BOSCH
Invented for life

Aula 6 - React App

Docupedia Export

Author: Siqueira Joao (CtP/ETS)
Date: 04-Jul-2022 13:57

Table of Contents

1 Lista de tarefas 3
2 Criando uma aplicação com React 4
3 Configurações inicias 5
4 JSX 7
5 Salvando dados no localStorage 14
6 Separando os componentes 16
7 Exercício 19
8 Correção 20
9 Criando uma nova aplicação 24
10 Exercício 31
11 Router DOM 32
12 Rotas privadas 34
13 Toastify 36
14 Redux 38
15 rootReducer 41
16 Redux Saga 45
17 Redux Persist 49
18 Projeto React 51
19 Exercício 52

1 Lista de tarefas

2 Criando uma aplicação com React

Para criarmos uma aplicação React basta digitar "npx create-react-app <nome do projeto>"

```
PS U:\JS> npx create-react-app projeto_react

Creating a new React app in U:\JS\projeto_react.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...

[#####.............] - idealTree:webpack-dev-server: timing idealTree:node_modules/webpack-dev-server Completed in 2043ms
```

Ao concluir a criação da pasta podemos rodar a nossa aplicação com "npm start", feito isso devemos esperar um pouco e uma pagina Web ira abrir com a tela inicial do React

```
Compiled successfully!

You can now view projeto_react in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://10.234.198.168:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

3 Configurações inicias

Para começar a criar a nossa própria pagina temos de deletar alguns arquivos começando pela pasta "src" vamos deixar apenas os arquivos index.js, App.js e App.css e na pasta "public" vamos deixar apenas index.html

Agora dentro dos arquivos temos que apagar uma parte deles também, começando pelo index.js

index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

Depois de modificar o index.js devemos modificar o App.js, nesse arquivo utilizamos uma linguagem parecida com HTML, que é o JSX, com ela que poderemos integrar o JavaScript com o HTML e fazer paginas mais dinâmicas

App.js

```
import './App.css';

function App() {
  return (
    <h1>Hellow world!</h1>
  );
}

export default App;
```

Por ultimo no index.html inclua:

index.html

```
<!DOCTYPE html>
```

```
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Exemplo</title>
</head>
<body>
  <noscript>You need to enable JavaScript to run this app.</noscript>
  <div id="root"></div>
</body>
</html>
```

4 JSX

Para criar elementos maiores em JSX precisamos que primeiramente tenha um elemento que englobe todos os outros

App.js

```
1  import './App.css';
2
3  function App() {
4    return (
5      <h1>Hellow world!</h1>
6      <h2>Isso não funciona</h2>
7    );
8  }
9
10 export default App;
```

Esse código não funciona, pois estamos passando mais de um elemento no return, então temos que juntar os dois por uma div ou criar um fragment, que é uma tag vazia

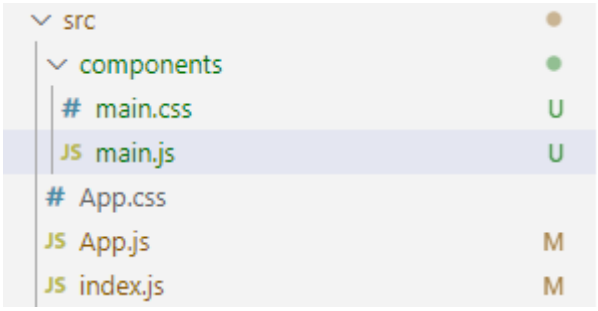
App.js

```
1  import './App.css';
2
3  function App() {
4    return (
5      <>
6        <h1>Hellow world!</h1>
7        <h2>Isso funciona!</h2>
8      </>
9    );
10 }
11
12 export default App;
```

Elementos State

Quando o state muda, o componente responde renderizando novamente.

E para criar esse tipo de elemento temos que primeiro criar uma nova pasta no diretório src e um arquivo main.js e um main.css



Caso queira pode criar um css para usar durante a aula ou usar esse:

main.css

```css
.main{
    background: #fff;
    padding: 30px;
    margin: 50px auto;
    max-width: 500px;
    border-radius: 4px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

.main h1 {
    font-size: 20px;
    text-align: center;
}

.form{
    margin-top: 30px;
    display: flex;
    align-items: center;
    justify-content: center;
}

.form input[type="text"] {
    height: 40px;
    padding: 0 20px;
    font-size: 20px;
    border: 1px solid #ccc;
    min-width: 80%;
}

.form button[type="submit"] {
    border: none;
    height: 40px;
    width: 40px;
    background: #f04c64;
    display: flex;
    align-items: center;
    justify-content: center;
}

.tarefas{
    margin-top: 30px;
}
```

```
43
44  .tarefas li {
45      display: flex;
46      align-items: center;
47      justify-content: space-between;
48      padding: 10px;
49  }
50
51  .tarefas li .edit {
52      margin-right: 15px;
53      color: #51c5de;
54  }
55  .tarefas li .delete {
56      margin-right: 15px;
57      color: #f04c64;
58  }
59
60  .tarefas li .edit,
61  .tarefas li .delete {
62      cursor: pointer;
63  }
64
65  .tarefas li span {
66      display: flex;
67      align-items: center;
68  }
```

Feito isso vamos abrir o main.js e incluir:

Perceba que no JSX o class é className e que sempre precisamos colocar "/" mesmo em tag únicas

main.js

```
1   import React, { Component } from 'react'; // Importando componente
2   import './main.css';
3
4   import { FaPlus } from 'react-icons/fa'; // Biblioteca de icones
    disponiveis para usar no projeto "npm i react-icons"
5
6   class Main extends Component {
7       state = { // Criando o state
8           novaTarefa: '',
9       };
10
11      handleChange = (e) => { // Função que recebe o que esta sendo escrito
12          this.setState({ // Setando o state
13              novaTarefa: e.target.value,
14          });
15      };
16
17      render() {
18          const { novaTarefa } = this.state; // Criando a constante
19
20          return (
21              <div className='main'>
22                  <h1>Lista de tarefas</h1>
23                  <p>Tarefa sendo digitada: { novaTarefa }</p>
```

```
                <form action='#' className='form'>
                    <input onChange={this.handleChange} type='text' />
                    <button type='submit'><FaPlus /></button>
                </form>
            </div>
        );
    };
};

export default Main;
```

Dessa forma já é possível ver o que esta sendo digitado sem a necessidade de dar no form submit

Adicionando itens na lista

Agora para adicionar e excluir vamos ter que mudar um pouco o código adicionando uma array no state:

state - main.js

```
//...
    state = { // Criando o state
        novaTarefa: '',
        tarefas: [],
    };
//...
```

Modificando o submit do form e adicionando um value para o input:

render - main.js

```
//...
 <form action='#' className='form' onSubmit={ this.handleSubmit }>
    <input onChange={this.handleChange} type='text' value={ novaTarefa }/>
    <button type='submit'><FaPlus /></button>
 </form>
//...
```

Incluindo uma tag ul depois do form:

render - main.js

```
//...
 <ul className='tarefas'>
    { tarefas.map((tarefa, index) =>(
        <li key={ tarefa }>
            { tarefa }
            <span>
```

```
                <FaEdit className='edit' />
                <FaWindowClose className='delete' />
            </span>
        </li>
    )) }
 </ul>
//...
```

E por fim incluindo uma nova função:

handleSubmit - main.js

```
//...
 handleSubmit = (e) => {
        e.preventDefault();
        const { tarefas } = this.state;
        let { novaTarefa } = this.state;
        novaTarefa = novaTarefa.trim(); // trim tira todos os espaços que esteja na esquerda ou na direita

        if(tarefas.indexOf(novaTarefa) !== -1) return; // indexOf retorna o index do item na array caso ele exista, caso contrario ele retorna -1

        this.setState({
            tarefas: [...tarefas, novaTarefa],
            novaTarefa: ''
        });
    }
//...
```

Testando agora já é possível adicionar novas tarefas a sua lista

Apagando e editando itens da lista

Para apagar e editar itens da lista vamos incluir:

render - main.js

```
//...
 <ul className='tarefas'>
    { tarefas.map((tarefa, index) =>(
        <li key={ tarefa }>
            { tarefa }
            <span>
                <FaEdit className='edit'  onClick={ (e) => this.handleEdit
(e, index) } />
                <FaWindowClose className='delete' onClick={ (e) => this.ha
ndleDelete(e, index) } />
            </span>
        </li>
    )) }
 </ul>
//...
```

Também uma função de delete:

handleDelete - main.js

```
//...
    handleDelete = (e, index) => {
        const { tarefas } = this.state;
        const novasTarefas = [...tarefas];
        novasTarefas.splice(index, 1);

        this.setState({
            tarefas: [...novasTarefas],
        })
    }
//...
```

Uma função de edit:

handleEdit - main.js

```
//...
    handleEdit = (e, index) => {
        const { tarefas } = this.state;
        this.setState({
            index, // colocamos o index como o index que pegamos do item
```

```
            novaTarefa: tarefas[index],
        });
    };
//...
```

E por fim modificar o handleSubmit e o state:

handleSubmit - main.js

```
//...
    handleSubmit = (e) => {
        e.preventDefault();
        const { tarefas, index } = this.state;
        let { novaTarefa } = this.state;
        novaTarefa = novaTarefa.trim();

        if(tarefas.indexOf(novaTarefa) !== -1) return;

        if(index === -1) {
            this.setState({
                tarefas: [...tarefas, novaTarefa],
                novaTarefa: ''
            });
        } else {
            tarefas[index] = novaTarefa;

            this.setState({
                tarefas: [...tarefas],
                index: -1,
                novaTarefa: ''
            })
        };
    };
//...
```

state - main.js

```
//...
    state = {
        novaTarefa: '',
        tarefas: [],
        index: -1, // Adicionamos um index, para poder verificar se
estamos modificando ou criando um elemento
    };
//...
```

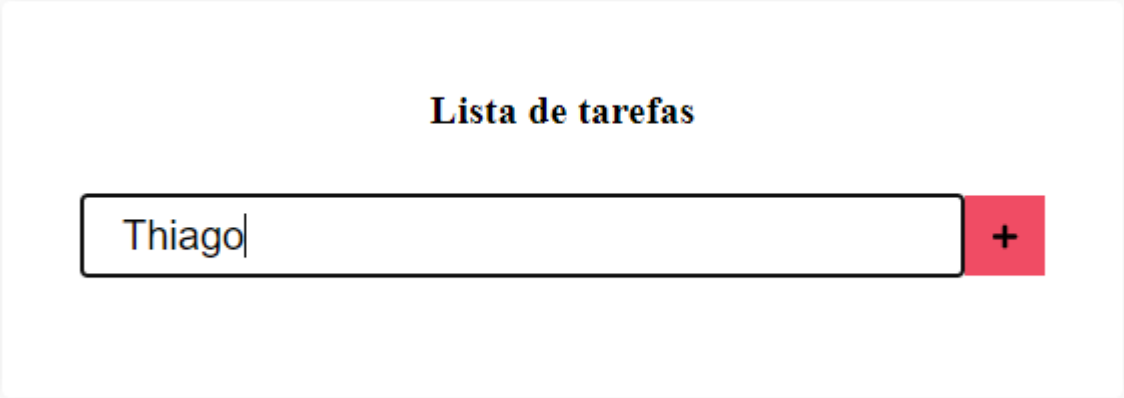
5 Salvando dados no localStorage

Para salvar dados no localStorage vamos usar o componentDidUpdate e componentDidMount.

O componentDidUpdate verifica se algum componente for modificado, por exemplo:

main.js

```
componentDidUpdate(prevProps, prevState) {
        console.log(this.state.novaTarefa);
};
```




```
T
Th
Thi
Thia
Thiag
Thiago
>
```

Cada vez que foi digitado uma letra o componentDidUpdate atualizo e mostro no console a atualização

Dessa forma pode usar esse comando para verificar e salvar dados no localStorage:

main.js

```
componentDidUpdate(prevProps, prevState) // Passamos para a função tanto
as proprieadas anteriores quanto o estado anterior
    {
        const { tarefas } = this.state;

        if (tarefas === prevState.tarefas) return;

        localStorage.setItem('tarefas', JSON.stringify(tarefas));
    };
```

Agora com o componentDidMount podemos recuperar esses dados que foram salvos:

main.js

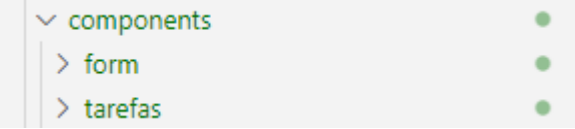
```
componentDidMount() {
        const tarefas = JSON.parse(localStorage.getItem('tarefas'));

        if(!tarefas) return;

        this.setState({ tarefas });
    };
```

Agora mesmo atualizando a pagina não perdemos os dados

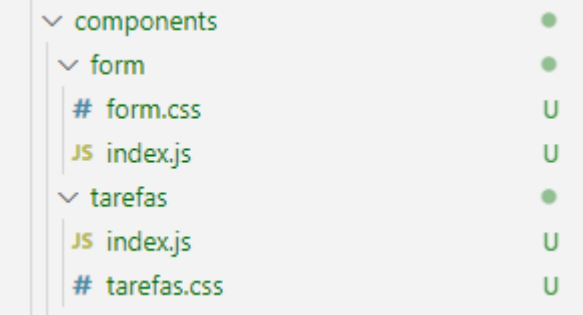
6 Separando os componentes

Separar os arquivos é importante para que o projeto fique mais organizar e fácil de se mexer, por isso é uma boa pratica da programação que facilita tanto para quem esta fazendo o sistema quando para quem for mexer nele no futuro

Para separar os componentes vamos começar criando pastas para cada um dos componentes que não é o main



Dentro das pastas vamos criar um index.js e um css com mesmo nome da pasta



Feito isso temos que instalar o prop-types, "npm i prop-types"

Começando pelo index do form, vamos incluir:

index.js - Form

```
1  import React from 'react';
2  import PropTypes from 'prop-types';
3  import './form.css';
4
5  Function Form() {
6      return(
7
8      );
9  }
10
11 export default Form;
```

E após isso vamos tirar tudo do main.js que esta relacionado com form e colocar no index.js do form:

index.js - Form

```
import React from 'react';
import { FaPlus } from 'react-icons/fa';
import PropTypes from 'prop-types';
import './form.css';

function Form() {
    return(
        <form action='#' className='form' onSubmit={ handleSubmit }>
            <input onChange={ handleChange } type='text'
 value={ novaTarefa }/>
            <button type='submit'><FaPlus /></button>
        </form>
    );
};

export default Form;
```

Lembrando de fazer o mesmo com o main.css e form.css

form.css

```
.form{
    margin-top: 30px;
    display: flex;
    align-items: center;
    justify-content: center;
}

.form input[type="text"] {
    height: 40px;
    padding: 0 20px;
    font-size: 20px;
    border: 1px solid #ccc;
    min-width: 80%;
}

.form button[type="submit"] {
    border: none;
    height: 40px;
    width: 40px;
    background: #f04c64;
    display: flex;
    align-items: center;
    justify-content: center;
}
```

Porem dessa forma os funções ainda não funcionam, então temos que passar elas como parâmetros para a função:

main.js

```
//...
    <h1>Lista de tarefas</h1>
    <Form
        handleSubmit={this.handleSubmit}
        handleChange={this.handleChange}
        novaTarefa={novaTarefa}
    />
//...
```

Também temos que incluir no index.js esses parâmetros e a verificação deles, que é feita com PropTypes:

index.js - Form

```
import React from 'react';
import { FaPlus } from 'react-icons/fa';
import PropTypes from 'prop-types';
import './form.css';

function Form({ handleSubmit, handleChange, novaTarefa }) {
    return(
        <form action='#' className='form' onSubmit={ handleSubmit }>
            <input onChange={ handleChange } type='text'
 value={ novaTarefa }/>
            <button type='submit'><FaPlus /></button>
        </form>
    );
};

Form.propTypes = {
    handleSubmit: PropTypes.func.isRequired,
    handleChange: PropTypes.func.isRequired,
    novaTarefa: PropTypes.string.isRequired
};

export default Form;
```

7 Exercício

Tente agora separar as tarefas da mesma forma que foi feito com o Form

Lembrando que para a verificação com PropTypes de tarefas vamos ter que usar PropTypes.array

8 Correção

Se tudo estiver certo seus arquivos devem estar em torno de:

index.js - Tarefas

```
import React from 'react';
import { FaEdit, FaWindowClose } from 'react-icons/fa';
import PropTypes from 'prop-types';
import './tarefas.css';

function Tarefas({ tarefas, handleEdit, handleDelete }) {
    return(
        <ul className='tarefas'>
            { tarefas.map((tarefa, index) =>(
                <li key={ tarefa }>
                    { tarefa }
                    <span>
                        <FaEdit className='edit' onClick={ (e) =>
handleEdit(e, index) } />
                        <FaWindowClose className='delete' onClick={ (e) =>
handleDelete(e, index) } />
                    </span>
                </li>
            )) }
        </ul>
    );
};

Tarefas.propTypes = {
    handleEdit: PropTypes.func.isRequired,
    handleDelete: PropTypes.func.isRequired,
    tarefas: PropTypes.array.isRequired
};

export default Tarefas;
```

tarefas.css

```
.tarefas{
    margin-top: 30px;
}

.tarefas li {
    display: flex;
    align-items: center;
    justify-content: space-between;
    padding: 10px;
}

.tarefas li .edit {
    margin-right: 15px;
    color: #51c5de;
```

```
}
.tarefas li .delete {
    margin-right: 15px;
    color: #f04c64;
}

.tarefas li .edit,
.tarefas li .delete {
    cursor: pointer;
}

.tarefas li span {
    display: flex;
    align-items: center;
}
```

main.js

```
import React, { Component } from 'react';
import './main.css';
import Form from './form';
import Tarefas from './tarefas';

class Main extends Component {
    state = {
        novaTarefa: '',
        tarefas: [],
        index: -1,
    };

    componentDidMount() {
        const tarefas = JSON.parse(localStorage.getItem('tarefas'));

        if(!tarefas) return;

        this.setState({ tarefas });
    };

    componentDidUpdate(prevProps, prevState)
    {
        const { tarefas } = this.state;

        if (tarefas === prevState.tarefas) return;

        localStorage.setItem('tarefas', JSON.stringify(tarefas));
    };

    handleChange = (e) => {
        this.setState({
            novaTarefa: e.target.value,
        });
    };

    handleSubmit = (e) => {
```

```
        e.preventDefault();
        const { tarefas, index } = this.state;
        let { novaTarefa } = this.state;
        novaTarefa = novaTarefa.trim();

        if(tarefas.indexOf(novaTarefa) !== -1) return;

        if(index === -1) {
            this.setState({
                tarefas: [...tarefas, novaTarefa],
                novaTarefa: ''
            });
        } else {
            tarefas[index] = novaTarefa;

            this.setState({
                tarefas: [...tarefas],
                index: -1,
                novaTarefa: ''
            })
        };
    };

    handleEdit = (e, index) => {
        const { tarefas } = this.state;
        this.setState({
            index,
            novaTarefa: tarefas[index],
        });
    };

    handleDelete = (e, index) => {
        const { tarefas } = this.state;
        const novasTarefas = [...tarefas];
        novasTarefas.splice(index, 1);

        this.setState({
            tarefas: [...novasTarefas],
        })
    };

    render() {
        const { novaTarefa, tarefas } = this.state;

        return (
            <div className='main'>
                <h1>Lista de tarefas</h1>
                <Form
                    handleSubmit={ this.handleSubmit }
                    handleChange={ this.handleChange }
                    novaTarefa={ novaTarefa }
                />

                <Tarefas
                    handleEdit={ this.handleEdit }
                    handleDelete={ this.handleDelete }
                    tarefas={ tarefas }
                />
            </div>
        );
```

```
    };
};

export default Main;
```

tarefas.css

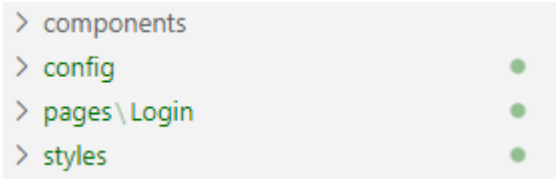
```
.main{
    background: #fff;
    padding: 30px;
    margin: 50px auto;
    max-width: 500px;
    border-radius: 4px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

.main h1 {
    font-size: 20px;
    text-align: center;
}
```

9 Criando uma nova aplicação

Vamos criar outra aplicação agora, então siga as configurações iniciais novamente , porem dessa vez vamos excluir o App.css também

Nessa aplicação iremos utilizar o styled components, que é uma outra forma de estilizar as paginas, uma das vantagens do styled em relação ao css é a possibilidade de usar funções pra modificar as tags

Antes de criarmos a nossa primeira pagina vamos criar as pastas pages, components, styles e config, e dentro de pages vamos criar Login



Agora criamos um index.js dentro de Login e incluímos:

index.js - Login

```
import React from 'react';

export default function Login() {
    return (
        <>
            <h1>
                Login
                <small>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut ultricies leo et urna fringilla, tempus dictum est gravida.</small>
            </h1>
            <p>Pellentesque facilisis tempus lobortis.</p>
        </>
    );
};
```

E em App.js incluímos:

App.js

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

function App() {
  return (
    <>
      <Login />
    </>
  );
};

```

```
11  export default App;
```

Feito isso vamos começar a estilizar com styled, para isso temos que instalar ele, baixar a extensão e criar um arquivo styled.js no mesmo diretório do que vai ser estilizado, no caso o index.js da pagina de Login


```
PS C:\Users\liq1ct\Documents\projeto_react> npm i styled-components
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.
```

No styled.js inclua:

App.js

```
1   import styled from 'styled-components';
2
3   export const Title = styled.h1`
4       color:  ${ props => props.isRed ? 'red' : 'blue' }; // Ao invés de
    passarmos um cor direto para o titulo agora podemos mudar ela direto no
    index.js
5       small {
6           font-size: 0.5em;
7           margin-left: 15px;
8           color: black;
9       }
10  `;
```

Já no index.js inclua:

App.js

```
1   import React from 'react';
2
3   import { Title } from './styled';
4
5   export default function Login() {
6       return (
7           <>
8               <Title isRed>
9                   Login
10                  <small>Lorem ipsum dolor sit amet, consectetur adipiscing
    elit. Ut ultricies leo et urna fringilla, tempus dictum est gravida.</
    small>
```

```
            </Title>
            <p>Pellentesque facilisis tempus lobortis.</p>
        </>
    );
};
```

Perceba que trocamos o h1 por Title, mas não perdemos o formato da tag por que o styled da tag Title também é h1, além disso como foi passado o parâmetro isRed, o Login ficou vermelho, porem se tirarmos ele ficara azul

Agora vamos criar um estilo global, para isso temos que criar dentro da pasta styles um arquivo GlobalStyles.js

Nesse arquivo vamos criar um estilo global e também as tag que serão utilizadas em todas as paginas do site

Por exemplo:

GlobalStyles.js

```
import styled, { createGlobalStyle } from 'styled-components';

export default createGlobalStyle`
    * {
        margin: 0;
        padding: 0;
        outline: 0;
        box-sizing: border-box;
    }

    body {
        font-family: sans-serif;
        background: #eee;
    }

    html, border-style, #root {
        height: 100%;
    }

    button {
        cursor: pointer;
    }

    a {
        text-decoration: none;
    }

    ul {
        list-style: none;
    }
`;

export const Container = styled.section`
    max-width: 75%;
    background: #fff;
    margin: 30px auto;
```

```
    padding: 30px;
    border-radius: 4px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
`;
```

Agora para esse estilo global começar a funcionar no site temos que importa-lo no App.js:

App.js

```
import Login from './pages/Login';
import GlobalStyles from './styles/GlobalStyles';

function App() {
  return (
    <>
      <Login />
      <GlobalStyles />
    </>
  );
};

export default App;
```

E importar o container no index.js:

index.js

```
import React from 'react';

import { Container } from '../../styles/GlobalStyles';
import { Title } from './styled';

export default function Login() {
    return (
        <Container>
            <Title isRed>
                Login
                <small>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut ultricies leo et urna fringilla, tempus dictum est gravida.</small>
            </Title>
            <p>Pellentesque facilisis tempus lobortis.</p>
        </Container>
    );
};
```

Um boa pratica que ajuda muito na estilização do site é a criação de uma paleta de cores, nos podemos criar ela com um arquivos colors.js dentro da pasta config, por exemplo:

colors.js

```
export const primaryColor = '#C3073F';
export const secondaryColor = '#fff';
export const darkColor = '#1A1A1D';
```

Agora no GlobalStyles.js podemos importar essas cores e utiliza-las assim como podemos fazer em outras paginas, dessa forma se precisarmos mudar uma cor será mais já que teremos que mexer somente em um arquivo

GlobalStyles.js

```
import styled, { createGlobalStyle } from 'styled-components';
import { darkColor, secondaryColor } from '../config/colors'

export default createGlobalStyle`
    * {
        margin: 0;
        padding: 0;
        outline: 0;
        box-sizing: border-box;
    }

    body {
        font-family: sans-serif;
        background: ${ darkColor };
    }

    html, border-style, #root {
        height: 100%;
    }

    button {
        cursor: pointer;
    }

    a {
        text-decoration: none;
    }

    ul {
        list-style: none;
    }
`;

export const Container = styled.section`
    max-width: 75%;
    background: ${ secondaryColor };
    margin: 30px auto;
    padding: 30px;
    border-radius: 4px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
`;
```

Criando um Header

Para criar o Header primeiro temos que criar uma pasta Header dentro de components e os arquivos index.js e styled.js dentro de Header

Feito isso inclua no index.js:

GlobalStyles.js

```
import React from 'react';
import { FaHome,  FaUserAlt, FaSignOutAlt } from 'react-icons/fa';
import { Nav } from './styled';

export default function Header() {
    return (
        <Nav>
            <a href='/'>
                <FaHome size={24} />
            </a>
            <a href='login'>
                <FaUserAlt size={24} />
            </a>
            <a href='logout'>
                <FaSignOutAlt size={24} />
            </a>
        </Nav>
    );
}
```

E no styled.js inclua:

styled.js

```
import styled from "styled-components";
import { secondaryColor, primaryColor } from '../../config/colors'


export const Nav = styled.nav`
    background: ${ primaryColor };
    padding: 20px;
    display: flex;
    align-content: center;
    justify-content: right;

    a {
        color: ${ secondaryColor };
        margin: 0 10px 0 0;
        font-weight: bold;
    }
`;
```

E por fim no App.js inclua:

App.js

```
import Login from './pages/Login';
import GlobalStyles from './styles/GlobalStyles';
import Header from './components/Header';

function App() {
  return (
    <>
      <Header />
      <Login />
      <GlobalStyles />
    </>
  );
};

export default App;
```

10 Exercício

Agora que temos um Header para a nossa aplicação, temos que fazer um Footer, crie um pasta Footer em components e faça o seu próprio Footer da mesma forma que foi feto o Header

11 Router DOM

Assim como no backend existe roteamento no frontend também há roteamento, e para organizar isso temos que usar o pacote Router DOM, instalamos ele com "npm i react-router-dom"

Após instalar o pacote crie uma pasta routes dentro de src, e um arquivo index.js dentro dessa pasta, nesse arquivo inclua:

index.js - routes

```
import React from 'react';
import { Routes, Route } from 'react-router-dom';
import Login from '../pages/Login';

export default function Urls() {
    return (
        <>
            <Routes>
                <Route exact path='/' element={<Login />} />
                {/* Criamos uma rota com o caminho para a raiz do site, e dizemos que ela tem que ser exata, quando isso acontecer o componente Login é mostrado */}
            </Routes>
        </>
    );
}
```

Agora App.js troque o Login por Routes:

App.js

```
import GlobalStyles from './styles/GlobalStyles';
import Header from './components/Header';
import Routes from './routes';
import { BrowserRouter } from 'react-router-dom';

function App() {
  return (
    <BrowserRouter>
      <Header />
      <Routes />
      <GlobalStyles />
    </BrowserRouter>
  );
};

export default App;
```

Dessa forma qualquer outra url que for usado em nosso site não vai funcionar, então podemos criar uma página 404

Para isso inclua no index das rotas:

index.js - routes

```
import React from 'react';
import { Routes, Route } from 'react-router-dom';

import Login from '../pages/Login';
import Page404 from '../pages/Page404';
```

```
6
7   export default function Urls() {
8       return (
9           <>
10              <Routes>
11                  <Route exact path='/' element={<Login />} />
12                  <Route path='*' element={<Page404 />} />
13              </Routes>
14          </>
15      );
16  }
```

Exercício

Cria a página 404 e estilize ela usando os conhecimento de styled-components

12 Rotas privadas

Muitas rotas em um sistemas podem ser fechadas para um login, para fazer essa verificação temos que fazer nosso sistema de verificação de rotas

Para isso crie na pasta routes um arquivo PrivateRoute.js e instale "npm prop-types"

Nesse novo arquivo inclua:

PrivateRoute.js

```
import React from 'react';
import { Navigate, Outlet, useLocation } from 'react-router-dom';
import PropTypes from 'prop-types';

export default function PrivateRoute(isClosed) {
    const isLoggedId = false;
    let location = useLocation();
    if (isClosed.isClosed && !isLoggedId) { // Caso a rota seja privada e o usuário não esteja logado vamos redireciona-lo para login
        return (
            <Navigate
                to="/login"
                state={{ from: location }}  // Depois de login voltamos o usuário para a ultima página que ele esteve
            />
        );
    }

    return <Outlet />;
}

PrivateRoute.defaultProps = {
    isClosed: false // Como isClosed não é requerido temos que passar um valor padrão
};

PrivateRoute.propTypes = {
    isClosed: PropTypes.bool
};
```

Feito isso temos que substituir Routes por privateRoutes no index.js do routes

index.js - routes

```
import React from 'react';
import { Routes, Route } from 'react-router-dom';

import PrivateRoute from './PrivateRoute';
import Login from '../pages/Login';
import Page404 from '../pages/Page404';

export default function Urls() {
```

```
    return (
        <Routes>
            <Route exact path='/' element={<PrivateRoute />}>
                <Route exact path='/' element={<Login />} />
            </Route>
            <Route path='*' element={<Page404 />} />
        </Routes>
    );
}
```

13 Toastify

Vamos usar o React Toastify para exibir mensagens de erro no nosso sistema, essas mensagem podem ser usadas de diversas maneira, mas as mais comuns são de erro no login ou no cadastro

Para começar temos que instalar "npm i react-toastify"

Depois de instalado temos que importa-lo no App.js

App.js

```
import GlobalStyles from './styles/GlobalStyles';
import Header from './components/Header';
import Routes from './routes';
import { BrowserRouter } from 'react-router-dom';
import { ToastContainer } from 'react-toastify'

function App() {
  return (
    <BrowserRouter>
      <Header />
      <Routes />
      <GlobalStyles />
      <ToastContainer
        autoClose={3000} //Define quanto tempo o alerta vai ficar na tela
      />
    </BrowserRouter>
  );
};

export default App;
```

Feito isso temos que ir em GlobalStyles.js e incluir:

GlobalStyles.js

```
//...
    import 'react-toastify/dist/ReactToastify.css';
//...
```

Agora podemos adicionar alertas com mensagens em nosso sistema, por exemplo:

PrivateRoute.js

```
import React from 'react';
import { toast } from 'react-toastify';
import { Navigate, Outlet, useLocation } from 'react-router-dom';
import PropTypes from 'prop-types';

export default function PrivateRoute(isClosed) {
```

```
    const isLoggedId = false;
    let location = useLocation();
    if (isClosed.isClosed && !isLoggedId) {
        toast.error('Acesso negado');
        return (
            <Navigate
                to="/login"
                state={{ from: location }}
            />
        );
    }
    toast.success('Acesso liberado');
    return <Outlet />;
}

PrivateRoute.defaultProps = {
    isClosed: false
};

PrivateRoute.propTypes = {
    isClosed: PropTypes.bool
};
```

14 Redux

O redux é usado para termos parâmetros que são usados no sistema inteiro, por exemplo se o usuário esta ou não logado ou se há algo no carrinho de compra, para fazer isso o redux verifica todas as ações que são disparadas no site,

analisando as ações disparadas podemos usar o reducer para mudar um estado global caso alguma ação especifica aconteça, lembrando que não é uma boa pratica de programação manipular o estado atual, sempre se deve criar um novo estado,

manipula-lo e depois passar ele como novo estado

Entendido isso vamos instalar o "npm i redux react-redux redux-saga redux-persist"

Feito isso vamos criar uma pasta chamada store dentro de src e um arquivo index.js dentro dessa pasta, nesse arquivo vamos incluir:

index.js - store

```
import { createStore } from 'redux';

const initialState = {
    logado: false,
}; // Todos os estados iniciais

const reducer = (state = initialState, action) => {
    const newState = { ...state }; // Criamos o novo estado
    switch(action.type) {
        case 'LOGIN':
            newState.logado = true; // Caso a ação seja de login deixamos o estado como verdadeiro
            return newState;

        case 'LOGOUT':
            newState.logado = false; // Caso a ação seja de logout deixamos o estado como falso
            return newState;

        default:
            return state;
    }
};

const store = createStore(reducer);

export default store;
```

Com o redux criado, vamos colocar ele para rodar junto do resto do sistema

App.js

```
import GlobalStyles from './styles/GlobalStyles';
import Header from './components/Header';
```

```
import Routes from './routes';
import store from './store';

import { Provider } from 'react-redux';
import { BrowserRouter } from 'react-router-dom';
import { ToastContainer } from 'react-toastify'

function App() {
  return (
    <Provider store={store}>
      <BrowserRouter>
        <Header />
        <Routes />
        <GlobalStyles />
        <ToastContainer
          autoClose={3000}
        />
      </BrowserRouter>
    </Provider>
  );
};

export default App;
```

Pronto agora temos como criar variáveis para o sistema, para começar vamos fazer um sistema simples de login e logout

index.js - Header

```
import React from 'react';
import { FaHome,  FaUserAlt, FaSignOutAlt } from 'react-icons/fa';
import { Nav } from './styled';
import { useDispatch, useSelector } from 'react-redux';

export default function Header() {
    const logado = useSelector(state => state.logado);

    const dispath = useDispatch();

    function handleLogin(e) {
        e.preventDefault(); // Desativamos as funções normais do botão para que a pagina não seja atualizada

        dispath({
            type: "LOGIN" // Passamos que o tipo da ação é de login
        });
    }

    function handleLogout(e) {
        e.preventDefault();

        dispath({
            type: "LOGOUT" // Passamos que o tipo da ação é de logout
        });
    }
    return (
```

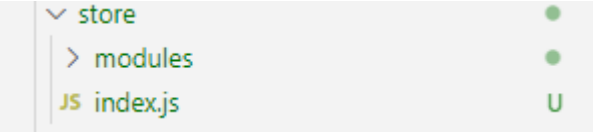
```
27          <Nav>
28              {logado ? 'Logado' : 'Não Logado'}
29              <a href='/'>
30                  <FaHome size={24} />
31              </a>
32              <a href='login' onClick={handleLogin}>
33                  <FaUserAlt size={24} />
34              </a>
35              <a href='logout'>
36                  <FaSignOutAlt size={24} onClick={handleLogout}/>
37              </a>
38          </Nav>
39      );
40  }
```

15 rootReducer

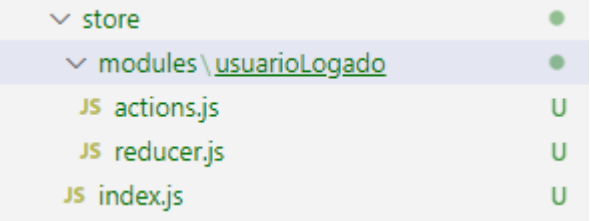
Em algumas aplicações pode haver a necessidade de ter mais de um reducer, e para fazer o sistema ler mais de uma ação por vez temos que usar o rootReducer, para isso vamos primeiro separar nossas ações em módulos,

cada modulo é responsável por um parâmetro global

Para começar vamos criar uma pasta modules dentro de store e uma pasta usuarioLogado dentro de modules



Agora dentro de usuarioLogado vamos criar actions.js e reducer.js



Feito isso dentro de index.js - Header tire os valores dos types dentro do objeto de dispath:

index.js - Header

```
//...
    dispath({
        type: "LOGIN"
    });
//...
    dispath({
        type: "LOGOUT"
    });
//...
```

index.js - Header

```
//...
    dispath({
        type:
    });
//...
    dispath({
        type:
    });
//...
```

Então vamos em actions e retornamos esses funções com esses objetos:

ations.js - usuarioLogado

```
export function login() {
    return {
        type: "LOGIN"
    }
}

export function logout() {
    return {
        type: "LOGOUT"
    }
}
```

E importamos eles em index.js - Header

index.js - Header

```
import * as usuarioLogado from '../../store/modules/usuarioLogado/actions'

//...
    dispath({
        type: usuarioLogado.login()
     });
//...
    dispath({
        type: usuarioLogado.logout()
     });
//...
```

Agora em index.js - store vamos tirar tudo que é relacionado ao reducer da contaLogado e movelo para reducer.js - ususarioLogado:

index.js - store

```
import { createStore } from 'redux';

const initialState = {
    logado: false,
};

const reducer = (state = initialState, action) => {
    const newState = { ...state };
    switch(action.type) {
        case 'LOGIN':
            newState.logado = true;
            return newState;

        case 'LOGOUT':
```

```
            newState.logado = false;
            return newState;

        default:
            return state;
    }
};

const store = createStore(reducer);

export default store;
```

index.js - store

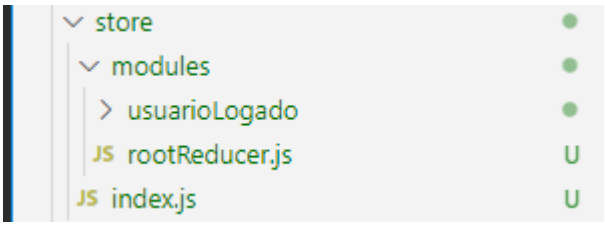
```
import { createStore } from 'redux';

const store = createStore();

export default store;
```

Movendendo tudo isso para o reducer.js - usuarioLogado fica:

reducer.js - usuarioLogado

```
const initialState = {
    logado: false,
};

export default function (state = initialState, action) {
    const newState = { ...state };
    switch(action.type) {
        case 'LOGIN':
            newState.logado = true;
            return newState;

        case 'LOGOUT':
            newState.logado = false;
            return newState;

        default:
            return state;
    }
};
```

Pronto agora temos tudo preparado para fazer o rootReducer, em vamos cria-lo dentro da pasta modules:



E incluímos nele:

rootReducer.js

```
import { combineReducers } from 'redux';

import usuarioLogadoReducer from './usuarioLogado/reducer';

export default combineReducers({
    usuarioLogadoReducer,
});
```

E em index.js - store incluimos:

index.js - store

```
import { createStore } from 'redux';

import rootReducer from './modules/rootReducer';

const store = createStore(rootReducer);

export default store;
```

Porem agora o nossos parâmetros ficam dentro de um objeto, então temos que mudar o modo de chamar eles em index.js - Header

index.js - store

```
//...
    const logado = useSelector(state =>
state.usuarioLogadoReducer.logado); // Temos que adicionar o nome da chave
passa em rootReducer
//...
```

16 Redux Saga

Em requisições assíncronas precisamos saber se as promises vão retornar resolve ou reject, então só podemos mudar o estado da nossa aplicação após essa promise ser realizada, para isso podemos usar o middleware redux saga

e para começar a usar ele temos que modificar um pouco nosso código, começando por colocar todos os types em um arquivo único, então crie um arquivo types.js dentro de modules e inclua:

types.js

```
export const BOTAO_LOGIN = "LOGOUT";
export const BOTAO_LOGOUT = "LOGIN";
export const LOGIN_REQUEST = "LOGIN_REQUEST";
export const LOGOUT_REQUEST = "LOGOUT_REQUEST";
export const BOTAO_FAILURE = "FAILURE";
```

Agora em actions.js - usuarioLogado e reducer.js - usuarioLogado temos que mudar a chamada dos types e adicionar os cases e funções de request e failure:

actions.js - usuarioLogado

```
import * as types from '../types'

export function login() {
    return {
        type: types.BOTAO_LOGIN
    };
}

export function logout() {
    return {
        type: types.BOTAO_LOGOUT
    };
}

export function login_request() {
    return {
        type: types.LOGIN_REQUEST
    };
}

export function logout_request() {
    return {
        type: types.LOGOUT_REQUEST
    };
}

export function failure() {
    return {
        type: types.BOTAO_FAILURE
    };
```

```
31  }
```

reducer.js - usuarioLogado

```
1   import * as types from '../types';
2
3   const initialState = {
4       logado: false,
5   };
6
7   // eslint-disable-next-line
8   export default function(state = initialState, action) {
9       const newState = { ...state };
10
11      switch(action.type) {
12          case types.BOTAO_LOGIN:
13              console.log("Login realizado");
14              newState.logado = true;
15              return newState;
16
17          case types.BOTAO_LOGOUT:
18              console.log("Logout realizado");
19              newState.logado = false;
20              return newState;
21
22          case types.LOGIN_REQUEST:
23              console.log("Fazendo requisição");
24              return state;
25
26          case types.LOGOUT_REQUEST:
27              console.log("Fazendo requisição");
28              return state;
29
30          case types.BOTAO_FAILURE:
31              console.log("Requisição negada");
32              return state;
33
34          default:
35              return state;
36      };
37  };
```

Agora vamos começar a utilizar o sagas, para isso dentro de usuarioLogado crie um arquivo sagas.js e inclua:

sagas.js - usuarioLogado

```
1   import { call, put, all, takeLatest } from 'redux-saga/effects';
2   import * as actions from './actions';
3   import * as types from '../types';
4
```

```
const requisicao = () =>
    new Promise((resolve, reject) => { // Como não estamos conectados a nenhum backend vamos simular uma requisição
        setTimeout(() => {
            resolve();
        }, 2000);
    });

function* loginRequest() {
    try {
        yield call(requisicao);
        yield put(actions.login());
    } catch {
        yield put(actions.failure());
    }
}

function* logoutRequest() {
    try {
        yield call(requisicao);
        yield put(actions.logout());
    } catch {
        yield put(actions.failure());
    }
}

export default all([
    takeLatest(types.LOGIN_REQUEST, loginRequest), // O usuário pode clicar varias vezes no botão, mas só a ultima que será realizada por causa do takeLatest
    takeLatest(types.LOGOUT_REQUEST, logoutRequest),
]);
```

Com o saga criado precisamos de um rootSaga também, para caso haja mais de um modulo com saga todos possam funcionar

Então em modules crie um rootSaga.js e inclua:

rootSagas.js

```
import { all } from 'redux-saga/effects';

import usuarioLogadoSaga from './usuarioLogado/sagas';

export default function* rootSaga() {
    return yield all([
        usuarioLogadoSaga,
    ]);
}
```

Com isso pronto só temos que passar o rootSagas como middleware para o nosso index.js - store, então nele inclua:

sagas.js

```
import { createStore, applyMiddleware } from 'redux';
import createSagaMiddleware from 'redux-saga';

import rootReducer from './modules/rootReducer';
import rootSaga from './modules/rootSaga';

const sagaMiddleware = createSagaMiddleware();

const store = createStore(rootReducer, applyMiddleware(sagaMiddleware));

sagaMiddleware.run(rootSaga);

export default store;
```

Clicando nos botões de login e logout teremos no console

```
Fazendo requisição
Login realizado
Fazendo requisição
Logout realizado
```

17 Redux Persist

O redux persist serve para salvarmos parâmetros no navegador assim como fizemos com localStorage, mas com o persist podemos fazer isso de modo mais simples e também é garantido que a página só será renderizada depois de puxar todos os dados salvos,

para começar vamos criar um arquivo reduxPersist.js dentro de modules e incluir nele:

reduxPersist.js

```
import storage from 'redux-persist/lib/storage';
import { persistReducer } from 'redux-persist';

// eslint-disable-next-line
export default reducers => {
    const persistReducers = persistReducer(
        {
            key: "REACT-BASE",
            storage,
            whitelist: ['usuarioLogadoReducer']
        },
        reducers
    );

    return persistReducers;
}
```

Feito isso vamos adicionar o persist no index.js - store:

index.js - store

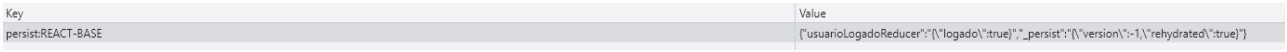
```
import { persistStore } from 'redux-persist';
import { createStore, applyMiddleware } from 'redux';
import createSagaMiddleware from 'redux-saga';

import persistedReducer from './modules/reduxPersist';
import rootReducer from './modules/rootReducer';
import rootSaga from './modules/rootSaga';

const sagaMiddleware = createSagaMiddleware();

const store = createStore(
    persistedReducer(rootReducer),
    applyMiddleware(sagaMiddleware)
);

sagaMiddleware.run(rootSaga);

export const persistor = persistStore(store);
export default store;
```

Agora so precisamos envolver nosso App.js com o persist:

App.js

```
import { Provider } from 'react-redux';
import { BrowserRouter } from 'react-router-dom';
import { ToastContainer } from 'react-toastify'
import { PersistGate } from 'redux-persist/integration/react'

import GlobalStyles from './styles/GlobalStyles';
import Header from './components/Header';
import Routes from './routes';
import store, { persistor } from './store';


function App() {
  return (
    <Provider store={store}>
      <PersistGate persistor={persistor}>
        <BrowserRouter>
          <Header />
          <Routes />
          <GlobalStyles />
          <ToastContainer
            autoClose={3000}
          />
        </BrowserRouter>
      </PersistGate>
    </Provider>
  );
};

export default App;
```

Agora no console é possível ver os dados armazenados e também podemos atualizar a pagina sem perder a nossa variável:



Key	Value
persist:REACT-BASE	{"usuarioLogadoReducer":"{\"logado\":true}","_persist":"{\"version\":-1,\"rehydrated\":true}"}

18 Projeto React

Para começar baixe o projeto "projeto_react.zip", há poucas diferenças entre esse projeto e o antigo, uma das maiores dela sendo o axios que vai servir para conectarmos com a API Rest

Para utilizar as rotas que criamos na API vamos na página em que desejamos fazer essa consulta no banco de dados e incluímos:

index.js - Produtos

```
import React, { useState } from 'react';

import { Container } from '../../styles/GlobalStyles';
import axios from '../../services/axios';


export default function Produtos() {
    const [produtos, setProdutos] = useState([]); // Usamos para inicializar uma variavel que vai ser puxado do banco de dados, no caso queros que ela seja inicializada como array

    React.useEffect(() => {
        async function getData() {
            const response = await axios.get('/List'); // Passamos a rota que queremos fazer buscar na API
            setProdutos(response.data); // Usando o setProdutos vamos iniciar uma varavel array chamada produtos, como definimos acima
        }
        getData();

    }, [])

    return (
        <Container>
            <h1>Produtos</h1>
            <br />
            {produtos.map(produto => (
                <div key={String(produto.Codigo)}>
                    {produto.Descricao}
                </div>
            ))}
        </Container>
    );
};
```

19 Exercício

- Faça uso de todas rota que tiver na sua API Rest
- A página de criação de novo produto deve ser privada e ter algum tipo de autenticação para a pessoa conseguir acessar ela
- A autenticação deve passar pelo Redux Saga para garantir que será executada através de uma promise
- A autenticação deve ser salva no localStorage
- As páginas devem estilizadas usando styled-component