



Plataformas de aplica  es Web

Aula 12 - PetTopStore - PDV - Parte 1



Material Did tico do Instituto Metr pole Digital - IMD

Termo de uso

Os materiais did ticos aqui disponibilizados est o licenciados atrav s de Creative Commons **Atribui  o-SemDeriva  es-SemDerivados CC BY-NC-ND**. Voc  possui a permiss o para realizar o download e compartilhar, desde que atribua os cr ditos do autor. N o poder  alter -los e nem utiliza-los para fins comerciais.

Atribui  o-SemDeriva  es-SemDerivados

CC BY-NC-ND



<https://creativecommons.org/licenses/by-nc-nd/4.0/>

Apresentação

Nas próximas duas aulas iremos criar o sistema de PDV (Ponto de venda) da nossa solução da loja Pet Top Store.

O PDV é um sistema de uso exclusivo dos funcionários da loja e tem a função de criar novas vendas para os clientes da loja física.

Usaremos uma tecnologia chamada "React" para criar o sistema do PDV que irá se integrar com o sistema da administração através da API criada na aula passada para todas as suas operações.

As funcionalidades do sistema PDV serão

- Logar no sistema como funcionário
- Manter o token de autenticação salvo no LocalStorage do navegador
- Cadastrar clientes, caso eles não tenham cadastro ainda na loja.
- Realizar nova venda para um cliente com múltiplos produtos.

Essas próximas aulas não se tratam de um curso completo de React. Além de não ser algo possível em poucas aulas, a nossa intenção (assim como nas aulas anteriores) é dar uma visão geral de como cada tecnologia funciona e os caminhos para você aprender mais.

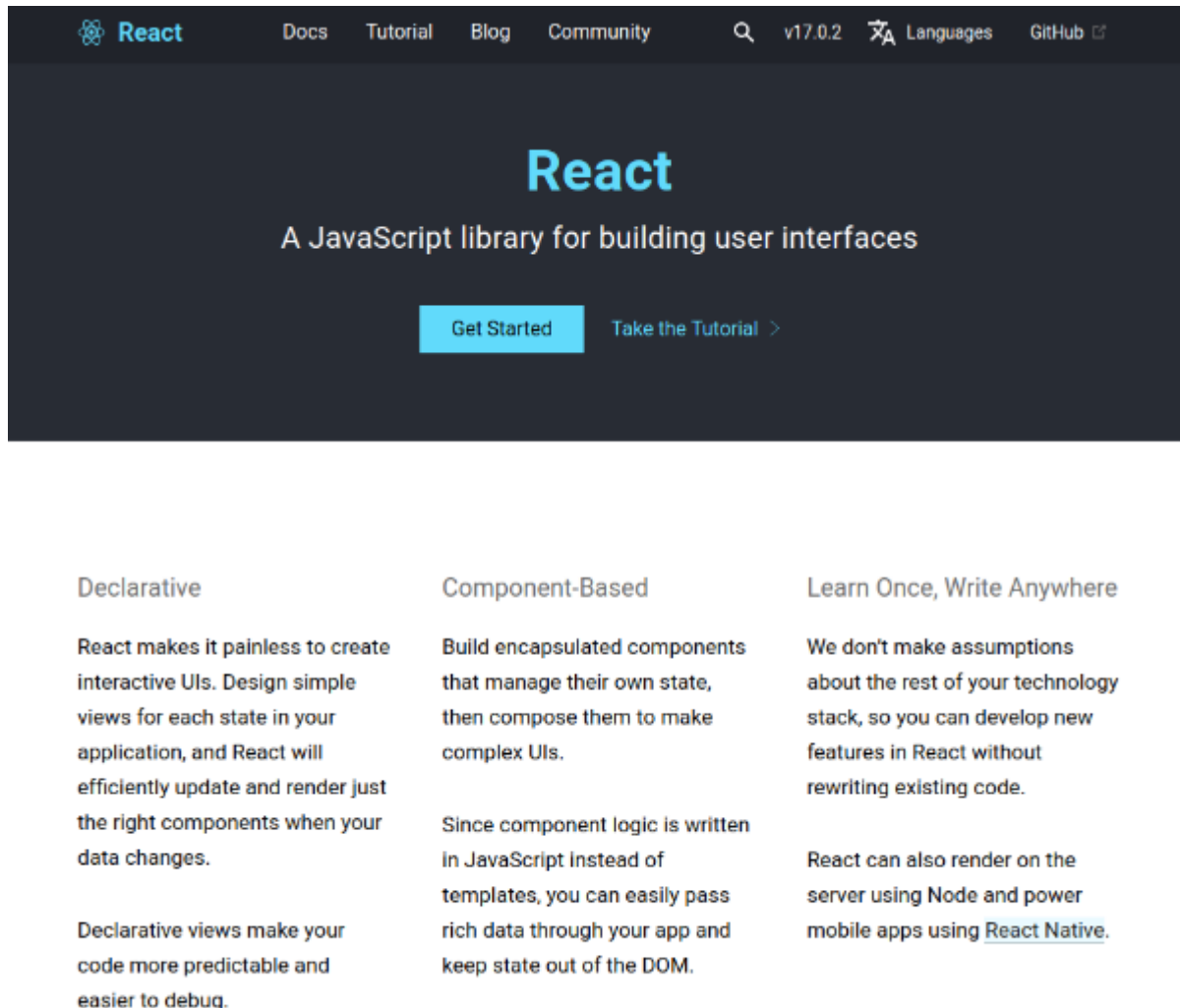
Baixe o projeto admin+API

Antes de iniciar você pode baixar o projeto administrativo e API, caso deseje: [Clique aqui para download](#)

React

website: <https://react.dev/>

Criado pelo Facebook em 2011 e liberado para a comunidade em 2013, o React se define como uma "biblioteca Javascript para se criar interfaces de usuário".



Website

do React Com o React você consegue criar interfaces interativas com mais facilidade através de uma arquitetura baseada em componentes que gerenciam seu próprio estado e usam esses dados para renderizar HTML no lado do cliente.

As lógicas dos componentes do react são escritas em Javascript ao invés de templates como o EJS e o estado de cada componente se trata de basicamente dados em variáveis especiais que podem ser alterados de maneira interativa, fazendo com que a interface "reaja" a essas mudanças, se alterando e apresentando os novos valores de forma correta, assim como vimos no VueJS.

Um simples componente React

No React componentes podem ser usando classes ou funções Javascript em conjunto com uma tecnologia chamada JSX, que lembra o HTML ou o XML e pode ser utilizado "dentro" de códigos Javascript de forma mais clara que em templates.

Um simples componente React usando uma classe e JSX é algo como:

```
class HelloMessage extends React.Component {  
  render() {
```

```
    return (  
      <div>  
        Hello {this.props.nome}  
      </div>  
    );  
  }  
}  
  
ReactDOM.render(  
  <HelloMessage nome="Maria José" />,  
  document.getElementById('hello-example')  
);
```

Resultado:

Hello Taylor

Repare que o componente `HelloMessage` é uma classe que estende `React.Component` e nessa classe existe um método chamado `"render()"` que retorna um bloco de código JSX que nada mais é que uma junção entre javascript e HTML similar ao uso de templates, mas com mais restrições e mais performático. O bloco é uma `DIV` com o texto `"Ol {this.props.nome}"`, e isso será substituído pelo atributo `"nome"` passado para o componente quando ele for utilizado. `"props"` é o objeto que guarda esses atributos.

Logo abaixo do código o componente `HelloMessage` é utilizado com o atributo `nome="Maria José"` o que faz com que dentro do componente exista `this.props.nome` com o valor em questão.

A parte que faz `ReactDOM.render(.....)` é usada quando se está se desejando se renderizar um componente do react em um elemento da página. Nesse caso o componente é o `HelloMessage` e o elemento seria algum qualquer com o id `'hello-example'` que exista em seu HTML.

Além disso, o React permite que você tenha um gerenciamento local do estado do componente com o que chamamos de `"state"`.

Existem duas formas de se criar componentes com o React e, da mesma forma, duas formas de se gerenciar states dentro de componentes.

Digamos que no exemplo do componente `HelloMessage`, a gente deseja criar um contador de cliques no nome. Seria necessário uma variável de estado para se guardar o número de vezes que o nome foi clicado e uma forma de atualizar o seu valor quando houver um novo click.

A primeira forma é usando classes (como já está), e essa funcionalidade seria assim:

```
class HelloMessage extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      contador: 0
    }
    // necessário usar o bind para usar o "this.state" dentro do
    método
    this.clicou = this.clicou.bind(this);
  }

  clicou(e) {
    this.setState({
      contador: this.state.contador + 1
    })
  }
  render() {
    return (
      <div onClick={this.clicou} style={{cursor: "pointer"}}>
        Oi {this.props.nome}<br/>
        clicou {this.state.contador} vezes
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage nome="Maria José" />,
  document.getElementById('hello-example')
);
```

Resultado:

```
Oi Maria José
clicou 10 vezes
```

Repare que algumas coisas precisaram ser feitas. Primeiro a classe precisa de um construtor que chama o método `super(props)`, depois cria uma variável de classe especial chamada `this.state` com um valor inicial com o contador = 0. Depois é necessário criar um método chamado `"clicou(e)"` e no construtor realizar um `bind`

dele com o "this" para se poder usar o this.setState dentro do método para trocar o valor do estado. Com tudo isso pronto, aí sim podemos dentro do render() agora colocar o {this.state.contador} para ser exibido e um onClick={this.clicou} no div principal para executar o método que incrementa o contador e altera o estado do componente.

É muito trabalhoso sim, e a equipe do React criou uma forma mais simples de se realizar o gerenciamento de estados locais, o React Hooks.

React Hooks

Os Hooks no react permitem você criar componentes sem criar classes, além de gerenciar estados sem criar o objeto especial this.state. É a forma recomendada de se criar componentes no React hoje em dia.

State Hook

Um componente similar ao que criamos anteriormente, porém utilizando o state Hook seria assim:

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

export function Contador(props) {
  // Declare a new state variable, which we'll call "count"
  const [contador, setContador] = useState(props.contadorInicial);
  const [nome, setNome] = useState(props.nomeInicial);

  return (
    <div onClick={() => setContador(contador + 1)} style={{cursor:
"pointer"}}>
      Oi {nome}<br/>
      clicou {contador} vezes
    </div>
  );
}
```

Bem mais simples né? Temos somente uma função chamada Contador que recebe os props (atributos) iniciais e utiliza o "useState" para se criar pares de variáveis especiais onde uma tem o valor do estado e a outra uma função para mudar o seu estado, então:

```
const [contador, setContador] = useState(props.contadorInicial);
```

Na verdade cria contador e setContador, com o valor inicial de contador igual a props.contadorInicial, que é um atributo passado ao componente quando ele é utilizado na aplicação, por exemplo assim:

```
<Contador nomeInicial="Isaac Franco" contadorInicial={1}></Contador>
```

Effect Hook

O Effect Hook permite você executar "efeitos colaterais" em componentes funcionais, ou seja, você pode com ele criar ações que alteram outros dados, quando algum outro dado é alterado.

Para usar o Effect Hook você precisa importar a função useEffect do react.

O useEffect recebe dois parâmetros. O primeiro é uma função que será executada ao reagir a mudança de dados. O segundo é um array com quais dados serão "monitorados", ou seja, quais são as variáveis que, quando mudam de valor, faz com que a função do primeiro parâmetro seja executada. É mais simples do que parece. Exemplo:

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

export function Contador(props) {
  // Declare a new state variable, which we'll call "count"
  const [contador, setContador] = useState(props.contadorInicial);
  const [nome, setNome] = useState(props.nomeInicial);
  const [mensagem, setMensagem] = useState('Contador ainda pequeno');

  useEffect(() => {
    if (contador > 5) {
      setMensagem("Contador está maior agora!");
    }
  }, [contador]);

  return (
    <div onClick={() => setContador(contador + 1)} style={{cursor: "pointer"}}>
      Oi {nome}<br/>
      clicou {contador} vezes<br/>
      Mensagem: {mensagem}
    </div>
  );
}
```

```
}
```

Veja que no exemplo acima temos mais uma variável de estado chamada "mensagem" que inicial com o texto "Contador ainda pequeno" e existe um `useEffect` que verifica se o contador é maior que 5, se for altera a mensagem. Esse `useEffect` "depende" de que o contador mude para que ele execute, e portanto `[contador]` é passado como segundo parâmetro do `useEffect`, que irá monitorar seu valor e executar a função passada quando ele mudar.

O `useEffect` pode ser chamado sem o segundo atributo, e então ele se comporta como algo que é executado somente na primeira vez que o componente for montado.

Para saber mais sobre React Hooks visite: <https://react.dev/reference/react>

Para saber mais sobre o JSX no React visite:
<https://react.dev/learn/writing-markup-with-jsx>

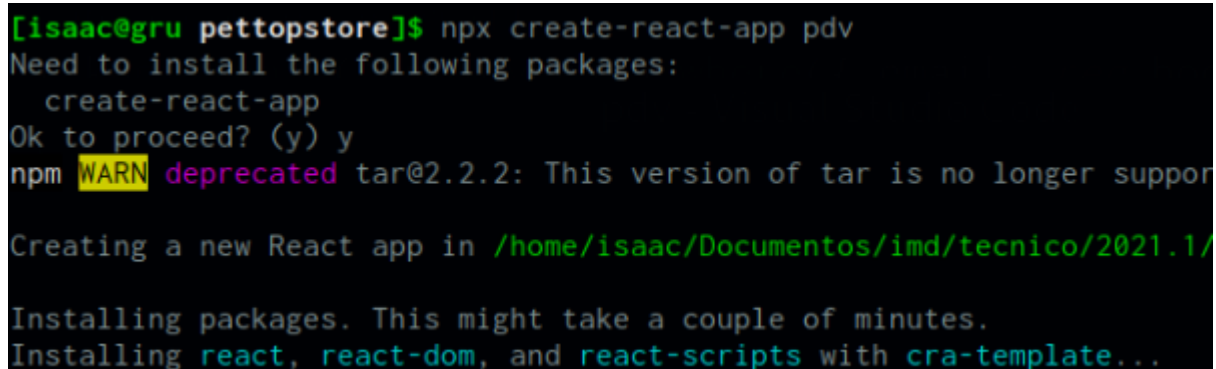
Criando o aplicativo PDV com o React

Como você deve ter visto na documentação do React, existem vários frameworks que criam aplicações que usam a tecnologia. Next.js, Remix e Gatsby são alguns mencionados e cada um deles tem suas peculiaridades.

Vamos usar nesse exemplo uma ferramenta bastante popular chamada `create-react-app` para criar o PDV. Mais informações em <https://create-react-app.dev/>

```
npx create-react-app pdv
```

Responda "y" quando perguntado se deseja prosseguir



```
[isaac@gru pettopstore]$ npx create-react-app pdv
Need to install the following packages:
  create-react-app
Ok to proceed? (y) y
npm WARN deprecated tar@2.2.2: This version of tar is no longer supported
Creating a new React app in /home/isaac/Documentos/imd/tecnico/2021.1/
Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...
```

Alguns

instantes depois...


```
Success! Created pdv at /home/isaac/dev/plataformas_de_aplicacoes_web/final_new/pettopstore/pdv
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd pdv
  npm start
```

Legenda da imagem

Entre na pasta criada:

```
cd pdv
```

Vamos agora configurar o sistema do PDV para rodar na porta 4000 por padrão.

Edite o arquivo **package.json** e adicione "PORT=4000 " no início do script start:

```
"scripts": {
  "start": "PORT=4000 react-scripts start",
  "build": "react-scripts build",
  "test": "react-scripts test",
  "eject": "react-scripts eject"
},
```

Legenda da imagem

Para iniciar o seu projeto do PDV em React entre na pasta "pdv" e execute o app na porta 4000

```
npm start
```

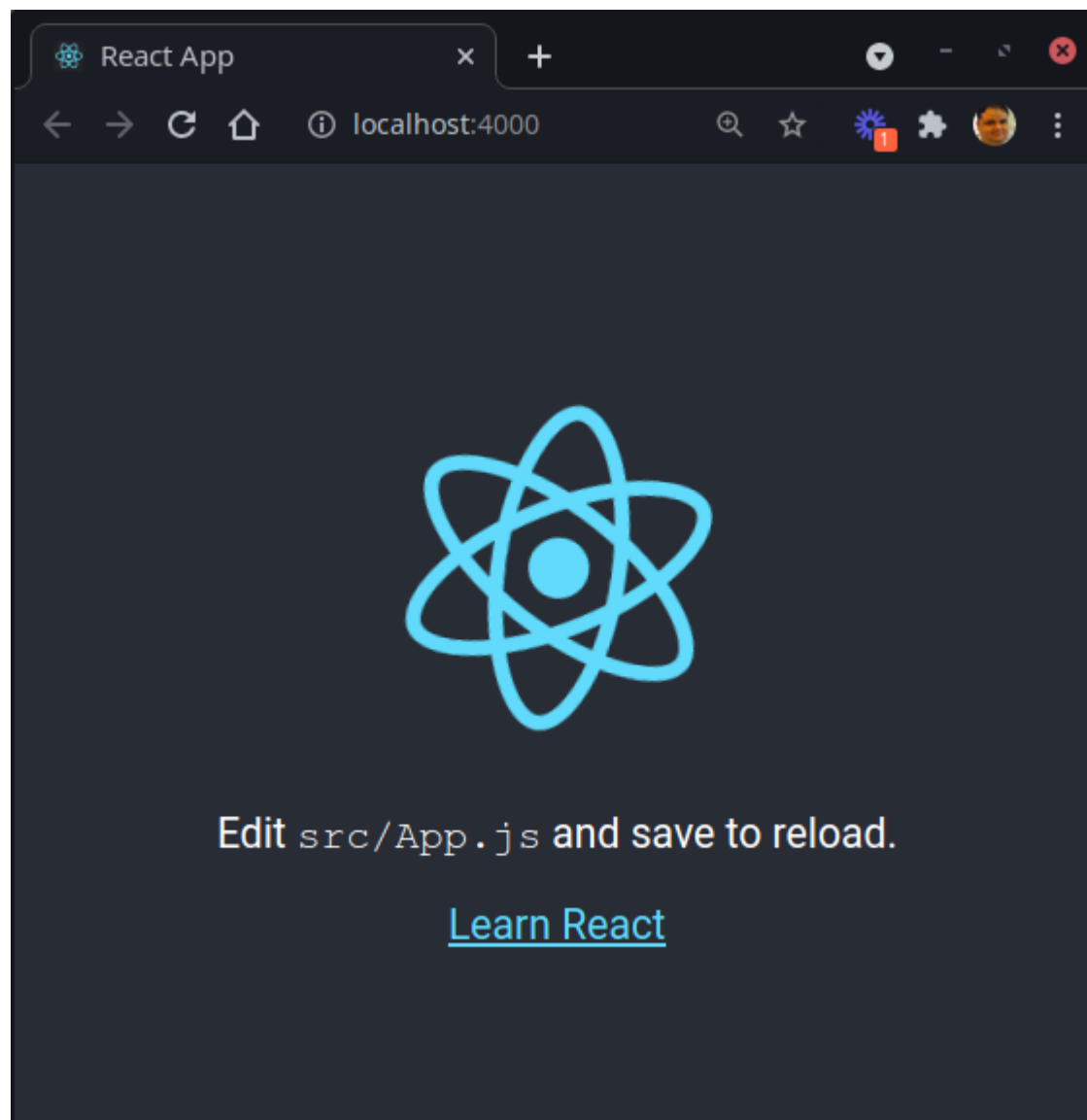
```
Compiled successfully!
```

```
You can now view pdv in the browser.
```

```
Local:      http://localhost:4000  
On Your Network: http://192.168.129.20:4000
```

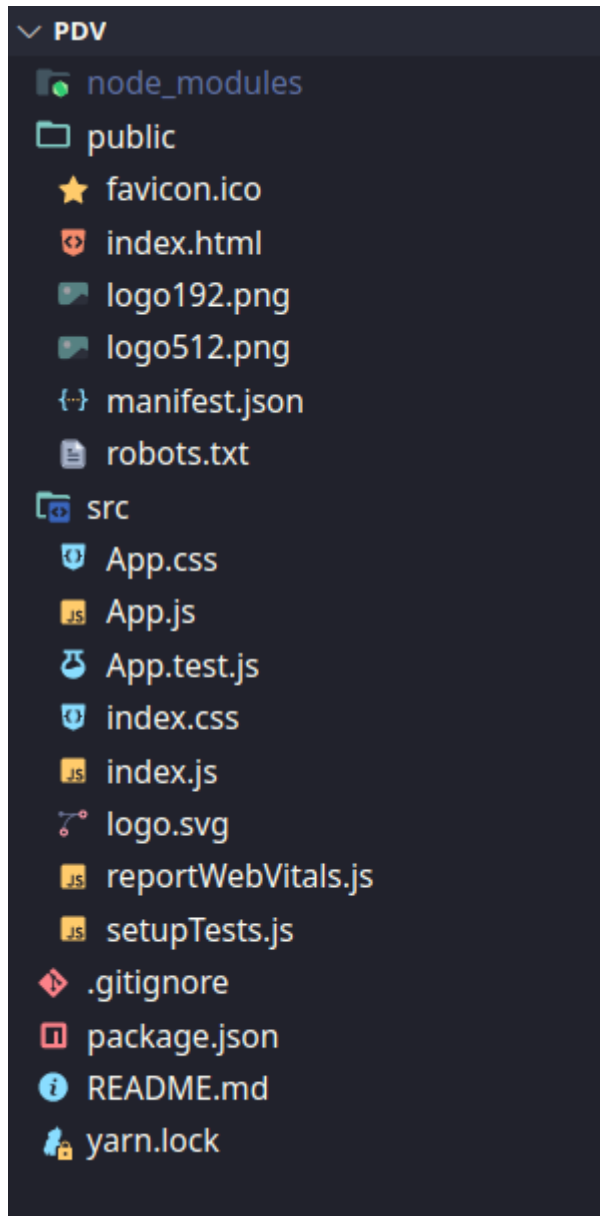
```
Note that the development build is not optimized.  
To create a production build, use yarn build.
```

```
[]
```



Escolhemos a porta 4000 pois a porta 3000 (padrão) já está sendo usada pela nossa API.

Pronto. agora basta digitar "npm start" e o pdf irá executar na porta 4000 automaticamente.



Estrutura de um projeto criado com o create-react-app

Componentes do PDV

Dependência

Antes de iniciar, por motivos de compatibilidade, precisamos instalar uma biblioteca chamada **buffer** pois iremos usar ela para decodificar uma parte do token JWT sem de maneira mais leve.

Na pasta do projeto, digite:

```
npm install buffer
```

Componentes

O nosso PDV terá quatro componentes

- **src/App.js**: Determina se o funcionário está logado e mostra o src/Sale.js (se estiver) ou src/Login.js (se não estiver)
- **src/Login.js**: Componente com um formulário de login
- **src/Sale.js**: Formulário para se adicionar novas vendas no sistema
- **src/AddClient.js**: Componente auxiliar que permite a criação de novos clientes na loja pelo funcionário.

Cada um desses componentes React será responsável por uma funcionalidade.

Existe um outro componente chamado src/Index.js que é padrão em um projeto criado com o create-react-app, mas ele somente importa o componente src/App.js o insere no public/index.html, iniciando assim a renderização do nosso aplicativo React na página. Sim, nosso aplicativo react tem somente uma página HTML base (public/index.html) e todo o restante dos elementos visuais serão componentes React interativos. Esse tipo de sistema é chamado de SPA (Single Page Application).

Iremos abordar os componentes src/App.js e src/Login.js nessa aula e o src/Sale.js e src/AddClient.js na aula seguinte

Componente src/App.js

É o componente será responsável por determinar se o usuário está logado ou não e dependendo do caso ele vai renderizar o componente src/Login.js ou src/Sale.js

Vamos ver o código completo comentado do src/App.js:

```
import './App.css'; // Importando um CSS para estilizar o App
import Sale from './Sale'; // Importando o componente de venda
import Login from './Login'; // Importando o componente de login
import { useState, useEffect } from 'react'; // React Hooks useState e useEffect
import { Buffer } from 'buffer'; // Biblioteca Buffer para manipular o JWT

// função auxiliar que determina se um token JWT está próximo de expirar (falta 1 hora ou menos)
function isTokenOld(jwt_token) {

  // converte o payload do JWT para um objeto Javascript
  const payload = JSON.parse(Buffer.from(jwt_token.split(".")[1], 'base64').toString('binary'));
```

```
// Calcula a expiração baseado no atributo "exp" o token. Esse foi
seratado quando se assinou o token com a expiração de 2 dias(se
assim você fez)
const expiration = new Date(payload.exp * 1000);
const now = new Date();
const oneHour = 1000 * 60 * 60;

// verifica se o token está próximo de expirar(menos de 1 hora),
baseado na sua expiração e na data atual
if( expiration.getTime() - now.getTime() < oneHour ){
  return true;
} else {
  return false;
}
}

// Componente principal App
function App() {

  // Estamos criando a variável de estado "employee" usando o
useState de forma especial
  // Aqui o useState recebe uma função como valor inicial.
  // isso significa que o retorno dessa função será o valor inicial
de employee.
  const [employee, setEmployee] = useState(() => {
    // 0 que essa função faz é verificar se "employee" está no
localStorage do navegador
    const saved = localStorage.getItem("employee");
    const employeeJSON = JSON.parse(saved);

    // se ele estiver no localStorage, então vamos verificar se o
token desse employee está perto de expirar
    if (employeeJSON) {
      if (isTokenOld(employeeJSON.token)){
        // se o token é antivo (está perto de expirar), retornamos
um valor inicial fazio '' para o estado inicial de employee
        return '';
      }
    }

    // caso contrário retornamos o employeeJSON que contém o
funcionário salvo no localStorage, ou em branco se ele não existir
no localStorage ainda (nunca foi feito login nesse navegador)
    return employeeJSON;
  });
}
```

```
});

// Estamos agora usando o useEffect da seguinte maneira:
// Se employee mudar, ou seja, o login for realizado pela API e
ele receber o valor de retorno
// então essa função abaixo passada ao useEffect será executada
// e ela seta a chave 'employee' no localStorage do navegador para
o valor que a variável employee tem.
// Essa estratégia, junto ao useState do employee, faz com que seu
valor fique sempre sincronizado com o que está salvo no
localStorage.
useEffect(() => {
  localStorage.setItem("employee", JSON.stringify(employee));
}, [employee]);

// Finalmente verificamos se employee existe.
if (employee) {
  //Se for o caso ele já está logado e o token não está próximo de
expirar()
  // retornar o componente de realizar venda (Sale)
  // Repare que passamos tanto a variável de estado "employee"
como o setEmployee para o componente Sale, permitindo que ele tenha
acesso a esses dados que são de responsabilidade desse componente.
  return <Sale employee={employee} setEmployee={setEmployee} />
} else {
  // Se não for o caso, o funcionário não está logado ou seu token
está próximo de expirar.
  // Retornar o Login.js que tem o formulário de login.
  // Repare que passamos tanto a variável de estado "employee"
como o setEmployee para o componente Login, permitindo que ele tenha
acesso a esses dados que são de responsabilidade desse componente.
  return <Login employee={employee} setEmployee={setEmployee} />
}

}

export default App;
```

Como visto nos comentários do código, o `src/App.js` é o componente principal do PDV, ele verifica se o `employee` (funcionário) está logado com uma estratégia que mistura o gerenciamento de estados do React com uma variável `"employee"` criada com o `useState`, mas utiliza `localStorage` para manter essa variável salva no armazenamento do navegador. Isso faz com que, mesmo que você recarregue a página, o funcionário se mantenha logado.

Foi necessário criar uma função que verifica se o token expirou (está próximo de expirar, na verdade) para evitar exibir o sistema para um token que não tem muito tempo de vida ainda. Nesse caso é preferencial mostrar o componente de Login.

No final o `src/App.js` verifica se, depois de todas as verificações, o `employee` existe é porque o funcionário logou no sistema. Se não existe, não logou. Para cada caso ele mostra um componente diferente (Sale ou Login).

Repare que passamos tanto para o componente Sale como para o Login o `employee` e o `setEmployee`. Isso faz com que dentro deles esses atributos possam ser acessados através dos "props" e a variável `employee` possa ser mudada pelos por esses componentes. Caso, por exemplo, o componente Login execute `setEmployee` (que foi passado pra ele), o efeito é o mesmo que isso tivesse sido feito dentro de `src/App.js` e uma reatividade irá acontecer, salvando o `employee` no `localStorage` e alterando o retorno do componente, fazendo com que o componente Sale apareça (se o login aconteceu com sucesso).

Segue também o conteúdo do arquivo `App.css`, que tem um estilo básico para o PDV:

```
.App {
  text-align: center;
}

.App-logo {
  height: 40vmin;
  pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}
```

```
.App-link {
  color: #61dafb;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
```

Para mais informações sobre o localStorage visite:

https://www.w3schools.com/html/html5_webstorage.asp

Componente src/Login.js

Esse componente recebe o "employee" e a função setEmployee como atributo (props) e é responsável por realizar o login, também chamado de sign-in do funcionário na API que criamos no admin. Caso o login seja efetivado com sucesso o valor de employee deve ser alterado com a função setEmployee.

Verifique o código e os comentários explicativos de cada comando no src/Login.js para entender como funciona todo o processo

Conteúdo comentado de src/Login.js:

```
import './Login.css';
import { useState } from "react";

// Componente login
function Login(props) {

  // variáveis de estado (email e password) com valores iniciais em
  branco
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');

  // função que realiza o login na API com o uso do fetch
  function do_login() {
    fetch('http://localhost:3000/api/auth/employee/sign_in', {
      method: 'POST',
      headers: {
```



```
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({ email, password })
})
.then((res) => {
  if (res.ok) {
    // caso o login tenha acontecido com sucesso
    // altera o valor de employee com o uso do setEmployee
    res.json().then(json => {
      props.setEmployee(json.employee);
    })
  } else {
    // Em caso de falha, exibir uma mensagem de erro
    alert('E-mail/senha inválidos');
  }
})
}
```

// Retorna um JSX que mostra um formulário de login com os INPUTS associados às variáveis

// de estado email e password

// o botão "Entrar" inicial o processo de login chamando "do_login" quando clicado.

```
return (
  <div className="Login">
    <h1>PetTopStore</h1>
    <h2>PDV - Autenticação</h2>
    <div className="LoginBox">
      Email:
      <input
        name="email"
        type="email"
        value={email}
        onInput={e => setEmail(e.target.value)}
      /><br/>
      Password:
      <input
        name="password"
        type="password"
        value={password}
        onInput={e => setPassword(e.target.value)}
      /><br/>
      <button
        onClick={do_login}
```

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

export default Login;
```

Aqui o conteúdo do Login.css para a estilização básica:

```
.Login
{
    display: flex;
    justify-content: center;
    flex-direction: column;
    align-items: center;
    color: cadetblue;
}

.LoginBox {
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: flex-start;
    padding: 20px;
    border: 2px solid cornflowerblue ;
    color: cadetblue;
}

input {
    color: cadetblue;
    border: 1px solid cornflowerblue;
    padding: 10px;
    font-size: 1.2em
}

button {
    color: cadetblue;
    border: 1px solid cornflowerblue;
    background-color: white;
    padding: 10px;
    font-size: 1em;
```

```
}
```

Iremos implementar src/Sale.js na próxima aula, mas por enquanto coloque no arquivo src/Sale.js um conteúdo vazio assim:

Conteúdo temporário de src/Sale.js:

```
function Sale(props) {  
  return <div>Realizar venda não implementado ainda</div>  
}  
  
export default Sale;
```

Execute novamente o app PDV Rect com o comando:

```
npm start
```

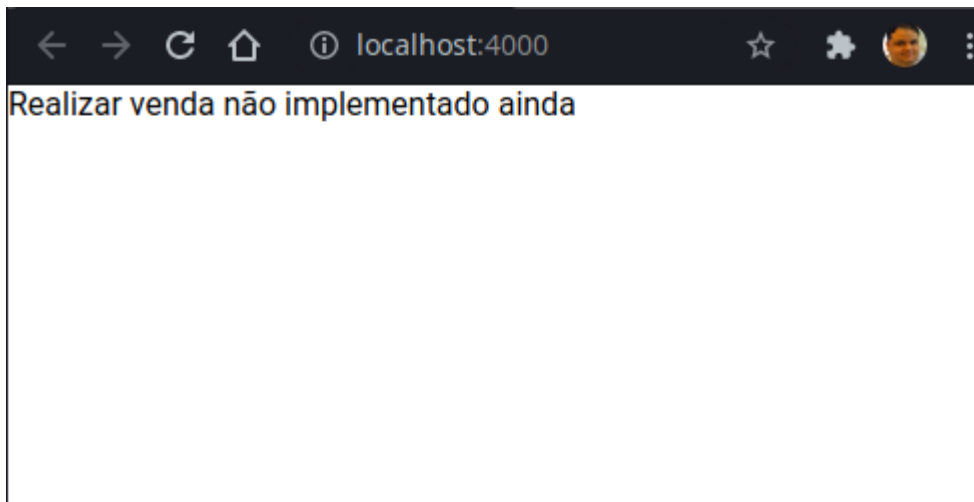
E acesse <http://localhost:4000/>

Você deverá ver algo assim:



Se você logou com um funcionário correto (email e senha) criado na aula anterior

no seu banco de dados através das seeds, você deverá conseguir logar no sistema e verá algo assim:



Não se preocupe, vamos implementar a tela de vendas na próxima a

Conclusão

Criamos a primeira parte do nosso app de PDV com a tecnologia React, consumindo a API do sistema admin para realizar o login do funcionário e fizemos um mecanismo para salvar o token JWT gerado pela api no localStorage do navegador.

Baixe os projetos

Você já pode baixar o app PDV completo: [Clique aqui para download](#)

O link acima contém o PDV já completo, que iremos explorar os detalhes restantes na próxima aula, mas você já pode ir olhando a implementação das outras funcionalidades.

Assim como disponibilizado no início da aula, aqui segue o link também do sistema de administração e API que é necessário para ser usado em conjunto com o PDV: [Clique aqui para download](#)