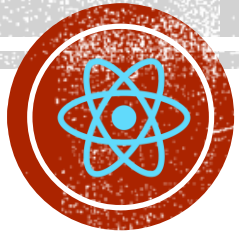
SMARPPY

Bootcamp Catalisa

SINGLE PAGE APPLICATIONS COM REACT-DOM



Douglas Nassif Roma Junior

/douglasjunior

/in/douglasjunior

smarppy.com

douglas@smarppy.com

AGENDA

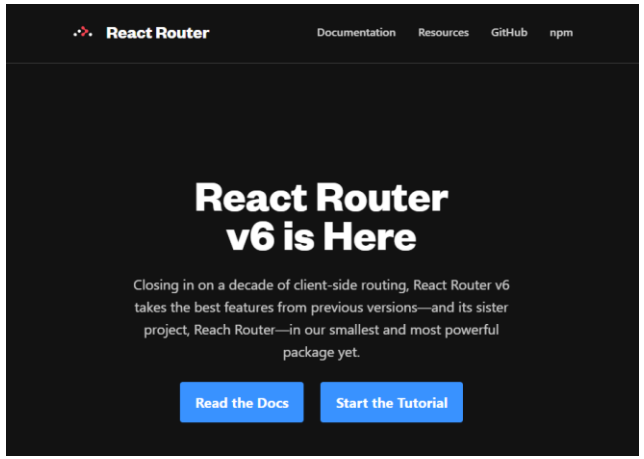
Tópico	Conteúdo
React Router	- Introdução ao React Router - Parâmetros de URL - Rotas privadas - Rotas desconhecidas (404)
Formulários, validação e data-binding	- Criando componentes reusáveis - Validação de formulários - Tabelas e busca - Requests com axios

REACT ROUTER

INTRODUÇÃO AO REACT ROUTER

- **React Router** é uma coleção de componentes de navegação que compõem declarativamente com sua aplicação.

- Se você quer ter URLs navegáveis para seu aplicativo **Web** ou uma maneira componentizada para navegar no **React Native**, o React Router funciona onde quer que o **React** esteja renderizando.



https://reactrouter.com/

INTRODUÇÃO AO REACT ROUTER

- Para instalar o React Router para Web, basta executar:

```
$ npm install react-router-dom
```

- Uso básico:

App.js

```
import {
  HashRouter as Router,
  Routes,
  Route,
  Link,
} from 'react-router-dom';

import Home from './pages/Home';
import Tasks from './pages/Tasks';
import About from './pages/About';

// ...
```

```
// ...
<Router>
  <ul>
    <li><Link to="/">Home</Link></li>
    <li><Link to="/tasks">Tarefas</Link></li>
    <li><Link to="/about">Sobre</Link></li>
  </ul>
  <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/tasks" element={<Tasks />} />
    <Route path="/about" element={<About />} />
  </Routes>
</Router>
// ...
```

PARÂMETROS DE URL

PARÂMETROS DE URL

- **React Router** permite que parâmetros sejam passados entre as rotas, usando parâmetros de URL.

- Para usar parâmetros de URL, basta usar o caractere ":" seguido do nome do parâmetro.

pages/Hero.js

```
import { useParams } from 'react-router-dom';

const Hero = () => {
    const params = useParams();
    return (
        <div>
            <h1>Hero Page</h1>
            <h3>Id: {params.heroId}</h3>
        </div>
    )
}

export default Hero;
```

App.js

```
// ...

    <li><Link to="/hero/strange">Dr Estranho</Link></li>
    <li><Link to="/hero/captain">Cap Amética</Link></li>
    <li><Link to="/hero/spider-man">Home Aranha</Link></li>

// ...

    <Route path="/hero/:heroId" element={<Hero />} />

// ...
```

ROTAS PRIVADAS

ROTAS PRIVADAS

- Usando o componente `Navigate` é possível redirecionar o usuário para uma página determinada.

- Esta abordagem é útil, por exemplo, quando alguns dos componentes exigem autenticação do usuário.

components/PrivateRoute.js

```
import { Navigate } from 'react-router-dom';
import fakeAuth from '../fake-auth';

const PrivateRoute = ({ children }) => (
    fakeAuth.isAuthenticated
        ? children
        : <Navigate to="/login" />
);

export default PrivateRoute;
```

App.js

```
import Login from './pages/Login';
import PrivateRoute from './components/PrivateRoute';
// ...
   <Route path="/tasks" element={
      <PrivateRoute>
         <Tasks />
      </PrivateRoute>
   } />
   <Route path="/login" element={<Login />} />
// ...
```


Continua...

ROTAS PRIVADAS

fake-auth.js

```
const fakeAuth = {
    isAuthenticated: true
}

export default fakeAuth;
```

pages/Login.js

```
import { useNavigate } from 'react-router-dom';
import fakeAuth from '../fake-auth';

const Login = () => {
    const navigate = useNavigate();
    const login = () => {
        fakeAuth.isAuthenticated = true;
        navigate('/tasks')
    }
    return (
        <div>
            <h1>Login page</h1>
            <input placeholder="Usuário"/> <br/>
            <input placeholder="Senha"/> <br/>
            <button onClick={login}>Entrar</button>
        </div>
    );
}

export default Login;
```


Continua...

ROTAS PRIVADAS

pages/Tasks.js

```
import { useNavigate } from 'react-router-dom';
import fakeAuth from '../fake-auth';

const Tasks = () => {
    const navigate = useNavigate();
    const logout = () => {
        fakeAuth.isAuthenticated = false;
        navigate('/login')
    }
    return (
        <div>
            <h1>Tasks page</h1>
            <button onClick={logout}>
                Sair
            </button>
        </div>
    );
}

export default Tasks;
```

ROTAS DESCONHECIDAS (404)

ROTAS DESCONHECIDAS (404)

- Em uma aplicação Web pode ser interessante exibir uma mensagem amigável para um usuário. Com **React Router**, basta utilizar o componente `Route` com `path="*"`.

- Deste modo, se nenhum outro `path` combinar com a URL aberta no navegador, então o `path="*"` será renderizado.

App.js

```
import NotFound from './pages/NotFound';

// ...

    <Route path="*" element={<NotFound />} />

// ...
```

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 nãããoooooo)

CHAMADAS PARA APIS

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

```
$ npm install axios
```

- Vamos aproveitar o 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.js

```
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.js

```
// ...
    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 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

- Vamos criar um `input` de texto para filtrar as *Todos* carregadas no exemplo anterior.

pages/Tasks.js

```
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 ..." onChange={handleSearch} />
        </Form.Item>
      </Form>
    </Col>
  </Row>
// ...
```


Continua...

FORMULÁRIOS E DATA-BINDING

- Em seguida, 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.js

```
// ...
  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]);
// ...
```

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.js

```
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.js

```
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

VALIDAÇÃO

VALIDAÇÃO

- Assim como o "**two-way 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).

- 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.js

```
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} />
    </Form.Item>
  );
}

export default InputText;
```

VALIDAÇÃO

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

```
<InputForm
 label="Filtrar"
 validate={this.validateInputSearch}
 input={{
  name: "todo-search",
  type: "text",
  placeholder: "Buscar por título",
  onChange: this.onSearchChange,
 }}
/>
```

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

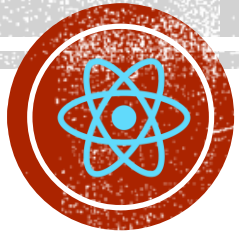
REFERÊNCIAS

- React Router - https://reactrouter.com/docs
- 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

SMARPPY

Bootcamp Catalisa

OBRIGADO!



Douglas Nassif Roma Junior

/douglasjunior

/in/douglasjunior

smarppy.com

douglas@smarppy.com