default TaskItem;
```

components/InputText.jsx

```
import { Form, Input } from 'antd';

const InputText = (props) => {
  const { label, ...others } = props;
  return (
    <Form.Item label={label}>
      <Input {...others} />
    </Form.Item>
  );
}

export default InputText;
```


Bônus: https://git.io/fjbAt

COMPONENTES REUSÁVEIS

- Atualizando a página de tarefas:

pages/Tasks.jsx

```
// ...

  const renderTask = (task) => {
    return (
      <TaskItem task={task} key={task.id} />
    )
  }

// ...
```

pages/Tasks.jsx

```
// ...

          <Form layout="vertical">
            <InputText
              label="Busca de tarefas"
              placeholder="Buscar por título"
              onChange={handleSearch}
              value={search}
            />
          </Form>

// ...
```


Bônus: https://git.io/fjbAt

VALIDAÇÃO

VALIDAÇÃO

- Assim como o "**two-way data binding**", o **React** não possui recursos automatizados para validação de campos e formulários.

- Sendo assim, é preciso implementar seu próprio componente de validação, ou utilizar alguma biblioteca especializada pra isso (Final-Form, Formik, Hook-Form).

- Continuando o exemplo anterior, vamos adicionar uma validação para o `input` de filtro, de modo que exista uma **limitação de 10 caracteres para o termo de busca**.

VALIDAÇÃO

- Primeiro precisamos adicionar ao componente `InputForm` os poderes para lidar com validações.

components/InputText.jsx

```
import { Form, Input } from 'antd';
import { useState } from 'react';

const InputText = (props) => {
  const { label, onChange, validate, ...others } = props;
  const [errorMessage, setErrorMessage] = useState(null);
  const [changed, setChanged] = useState(null);
  const validateStatus = errorMessage ? 'error' : 'success';
  const handleValidation = (event) => {
    setChanged(true);
    if (validate) {
      setErrorMessage(validate(event.target.value));
    }
    onChange(event);
  };

  // continua ...
```

```
  return (
    <Form.Item
      validateStatus={validateStatus}
      label={label}
      help={errorMessage}
      hasFeedback={changed}
    >
      <Input
         {...others}
         onChange={handleValidation}
         suffix={<span />}
      />
    </Form.Item>
  );
}

export default InputText;
```

VALIDAÇÃO

- Agora, sempre que desejar, basta fornecer uma função de validação na propriedade `validate` do componente `InputText`.

pages/Tasks.jsx

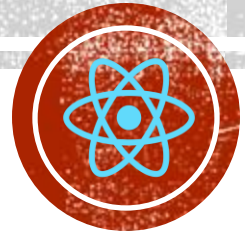
```
// ...
  const validateInputSearch = (value) => {
    return value && value.length > 10
      ? 'O termo de busca deve possui no máximo 10 caracteres.'
      : undefined;
  }
// ...
```

```
// ...
            <InputText
              label="Busca de tarefas"
              placeholder="Buscar por título"
              onChange={handleSearch}
              validate={validateInputSearch}
              value={search}
            />
// ...
```

REFERÊNCIAS

- Ant Design
  - Grid – https://ant.design/components/grid/
  - Input – https://ant.design/components/input/
- Axios - https://github.com/axios/axios
- JSON Placeholder - https://jsonplaceholder.typicode.com

OBRIGADO!



Douglas Nassif Roma Junior

/douglasjunior

/in/douglasjunior

nassifrroma@gmail.com