



ReactJs: Interfaces Front-End

Caderno de Atividades

Anderson Iwanezuk Thaczuk



ReactJs:
Interfaces Front-End

Caderno de Exercícios

Anderson Iwanezuk Thaczuk

Sumário

Sumário	3
Apresentação do curso	5
Versões	6
Instalação	6
CAPÍTULO 1 – O AMBIENTE DE TRABALHO	7
Atividade 1 – Preparar o ambiente de trabalho com Node.js	7
Atividade 2 – Instalar e configurar o Visual Studio Code	11
Atividade 3 – Acessar o Terminal no VS Code e criar app ReactJs	13
Atividade 4 – “Olá Mundo!” para cumprir a tradição.....	14
Atividade Extra.....	16
CAPÍTULO 2 – CRIANDO E NAVEGANDO NA SPA COM REACT	17
Atividade 1 – Criando páginas com ReactJs.....	17
Atividade 2 – CSS base utilizando variáveis.....	20
Atividade 3 – React Components para CSS específico	23
Atividade 4 – Uso de Rotas no ReactJs.....	24
Atividade Extra.....	26
CAPÍTULO 3 – APERFEIÇOANDO A SPA.....	27
Atividade 1 – Criando a página de erro 404.....	27
Atividade 2 – Carregamento Web Tradicional x SPA.....	29
Atividade 3 – Criando e integrando navegação ao projeto	32
Atividade 4 – Otimizando o carregamento de páginas na SPA.....	34
Atividade Extra.....	35
CAPÍTULO 4 – INTRODUÇÃO A API	36
Atividade 1 – Conectando com a API	36
Atividade 2 – Recebendo os dados da API.....	38
Atividade 3 – Estados de componente e visualização dinâmica de dados	39
Atividade 4 – Renderizando um componente específico	42
Atividade Extra.....	45
CAPÍTULO 5 – APERFEIÇOANDO O USO DA API	46
Atividade 1 – Criando Links para exibição por categoria	46
Atividade 2 – Rotas aninhadas por categoria.....	48
Atividade 3 – Exibindo Posts por categoria.....	50

Atividade 4 – Refinando as categorias em subcategorias.....	52
Atividade Extra.....	55
CAPÍTULO 6 – INTRODUÇÃO AO CRUD COM REACT	56
Atividade 1 – Preparando a área administrativa	56
Atividade 2 – Utilizando a biblioteca de estilos ‘MUI’	59
Atividade 3 – Formulário e cadastro de novas Categorias.....	62
Atividade 4 – Excluindo as categorias	65
Atividade Extra.....	66
CAPÍTULO 7 – MAIS UM POUCO DE CRUD COM REACT	67
Atividade 1 – Alterando as Categorias	67
Atividade 2 – Envio de dados do admin para subcategorias.....	70
Atividade 3 – Formulário das SubCategorias.....	74
Atividade 4 – Alterando os dados das subcategorias	76
Atividade Extra.....	78
CAPÍTULO 8 – FINALIZANDO O CRUD COM REACT	79
Atividade 1 – Navegação aninhada	79
Atividade 2 – Lista dos Posts	83
Atividade 3 – Formulário dos Posts com select dinâmico	86
Atividade 4 – Finalizando o admin, os Posts e projeto.....	89
Atividade Extra.....	92

Apresentação do curso

Atualmente, a internet se tornou um veículo de comunicação de grande abrangência, que atinge mercados públicos e privados, permeando todas as áreas da sociedade. Diante do volume de informações que oferece e a forma como é apresentada, é fundamental estabelecer formas de organização dos conteúdos, ampliar a interatividade e permitir o aumento de performance dos sites, tornando desta forma, a experiência de navegação mais rica e assertiva. Segundo pesquisa TIC Domicílios¹(2020), 70% dos brasileiros estão conectados à internet, mais de 46,5 milhões de domicílios possuem acesso e representam 67% do total e ainda, 48% adquiriram ou usaram algum tipo de serviço on-line, como aplicativos de transporte, streaming de filmes, música ou pedido de comida.

Segundo a pesquisa realizada pela plataforma de desenvolvedor Stack Overflow² o ReactJS é o framework mais procurado por um em cada quatro desenvolvedores; das 66.202 respostas, 25,12% dos desenvolvedores manifestaram interesse no uso do ReactJS.

“...Alunos estão despreparados para aproveitar as estruturas que os empregadores precisam, com as maiores lacunas de conhecimento residentes em AngularJS, React, Node.js e Spring. Nenhum aluno de um país poderia atender a metade das exigências do quadro dos empregadores³...”

O React é a biblioteca mais popular e amplamente utilizada por empresas grandes, médias e pequenas. Basta simples buscas em sites de vagas de trabalho para se ter essa percepção. Essa biblioteca do JavaScript é usada para construir uma interface, oferecendo ao usuário a possibilidade de adicionar comandos usando o método de renderizar sites.

Os componentes foram desenvolvidos pelo Facebook em 2013 como uma ferramenta JavaScript de código aberto e permanece a frente das concorrentes, como a Angular e a Bootstrap. O React é usado por grandes empresas como: Netflix, Airbnb, American Express, Facebook, WhatsApp, eBay, Instagram, Walmart e The New York Times.

O ReactJs permite que você reuse componentes em várias aplicações com a mesma função que é uma vantagem importante para desenvolvedores em geral. O componente do ReactJs é fácil de escrever porque usa JSX, uma excelente combinação de JavaScript e HTML que simplifica toda a estrutura de codificação de um site, fazendo com que a renderização de múltiplas funções seja mais fácil, até em componentes especiais ou aplicações de alto rendimento. Ao aprender ReactJs é necessário apenas um pouco mais para desenvolver Apps

¹ https://www.cetic.br/media/docs/publicacoes/2/20211124201505/resumo_executivo_tic_domicilios_2020.pdf

² <https://insights.stackoverflow.com/survey/2021>

³ <https://www.hackerrank.com/research/student-developer/2018#skills>

mobile utilizando React Native, que utiliza os mesmos conceitos e design. Tecnologia que cresce no foco do desenvolvimento multiplataforma.⁴

O ReactJs melhora, de maneira eficiente, o processo de atualização do DOM (Document Object Model), permitindo a construção do Virtual DOM e a hospedagem na memória. O sistema impede que o DOM real faça atualizações constantemente e a velocidade da sua aplicação não será interrompida.

O ReactJs cria uma interface que pode ser acessada em diversos motores de busca, melhora o processamento da aplicação, e os resultados, culminando na otimização de SEO (Search Engine Optimization).⁵

Este curso visa ao desenvolvimento da competência: Construir interfaces com recursos da biblioteca ReactJs, com a intenção de expandir os conhecimentos de JavaScript com o uso de bibliotecas e frameworks, otimizando o desenvolvimento de soluções web através do uso de componentes prontos disponíveis no ReactJs tanto para estruturação de páginas quanto da manipulação de dados.

Versões

O histórico completo de lançamentos do React está disponível no GitHub. A documentação mais recente pode ser encontrada em <https://pt-br.reactjs.org/versions/>.

Instalação

Para utilizar o **React** é necessário ter instalado o Node.js, os módulos do Yarn do Facebook, que são a base da Biblioteca e ambiente de desenvolvimento (IDE), dentre as várias opções utilizaremos para este curso o **Microsoft VS Code** – Ambiente desenvolvimento pela Microsoft que apresenta diversos recursos avançados e é gratuito, no desenvolvimento do curso.

É recomendável utilizar uma máquina virtual devido a instalação e configuração de vários softwares e serviços.

⁴ <https://medium.com/reactbrasil/5-motivos-para-aprender-react-2bb755c96304>

⁵ <https://www.hostinger.com.br/tutoriais/o-que-e-react-javascript#:~:text=O%20React%20%C3%A9%20um%20biblioteca%20mais%20popular%20do,comandos%20usando%20um%20novo%20m%C3%A9todo%20de%20renderizar%20sites.>

CAPÍTULO 1 – O AMBIENTE DE TRABALHO

Neste capítulo você vai instalar os softwares para trabalhar com ReactJs e a IDE.

Objetivos:

- Preparar o ambiente de trabalho instalando o Node.js IDE em um ambiente Windows.
- Instalar e Configurar o Visual Studio Code (VS Code);
- Acessar o terminal no VS Code e verificar a versão instalada.
- Renderizar na tela “Olá mundo!”, para cumprir a tradição dos programadores ao iniciar o estudo de uma linguagem;

Atividade 1 – Preparar o ambiente de trabalho com Node.js

Objetivos:

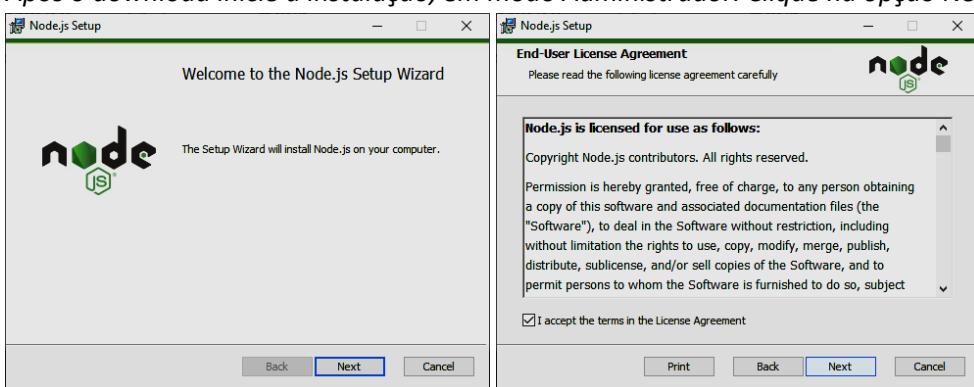
- Preparar o ambiente de trabalho instalando o Node.js IDE em um ambiente Windows.

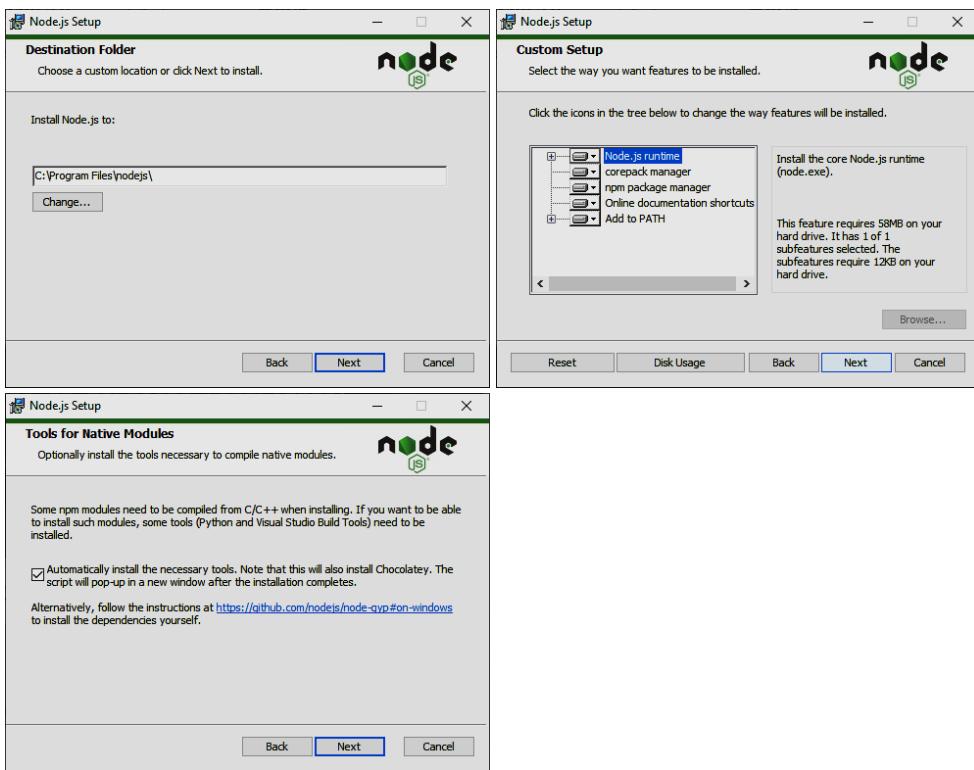
Instalando o Node.js

1. *Para utilizarmos o ReactJs, precisamos instalar o Node.js, que embora, não iremos utilizar serve de base para a biblioteca do React, abra o site: <https://nodejs.org/pt-br/>;*
2. *Clique no botão Downloads e escolha a versão mais atual do Node.js.*



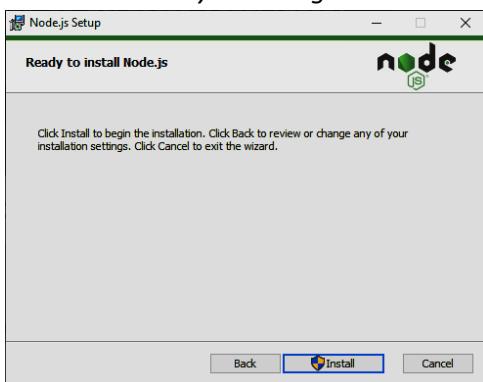
3. *Após o download inicie a instalação, em modo Administrador. Clique na opção Next.*



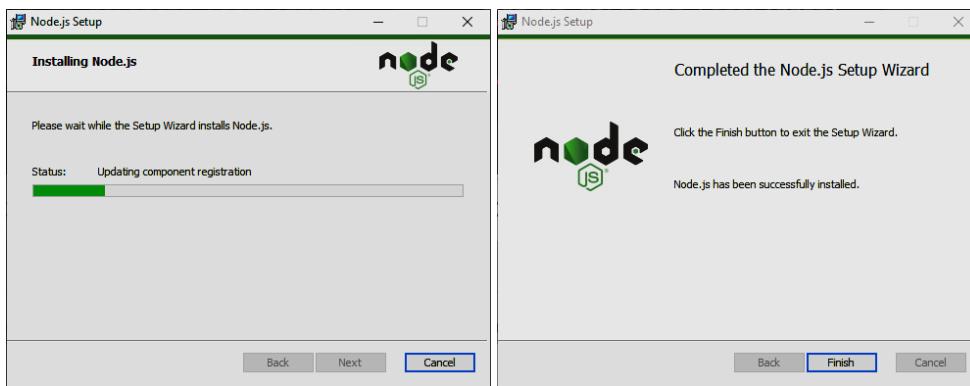


4. Adicione a instalação automática das ferramentas necessárias.

- Módulos nativos precisam ser compilados para funcionar. Fazer isso na instalação, traz benefícios de desempenho e atualizações do próprio Windows para funcionamento de linguagens, não sendo um conceito exclusivo de Javascript ou Node.js. Outras linguagens como Ruby e Python também possuem "módulos" que envolvem a compilação de código nativo para funcionar.
- O Node.js usa o Chocolatey para gerenciar sua cadeia de ferramentas e já possui pacotes disponíveis, isso reduz a sobrecarga na instalação de uma ampla variedade de utilitários. Além disso, pode ser instalado em todo o sistema ou apenas para uso de uma aplicação específica. Caso não utilize, terá que gerenciar ferramentas nativas por conta própria.
- O Chocolatey torna o gerenciamento de pacotes muito mais fácil no Windows.



5. A instalação do Node será realizada e em seguida as ferramentas nativas.



6. A instalação das ferramentas nativas, ocorre no CMD e em seguida no PowerShell e não é necessária nenhuma interação, apenas aguarde o processo, **QUE COSTUMA SER DEMORADO:**
- Caso o Windows não tenha sido atualizado recentemente, nesse momento serão realizadas várias atualizações,
 - é possível que ocorra algum erro devido as permissões de administrador, abordaremos formas de correção no próximo item.

```
Install Additional Tools for Node.js
=====
Tools for Node.js Native Modules Installation Script

This script will install Python and the Visual Studio Build Tools, necessary
to compile Node.js native modules. Note that Chocolatey and required Windows
updates will also be installed.

This will require about 3 Gb of free disk space, plus any space necessary to
install Windows updates. This will take a while to run.

Please close all open programs for the duration of the installation. If the
installation fails, please ensure Windows is fully updated, reboot your
computer and try to run this again. This script can be found in the
Start menu under Node.js.

You can close this window to stop now. Detailed instructions to install these
tools manually are available at https://github.com/nodejs/node-gyp#on-windows

Pressione qualquer tecla para continuar. . .

[Output continues with log messages from the script, includingChocolatey package installations and file upgrades.]
```

```
Using this script downloads third party software
=====
This script will direct to Chocolatey to install packages. By using
Chocolatey to install a package, you are accepting the license for the
application, executable(s), or other artifacts delivered to your machine as a
result of a Chocolatey install. This acceptance occurs whether you know the
license terms or not. Read and understand the license terms of the packages
being installed and their dependencies prior to installation:
- https://chocolatey.org/packages/chocolatey
- https://chocolatey.org/packages/python
- https://chocolatey.org/packages/visualstudio2019-workload-vctools

This script is provided AS-IS without any warranties of any kind
...
Chocolatey has implemented security safeguards in their process to help
protect the community from malicious or pirated software, but any use of this
script is at your own risk. Please read the Chocolatey's legal terms of use
as well as how the community repository for Chocolatey.org is maintained.

Pressione qualquer tecla para continuar. . .
```

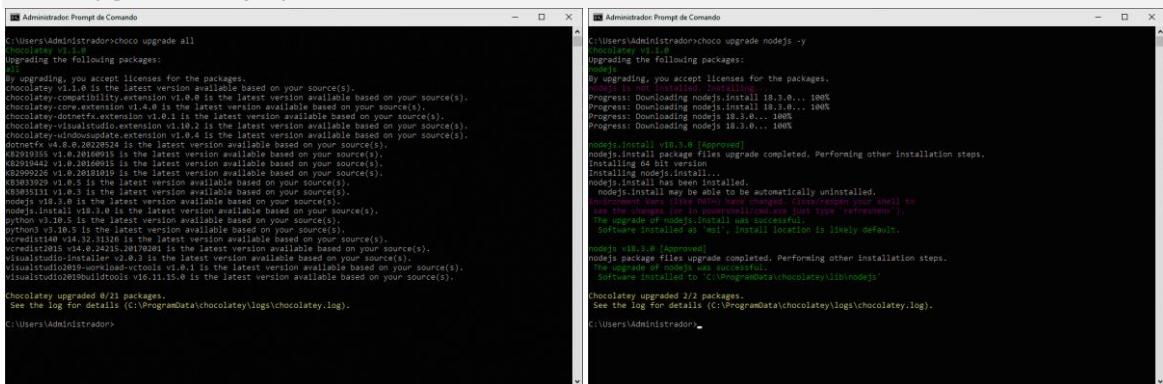
```
[Output continues with log messages from the Chocolatey package manager, showing the upgrade of vcredist140 and other package installations.]
```

7. Ao finalizar o processo basta pressionar **Enter** e a janela será fechada automaticamente. Será necessário reiniciar manualmente o computador para finalizar a instalação.
8. Caso ocorra erro durante a instalação, como mostrado abaixo, será necessário executar o **Windows PowerShell** ou **CMD**, como administrador para corrigir a instalação, também é recomendável executar os comandos para instalar possíveis atualizações:



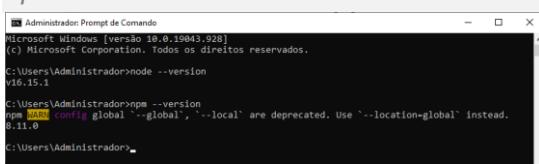
9. Execute os seguintes comandos:

```
choco upgrade all
choco upgrade nodejs -y
```



10. Vamos verificar se o Node e o Npm estão instalados. Execute o **CMD**, como administrador e digite o seguinte comando:

```
node --version
npm --version
```



11. Quando instalamos o Node.js o NPM também é instalado no computador como parte do mesmo pacote, para utilizá-lo basta abrir a ferramenta de linha de comando e digitar **npm**, uma lista com os comandos será exibida.

- a. Através do NPM iremos instalar o gerenciador de dependências do Facebook Yarn, que é mais recomendado para se trabalhar usando o React.
- b. Execute no **CMD** como administrador, os seguintes comandos:

O primeiro para atualizar o **npm**, e o segundo para instalar as dependências do **Yarn**:

```
npm install -g npm@latest
npm install -g yarn
```

```
C:\ Administrador: Prompt de Comando

C:\Users\Administrador>npm install -g npm@0.12.1
npm WARN config global --global, --local are deprecated. Use '--location=global' instead.
npm WARN config global --global, --local are deprecated. Use '--location=global' instead.

added 1 package, and audited 202 packages in 4s

11 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

C:\Users\Administrador>npm --version
npm WARN config global --global, --local are deprecated. Use '--location=global' instead.
0.12.1

C:\Users\Administrador>npm install -g npm@latest
npm WARN config global --global, --local are deprecated. Use '--location=global' instead.

changed 14 packages, and audited 202 packages in 4s

11 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

C:\Users\Administrador>npm --version
npm WARN config global --global, --local are deprecated. Use '--location=global' instead.
0.12.1

C:\Users\Administrador>npm install -g yarn
npm WARN config global --global, --local are deprecated. Use '--location=global' instead.

added 1 package, and audited 2 packages in 747ms

found 0 vulnerabilities

C:\Users\Administrador>
```

12. Pronto o Node.js, npm, Facebook Yarn e suas dependências já estão instaladas, recomenda-se nesse momento reiniciar o computador.

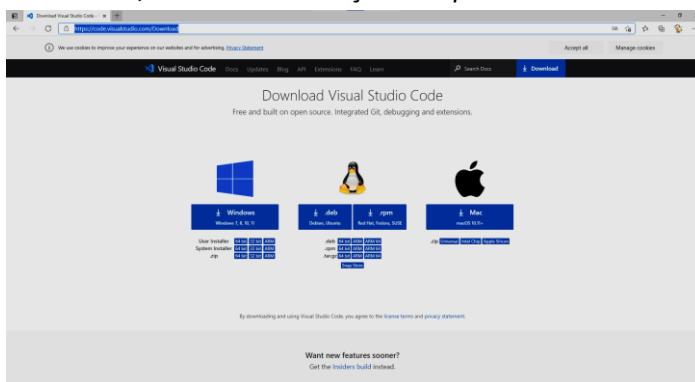
Atividade 2 – Instalar e configurar o Visual Studio Code

Objetivos:

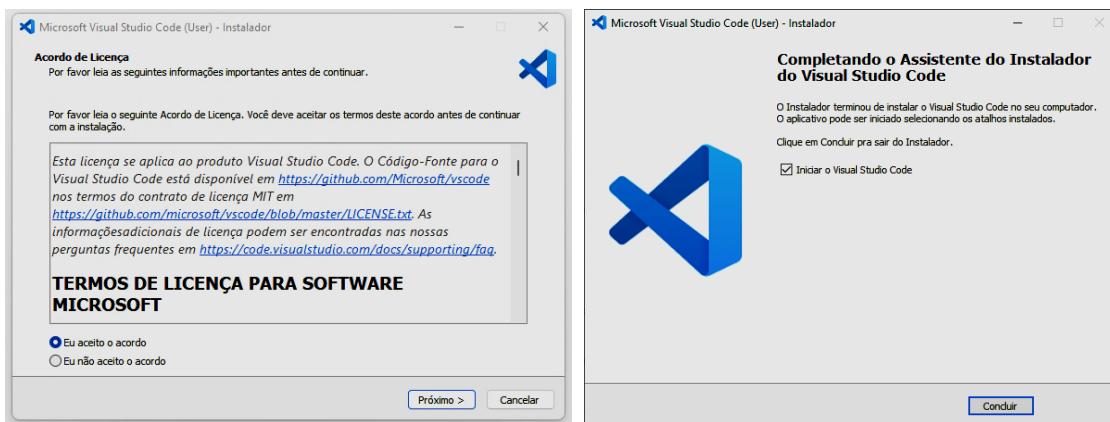
- Instalar e Configurar o Visual Studio Code (VS Code);

Vamos instalar o IDE VS Code e as extensões que alteram o idioma e implementam funcionalidades e ferramentas para ajudar na linguagem de programação que utilizaremos.

1. Acesse <https://code.visualstudio.com/Download> para realizar o Download da versão mais atual do VS Code; Escolha a instalação adequada ao seu sistema operacional.



- 2. Após o download, inicie a instalação. Basta seguir os passos de instalação.*
 - 3. Marque a opção “Eu aceito o acordo” e continue a instalação, nenhuma outra opção é necessário ser alterada.*
 - a. A Instalação do software é feita em inglês, mas é possível instalar a extensão em português.*

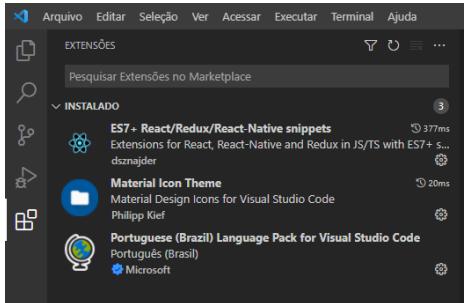


4. Abra o VS Code e na barra lateral procure o botão de Extensão ou pelo menu **Ver / Extensões** (View/Extension).

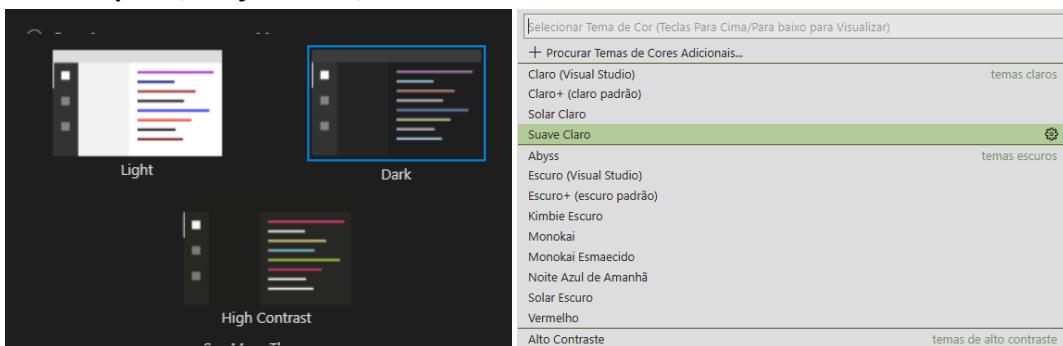


5. Para instalar as extensões basta realizar a pesquisa e clicar em instalar, faça a instalação destas 3 extensões:

- Portuguese (Brazil) Language Pack for Visual Studio Code* que altera o idioma do VS Code , será necessário reiniciar o programa para alterar o idioma;
- ES7+ React/Redux/React-Native snippets* inclui ferramentas para o desenvolvimento em React, algumas extensões extras ligadas a linguagem também serão instaladas;
- Material Icon Theme* que facilita a identificação dos arquivos do projeto.



6. É possível alterar as cores do ambiente de desenvolvimento, logo após a instalação ou usando o menu **Arquivo / Preferências / Tema de Cores**.



7. Com isso finalizamos a instalação e configuração do VsCode;

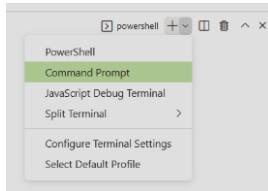
Atividade 3 – Acessar o Terminal no VS Code e criar app ReactJs

Objetivos:

- Acessar o terminal no VS Code;
- Verificar a versão instalada.

Antes de partir para o desenvolvimento da linguagem é preciso verificar se está tudo funcionando para criar o app .

1. *Abra o VSCode, em modo Administrador e utilize o menu Terminal / Novo Terminal ou CTRL+ ‘ para acessar o terminal;*
2. *Utilize preferencialmente o Prompt de Comando; no PowerShell é possível que ocorra erros ou não funcione alguns comandos*



3. *Com o ambiente devidamente configurado, criamos nosso primeiro projeto em React. Para isso vamos:*
 - a. *Instalar a ferramenta de linha de comando ‘create-react-app’, que cria um projeto do zero sem bundling, otimização de arquivo e outros detalhes de configuração que podem ser extensos quando realizados manualmente.*
 - b. *Para instalar o create-react-app digite o comando:*

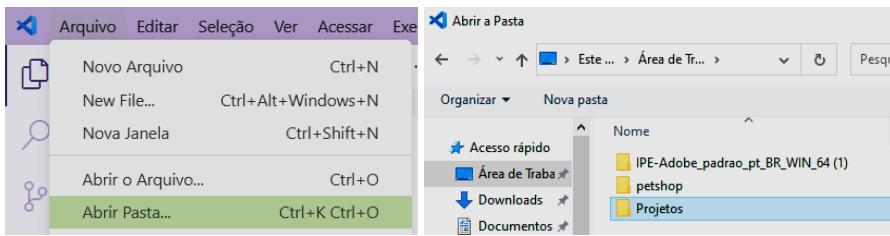
```
npm install -g create-react-app
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\Administrador\Desktop\petshop> npm install -g create-react-app
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.
npm WARN deprecated tar@2.2.2: This version of tar is no longer supported, and will not receive security updates. Please upgrade asap.
added 67 packages, and audited 68 packages in 3s
4 packages are looking for funding
  run `npm fund` for details
2 high severity vulnerabilities

Some issues need review, and may require choosing
a different dependency.

Run `npm audit` for details.
PS C:\Users\Administrador\Desktop\petshop>
```

4. *Quando a instalação terminar, partir desse comando podemos criar o projeto da aplicação. Crie uma pasta chamada ‘Projetos’ em seu computador, no VSCode acesse o menu Arquivo > Abrir Pasta e selecione a pasta:*



5. Vamos criar a estrutura de pastas do projeto 'petshop', digite o comando:

```
create-react-app petshop
```

a. A criação da estrutura irá demorar um certo tempo, pois compila todos os módulos.

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Microsoft Windows [versão 10.0.19043.928]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\luis\Downloads\Desktop\Projetos>create-react-app petshop

Creating a new React app in C:\Users\luis\Downloads\Desktop\Projetos\petshop.

npm WARN config global '-global', '-local' are deprecated. Use '--location-global' instead.
Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...
npm WARN config global '-global', '-local' are deprecated. Use '--location-global' instead.
added 1383 packages in 2m

192 packages are looking for funding
  run `npm fund` for details
Gits repos not initialized. Error: Command failed: git --version
  at checkSync (C:\Users\luis\Downloads\Desktop\Projetos\node_modules\@react-native-community\cli-platform-android\lib\checkSync.js:11:11)
  at execSync (node:internal:child_process:517:11)
  at execSync (C:\Users\luis\Downloads\Desktop\Projetos\petshop\node_modules\react-scripts\scripts\init.js:46:5)
  at module.exports (C:\Users\luis\Downloads\Desktop\Projetos\petshop\node_modules\react-scripts\scripts\init.js:276:7)
  at (C:\Users\luis\Downloads\Desktop\Projetos\petshop\node_modules\react-scripts\scripts\init.js:11:14)
  at Script.runInThisContext (node:vm:129:12)
  at Object.runInThisContext (node:vm:305:38)
  at node:internal/process/execution:76:19
  at (C:\Users\luis\Downloads\Desktop\Projetos\petshop\node_modules\react-scripts\scripts\init.js:6:22)
  at start (C:\Users\luis\Downloads\Desktop\Projetos\petshop\node_modules\react-scripts\scripts\init.js:1:1)
  signal: null,
  output: [ null, null, null ],
  pid: 21804,
  stdbuf: null,
  stderr: null
}

Installing template dependencies using npm...
npm WARN config global '-global', '-local' are deprecated. Use '--location-global' instead.
npm WARN deprecated source-map-resolve@0.6.0: See https://github.com/lydell/source-map-resolve#deprecated
added 39 packages in 7s

```

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Removing template package using npm...
npm WARN config global '-global', '-local' are deprecated. Use '--location-global' instead.
removed 1 package, and audited 1422 packages in 3s

192 packages are looking for funding
  run `npm fund` for details

6 high severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

Run 'npm audit' for details.

Success! Created petshop at C:\Users\luis\Downloads\Desktop\Projetos\petshop
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 petshop
  npm start

Happy hacking!

C:\Users\luis\Downloads\Desktop\Projetos>

```

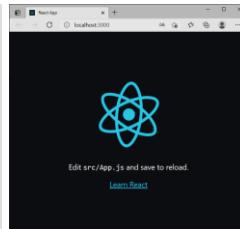
6. Após a instalação acesse a pasta do petshop pelo terminal

```
cd petshop
```

7. Digite o comando abaixo para executar a aplicação no navegador.

```
yarn start
```

8. O projeto será acessado na porta 3000 e o navegador padrão do sistema operacional irá abrir a página da aplicação automaticamente, mostrando a tela inicial padrão do React.



9. Toda fase instalação foi realizada, sua estrutura de projeto já está disponível. Vamos as ações práticas nas próximas atividades.

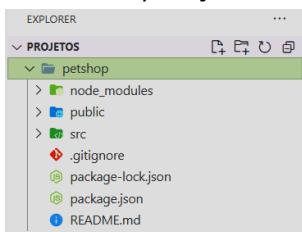
Atividade 4 – “Olá Mundo!” para cumprir a tradição

Objetivos:

- Renderizar na tela “Olá mundo!”, para cumprir a tradição dos programadores ao iniciar o estudo de uma linguagem;

Sempre que vamos aprender uma nova linguagem a primeira atividade em todos os cursos que conhecemos é fazer um “Olá mundo!”; é uma tradição entre os programadores e vamos fazer isso agora:

1. Acesse o diretório do projeto, você irá encontrar três pastas, com arquivos que são utilizados pelo React:
 - a. **node_modules** - Guarda todas as dependências do projeto. Criada pelo NPM ou pelo Yarn no momento em que uma biblioteca é adicionada ao projeto.
 - b. **public** - Pasta pública da aplicação web. Ficam os arquivos iniciais da aplicação, como o index.html, o ícone do site e um arquivo JSON com os parâmetros de configuração.
 - c. **src** - Código fonte da aplicação: Estão disponíveis todos os arquivos JavaScript que compõem a nossa aplicação.



2. No VSCode Abra a pasta do 'projeto/petshop/src' com o código fonte da aplicação. Abra o arquivo 'App.js' para alterar o código da seguinte forma:

```
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <h1> Hello World! </h1>
      </header>
    </div>
  );
}

export default App;
```

3. O Resultado visto no navegador será o mostrado abaixo:



Atividade Extra

Objetivos:

- Rever e aprimorar os conhecimentos construídos.

Vamos trabalhar algumas atividades extras para rever e aprimorar o que trabalhamos até aqui:

1. *Uma boa prática recomendada pelo Wellington Vieira dos Santos, do Senac Itaquera, é a integração com o Git, para saber mais acesse o vídeo:*

<https://www.youtube.com/watch?v=peGUkhXD3Vw>

CAPÍTULO 2 – CRIANDO E NAVEGANDO NA SPA COM REACT

Neste capítulo você vai aprender mais como criar uma Single Page Application (SPA), em React, melhorar aspectos de aparência com CSS e estabelecer rotas.

Objetivos:

- Entender o funcionamento do React e sua estrutura;
- Criar as primeiras páginas e componentes.
- Utilizar variáveis para interagir com CSS em aplicação React;
- Utilizar componentes específicos para aplicação do CSS;
- Dividir a SPA em diferentes rotas para cada componente;

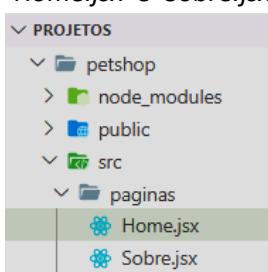
Atividade 1 – Criando páginas com ReactJs

Objetivos:

- Entender o funcionamento do React e sua estrutura;
- Criar as primeiras páginas e componentes.

Vamos criar as primeiras páginas e componentes em ReactJs.

1. *Vamos entender como o React renderiza as páginas Web, e como essa estrutura é criada, para isso vamos criar dentro do diretório ‘petshop/src’, uma pasta chamada ‘páginas’, e criamos os arquivos ‘Home.jsx’ e ‘Sobre.jsx’:*



2. *Dentro de ‘Home.jsx’ importamos o módulo do React, padrão para nosso trabalho, que traz as referências globais da biblioteca, e em seguida criamos uma função ‘Home’, sem elementos de propriedade, com um ‘return’ para atribuir o código HTML a ser utilizado e um ‘export’ para renderização posterior;*

```
import React from "react";  
  
const Home = () => {  
  return(  
    )  
}
```

```
}

export default Home
```

3. Insira o código HTML abaixo para compor a página 'Home.jsx':

```
<main>
  <div>
    <h2>Pet notícias</h2>
  </div>
  <section>
    <article>
      <h2>Banho no cão</h2>
      <p>Banhos regulares mantém os pelos e pele de seu cachorro limpa, renovada e saudável. O processo de ensaboar e escovar remove resíduos de pelos e peles que já estão mortas, providenciando um maior respiro da pele de seu cachorro</p>
    </article>
  </section>
</main>
```

4. Repita o processo no arquivo 'Sobre.jsx':

```
import React from "react";

const Sobre = () => {
  return(
    )
}

export default Sobre
```

5. Insira o código HTML abaixo para compor a página 'Sobre.jsx':

```
<main>
  <div>
    <h2>Sobre Meu amigo cão</h2>
  </div>
  <section>
    <article>
      Na Meu amigo cão, o nosso sucesso como uma organização é conduzida pelo cuidado que temos com nossos animais!!!
    </article>
  </section>
</main>
```

6. Para renderizar a página no navegador, acesse o arquivo 'petshop/src/App.js', e insira o seguinte código, inicialmente importando o 'Home' que foi criado, depois exportando no retorno da função do App, é importante o entendimento que o Home, é um componente, e será inserido do seguinte modo:

```
import logo from './logo.svg';
import './App.css';
import Home from './paginas/Home';
```

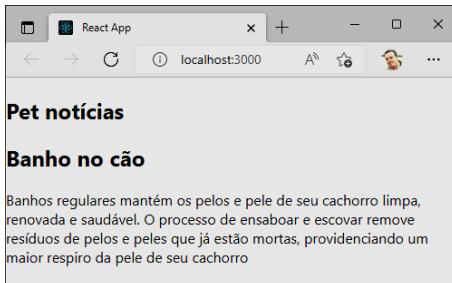
```

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

export default App;

```

7. Inicie o servidor acessando a pasta petshop e iniciando o servidor com 'npm start'. Observe o resultado após a renderização:



8. Vamos incluir a página 'Sobre', porém para renderizar mais de um arquivo é preciso, encapsular o JSX, abrindo e fechando tag no entorno das páginas a serem renderizadas:

```

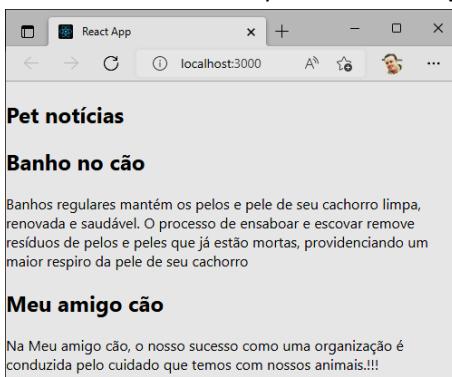
import './App.css';
import Home from './paginas/Home';
import Sobre from './paginas/Sobre';

function App() {
  return (
    <>
      <Home />
      <Sobre />
    </>
  );
}

export default App;

```

9. Observe o resultado após a renderização:



10. Finalizamos a construção de nossa primeira Single Page Application (SPA).

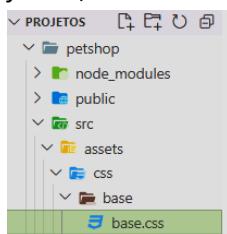
Atividade 2 – CSS base utilizando variáveis

Objetivos:

- Utilizar variáveis para interagir com CSS em aplicação React;

O próximo objetivo é utilizar o CSS para melhorar a aparência das páginas que carregamos.

1. *Vamos criar dentro do diretório ‘src’ uma pasta chamada ‘assets’ que irá armazenar todos os conteúdos estáticos das páginas, como imagens e códigos css, por isso já estruturamos desta forma, criando o arquivo ‘base.css’:*



2. *Dentro do diretório chamado ‘base’, copie do material o arquivo ‘cap02_arq_trabalho/_reset.css’, para pasta base e importe o código dentro o arquivo ‘base.css’:*

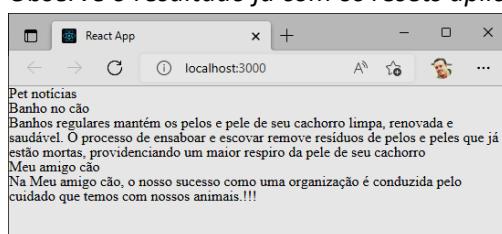
```
@import url(_reset.css);
```

3. *Para fazer com que o CSS seja aplicado a página precisamos importar o arquivo para a renderização do app. Abra o arquivo App.js, e realize a importação:*

```
import './App.css';
import Home from './paginas/Home';
import Sobre from './paginas/Sobre';
import './assets/css/base/base.css';

function App() {
  ...
  export default App;
```

4. *Observe o resultado já com os resets aplicados;*



5. *Voltamos a pasta ‘base’ e criamos um CSS ‘diferente’, que na verdade estabelece variáveis e valores, a serem utilizados em outros elementos e delimitando todos os parâmetros utilizados no projeto, crie um arquivo _variaveis.css e insira o código:*

```
:root {
  --fonte-familia-corpo: 'Montserrat', sans-serif;
  --fonte-familia-logo: 'Pacifico', cursive;
```

```

--fonte-familia-titulo-pagina: 'Pacifico', cursive;

--fonte-peso-input: 300;
--fonte-peso-cartao: 500;
--fonte-peso-categorias: 500;
--fonte-peso-menu-cabecalho-item: 500;

--fonte-tamanho-logo: 1.75rem;
--fonte-tamanho-cartao-titulo: 1.75rem;
--fonte-tamanho-cartao-post-titulo: 1.375rem;
--fonte-tamanho-input-label: 1.25rem;
--fonte-tamanho-botao: 1.375rem;
--fonte-tamanho-formulario-fieldset: 1.375rem;
--fonte-tamanho-tabela-cabecalho: 1.375rem;
--fonte-tamanho-titulo-pagina: 1.75rem;
--fonte-tamanho-menu-cabecalho-item: 1.2rem;
--fonte-tamanho-naoencontrado-texto: 1.25rem;

--cor-primaria: #0071ea;
--cor-secundaria: #d6eaff;
--cor-contraste-escuro: #4D4D4D;
--cor-contraste-claro: #fff;
--cor-divisao: #dadada;
--cor-aviso: #df2525;
--cor-sucesso: #46bb42;

--cor-categoria-bem-estar: #ff4949;
--cor-categoria-comportamento: #368dff;

--sombra: 0 5px 10px #55a6ff38;
}

@media(min-width: 800px) {
  :root {
    --fonte-tamanho-logo: 2.5rem;
    --fonte-tamanho-menu-cabecalho-item: 1rem;
  }
}

```

6. Isso não irá afetar a nossa página nesse momento, afinal não utilizamos esses parâmetros ainda. Vamos também importar o código dentro o arquivo 'base.css' e aproveitamos para importar uma fonte do Google para utilização no projeto:

```

@import url(_reset.css);
@import url(_variaveis.css);
@import
url('https://fonts.googleapis.com/css2?family=Montserrat:wght@300;400;500&family=Pacifico&display=swap');

```

7. Dando continuidade do código do arquivo 'base.css', vamos inserir dentro do CSS as variáveis que criamos, e sempre que precisarmos utilizar ou modificar os mesmos elementos não precisamos repetir o código e sim apenas carregar a variável:

```

body {
  background-color: var(--cor-secundaria);
}

```

```

        color: var(--cor-contraste-escuro);
        font-family: var(--fonte-familia-corpo);
    }

.container {
    padding-right: 1rem;
    padding-left: 1rem;
}

.flex {
    display: flex;
}

.flex--centro {
    align-items: center;
    justify-content: center;
}

.flex--coluna {
    flex-direction: column;
}

.titulo-pagina {
    display: flex;
    align-items: center;
    font-family: var(--fonte-familia-titulo-pagina);
    font-size: var(--fonte-tamanho-titulo-pagina);
    background-color: var(--cor-contrasteclaro);
    border-radius: 10px;
    box-shadow: var(--sombra);
    height: 4.75rem;
    margin-top: 1.25rem;
    margin-bottom: 2rem;
    padding-left: 2rem;
}

@media(min-width: 800px) {
    .container {
        padding-right: 2.5rem;
        padding-left: 2.5rem;
    }
}

@media(min-width: 1200px) {
    .container {
        padding-left: calc((100vw - 900px)/2);
        padding-right: calc((100vw - 900px)/2);
    }
}

```

8. Para que a página renderize o CSS, precisamos aplicar as classes aos componentes, Home e Sobre, observe que aqui para não haver confusão com as classes do HTML, utilizamos o parâmetro ‘className’:

```
<div className="container">
```

```

<h2 className="titulo-pagina">Pet notícias</h2>
</div>
<section className="container flex flex--centro">

```

9. Observe o resultado com os elementos CSS aplicados:



10. Com isso finalizamos mais uma atividade;

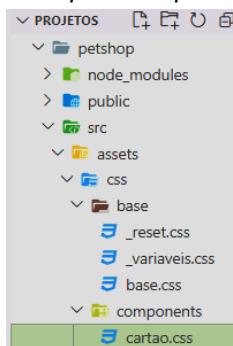
Atividade 3 – React Components para CSS específico

Objetivos:

- Utilizar componentes específicos para aplicação do CSS;

Vamos criar um CSS Específico para o nosso app, utilizando o conceito de componentes.

1. Vamos criar dentro do diretório 'css' uma pasta chamada 'components' que irá armazenar componentes específicos. Ao criar a pasta copie do material o arquivo 'cap02_arq_trabalho/cartao.css';
a. Aproveite para revisar os conceitos trabalhados nas atividades anteriores;



2. Para que seja aplicada uma formatação diferente as postagens, que importamos o componente 'cartao.css', e aplicamos as classes aos elementos específicos:

```

import React from "react";
import './assets/css/components/cartao.css';

const Sobre = () => {
  ...
    <section className="container flex flex--centro">

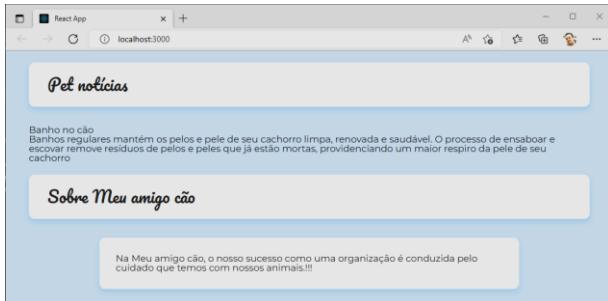
```

```

<article className="cartao">
    Na Meu amigo cão, o nosso sucesso como uma organização é
    conduzida pelo cuidado que temos com nossos animais!!!
</article>
...
export default Sobre

```

3. Observe o resultado com os elementos CSS aplicados:



4. Uma vez que é renderizado, um componente, pode ser utilizado por todos os outros que compõem a mesma página, por isso podemos aplicar as classes no componente Home, sem a necessidade de realizar a importação novamente:

```

import React from "react";

const Home = () => {
    return(
        ...
        <section className="container flex flex--centro">
            <article className="cartao post">
                <h2 className="cartao__titulo">Banco no cão</h2>
                <p className="cartao__texto">Banhos ... cachorro</p>
            </article>
        </section>
        ...
    )
}

export default Home

```

5. Finalizamos a criação de React Componentes específicos para o CSS, abordaremos novamente essa questão no futuro.



Atividade 4 – Uso de Rotas no ReactJs

Objetivos:

- Dividir a SPA em diferentes rotas para cada componente;

Embora nosso projeto esteja em uma única página, são dois assuntos diferentes, então vamos dividir os componentes através do uso de rotas.

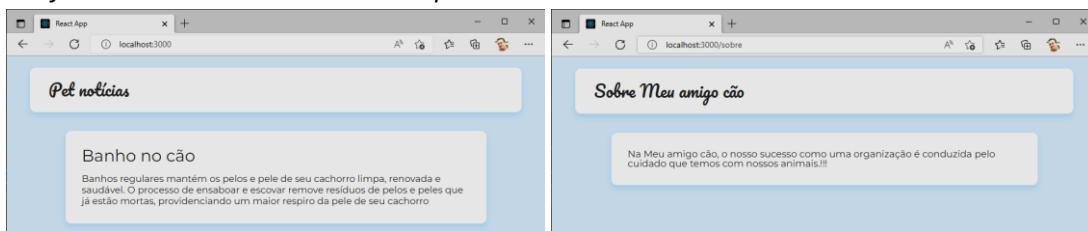
1. Abra o arquivo 'App.js', para inserirmos uma função para estabelecer rotas para o carregamento dos componentes; Fique atento aos passos e aos elementos do código:
 - a. Utilizamos aqui o 'window.location.pathname' para capturar a URL da aplicação;
 - b. Estabeleça uma condição para o retorno de um determinado componente;
 - c. Retorne dentro da função do App, a função de rota.

```
import './App.css';
import Home from './paginas/Home';
import Sobre from './paginas/Sobre';
import './assets/css/base/base.css';

function App() {
  const Router = () => {
    const location = window.location.pathname
    if(location === '/sobre'){
      return <Sobre />
    } else {
      return <Home />
    }
  }
  return (
    <>
      { Router()}
    </>
  );
}

export default App;
```

2. Conforme o URL visitado será o componente visualizado:



3. O objetivo foi atingido, porém se houver um número elevado de rotas, esse não é um caminho viável, então vamos utilizar uma biblioteca para otimizar o elementos de rota, sem uso do 'if'. Para instalar o 'React Router Dom' abra o terminal, com 'CTRL+j', pare o serviço do servidor utilizando 'CTRL+c' e utilize o seguinte comando:

- a. Se desejar mais informações sobre bibliotecas acesse <https://www.npmjs.com/>;
npm i react-router-dom@5

```

PROBLEMAS SAÍDA TERMINAL CONSOLE DE DEPURAÇÃO

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

webpack compiled successfully
Deseja finalizar o arquivo em lotes (S/N)? s

C:\Projetos\petshop>npm install -g react-router-dom@5
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.

changed 19 packages, and audited 20 packages in 4s

found 0 vulnerabilities

C:\Projetos\petshop>

```

4. Inicie o servidor novamente, com o comando:

```
npm start
```

5. Após a instalação, Importe as bibliotecas no arquivo 'App.js' e faça os ajustes de maneira a simplificar as rotas, como no código abaixo:

```

import './App.css';
import Home from './paginas/Home';
import Sobre from './paginas/Sobre';
import './assets/css/base/base.css';
import {BrowserRouter as Router, Route, Switch} from 'react-router-dom';

function App() {
  return (
    <Router>
      <Switch>
        <Route exact path='/'>
          <Home />
        </Route>
        <Route path='/sobre'>
          <Sobre />
        </Route>
      </Switch>
    </Router>
  )
}

export default App

```

Atividade Extra

Objetivos:

- Rever e aprimorar os conhecimentos construídos.

Vamos trabalhar algumas atividades extra para rever e aprimorar o que trabalhamos *até aqui*:

1. Faça a sugestão aos alunos que desenvolvam mais uma página para compor o app;

CAPÍTULO 3 – APERFEIÇOANDO A SPA

Neste capítulo você vai aprofundar o conceito de Single Page Application (SPA), orientar todas as rotas inexistentes para uma página de aviso, diferenciar sobre as formas de carregamento tradicional, também vai criar o cabeçalho e a barra de navegação do projeto, e integrar aos outros componentes, finalizando com a otimização do carregamento das páginas do PetShop.

Objetivos:

- Indicar a inexistência de uma rota.
- Entender as diferenças entre métodos de carregamento tradicional e SPA;
- Criar cabeçalho e barra de navegação.
- Integrar dois componentes a mesma página, e em todo o projeto;
- Otimizar o carregamento da SPA através de Link do React Router.

Atividade 1 – Criando a página de erro 404

Objetivos:

- Indicar a inexistência de uma rota.

Quando o usuário por algum motivo insere na URL um endereço inexistente, ou ocorre falha na navegação o navegador exibe a mensagem de erro 404, dizendo que a página não existe. Iremos cria um componente para personalizar a página e indicar que toda as rotas inexistente sejam direcionadas para ela.

1. *Inicie a atividade , copiando do material o arquivo ‘cap03_arq_trabalho/404.css’, para o diretório do projeto, ‘petshop/src/assets/css/404.css’, para a depois importar dentro o arquivo que criaremos a seguir;*

2. *Dentro do diretório ‘petshop/src/paginas’, criamos o arquivo ‘Pagina404.jsx’, import o React, o CSS, e crie o componente, conforme o código abaixo:*

```
import React from "react";
import './assets/css/404.css'

const Pagina404 = () => {
  return(
    )
}

export default Pagina404
```

3. Montamos a estrutura da página com a linguagem HTML, dentro do 'return', seguindo os mesmos padrões de CSS das outras páginas, para depois trabalharmos com os elementos do React, como segue:

```
import React from "react";
import '../assets/css/404.css'

const Pagina404 = () => {
  return(
    <main className="container flex flex--centro flex--coluna">
      <img className="catiorinho-imagem" src="" alt="Ilustração Catorinho" />
      <p className="naoencontrado-texto">
        Au, au, foi mau!
      </p>
      <p className="naoencontrado-texto">
        Página não encontrada!
      </p>
    </main>
  )
}

export default Pagina404
```

4. Para renderizar a página e direcionar todas as rotas que não são válidas para esse componente, abra o arquivo 'App.js', aproveite para organizar o arquivo, manter as importações gerais no início, logo abaixo os componentes importados, e inclua a rota para todas as páginas que não tiverem o seu caminho definido para que exibam a 'Pagina404', como segue no código abaixo:

a. Aqui usamos um caractere 'coringa' ou '*' (asterisco) que é interpretado como 'all', todos.

```
import React from 'react'
import './assets/css/base/base.css'
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom'

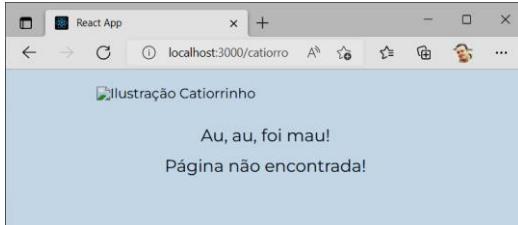
import Home from './paginas/Home';
import Sobre from './paginas/Sobre';
import Pagina404 from './paginas/Pagina404';

function App() {
  return (
    <Router>
      <Switch>
        <Route exact path='/'>
          <Home />
        </Route>
        <Route path='/sobre'>
          <Sobre />
        </Route>
        <Route path='*'>
          <Pagina404 />
        </Route>
      </Switch>
    </Router>
  )
}
```

```
}
```

```
export default App
```

5. *Digite na URL <http://localhost:3000/catiorro> e observe o resultado após a renderização:*

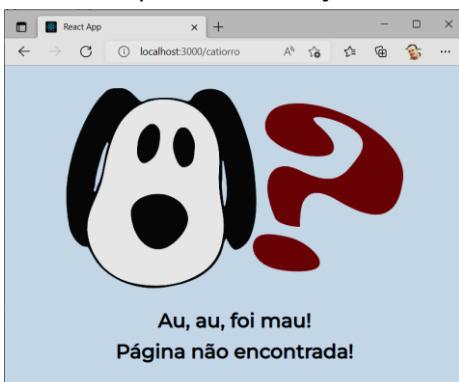


6. *Dando continuidade, crie uma pasta dentro do diretório 'assets', chamada 'img', copie do material o arquivo 'cap03_arq_trabalho/catiorrinho404.svg', para o diretório recém-criado, vamos importar para o componente 'Pagina404' e inserir no código, da seguinte forma:*
- Observe que importamos e inserimos o arquivo como função;*

```
import React from "react";
import '../assets/css/404.css'
import imagem from "../assets/img/catiorrinho404.svg";

const Pagina404 = () => {
  ...
    <img className="catiorrinho-imagem" src={imagem}
  alt="Ilustração Catiorrinho" />
  ...
}
```

7. *Inicie o servidor acessando a pasta petshop e iniciando o servidor com 'npm start'. Observe o resultado após a renderização:*



8. *Finalizamos a construção e roteamento da página 404.*

Atividade 2 – Carregamento Web Tradicional x SPA

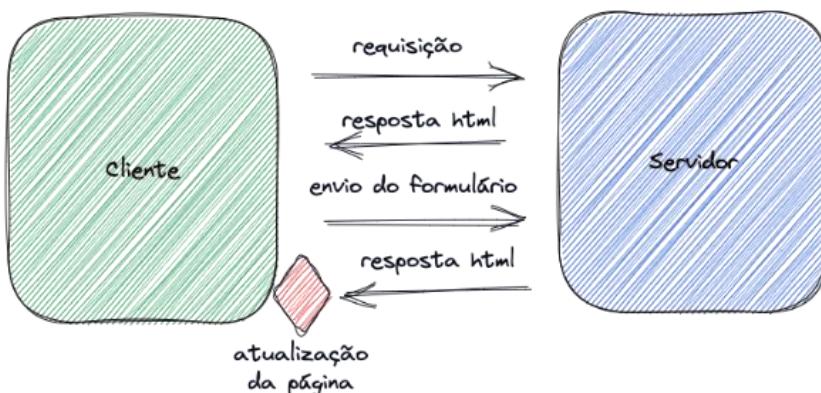
Objetivos:

- Entender as diferenças entre métodos de carregamento tradicional e SPA;

Antes de iniciar a próxima atividade vamos entender a diferença entre uma SPA e o carregamento de uma página tradicional.

Modelo Tradicional (Multi Page Application):

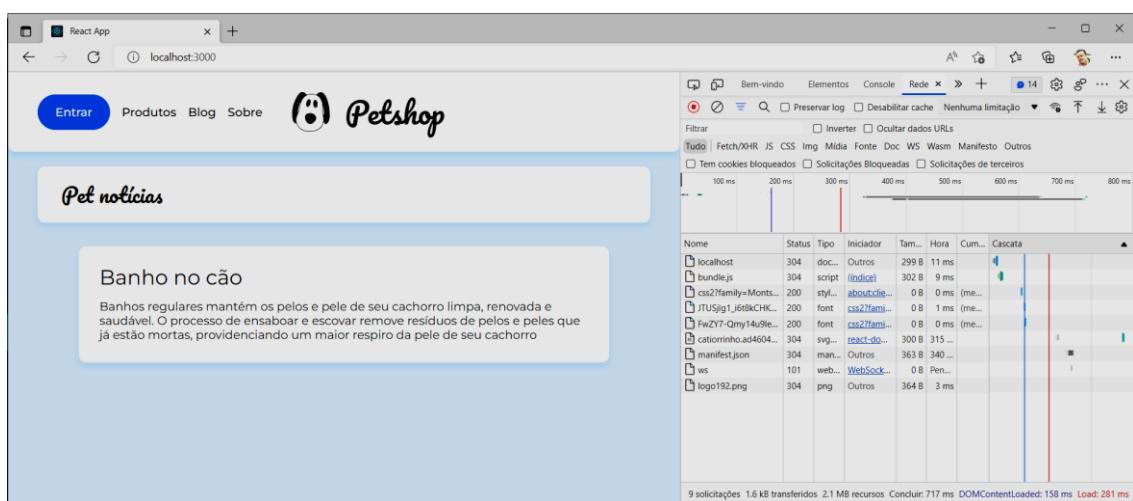
Ciclo de Vida de uma Página Tradicional



Ao abrir uma página, as informações são carregadas de algum lugar através de uma request.

Por exemplo, para exibir a informação do usuário logado em todas as páginas. A página é acessada, feita solicitação, banco de dados é consultado, resultado formatado e então exibido.

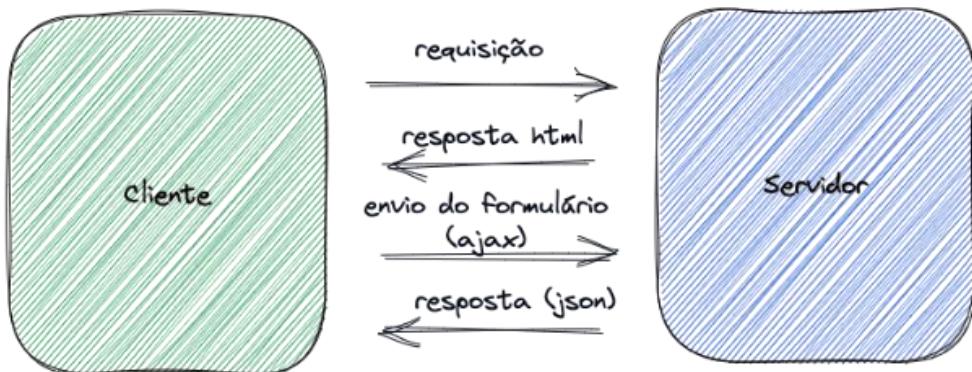
Para saber impacto que isso gera, abra o devtools do seu navegador (tecla F12, geralmente), vá até a guia Network e atualize a página. Cada item é uma request que foi processada. Se navegar para outra página, o fluxo é executado novamente para cada página.



Você está no celular, os dados serão recarregados a cada operação, consumindo tempo e seu pacote. Se é um software na nuvem, e tem limite de 30GB de transferência de dados mensais, irá consumir gradativamente com o fluxo de informações repetidas.

Single Page Application (SPA)

Ciclo de Vida de uma Single Page Application

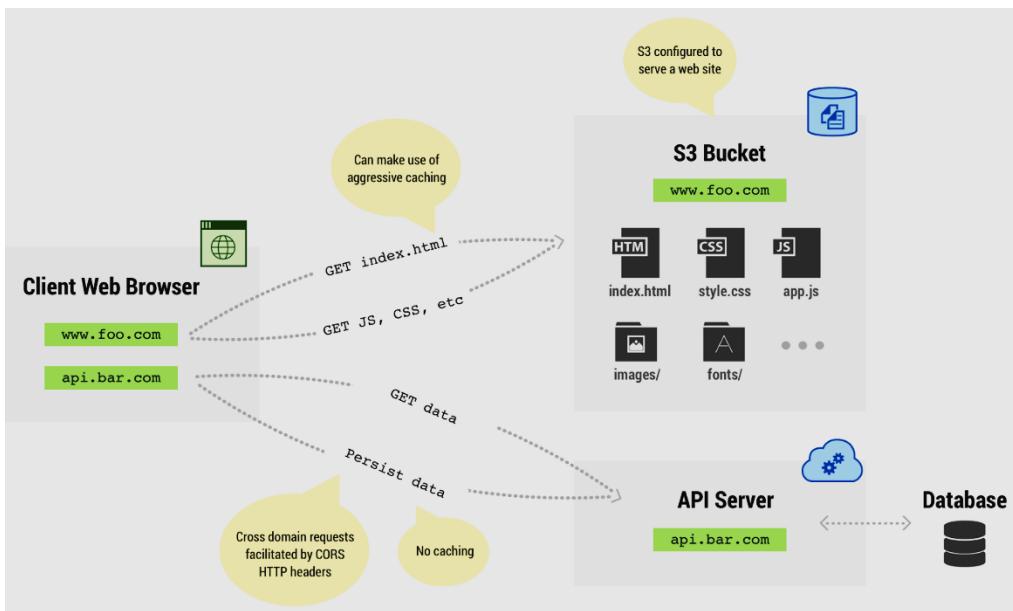


Nesse caso o browser vai renderizar o index.html (que renderiza as demais páginas da aplicação) apenas uma vez, todas as outras informações serão carregadas por demanda, conforme functions, módulos, etc.

Gera menos custos para renderizar a página, e oferece uma melhor experiência, pois o usuário não verá a página recarregando inúmeras vezes.

Ainda é possível compilar a aplicação para Desktop ou Mobile, tendo uma aplicação multiplataforma com um único código fonte.

Segue um esquema de uso de SPA:



9. Visite algumas páginas para validar as informações trabalhadas, e com a aba 'network' do 'modo dev' do seu navegador observar os fluxos de carregamento:
 - a. MPA: G1, UOL, Bradesco, Sites que utilizam Wordpress no geral, etc.;
 - b. Mesclam MPA e SPA: Youtube, Facebook, Github entre outros ;
 - c. SPA: Gmail, Netflix, Twitter, Airbnb, Microsoft Office, etc.

10. Após a visita, insira mais um site na lista, e pense junto com o docente e os outros alunos:

- Quando devemos usar SPA e quando não devemos?

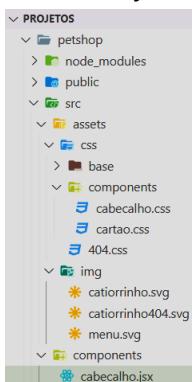
Atividade 3 – Criando e integrando navegação ao projeto

Objetivos:

- Criar cabeçalho e barra de navegação.
- Integrar dois componentes a mesma página, e em todo o projeto;

Vamos criar um cabeçalho e uma barra de navegação para integrar a todas das páginas carregadas.

- Copie do material os seguinte arquivos da pasta ‘cap03_arq_trabalho/’*
 - ‘cabecalho.css’, para o diretório ‘src/assets/css/components’;*
 - ‘catiorrinho.svg’ e ‘menu.svg’, para o diretório ‘src/assets/img/’;*
- Crie um diretório, dentro de ‘src’, chamado ‘components’, não confunda com o que já foi criado dentro da pasta ‘css’. Dentro do diretório ‘petshop/src/components’, criamos o arquivo ‘Cabecalho.jsx’. Verifique como ficou a estrutura de diretórios:*



- Importe o React, o CSS, a imagem e crie o componente ‘Cabecalho’, conforme o código abaixo:*

```
import React from "react";
import './assets/css/components/cabecalho.css';
import imagem from './assets/img/catiorrinho.svg';

const Cabecalho = () => {
    return(
        )
}

export default Cabecalho
```

- Montamos a estrutura da página com a linguagem HTML, dentro do ‘return’, seguindo os mesmo padrão de CSS das outras páginas, para depois trabalharmos com os elementos do React, como segue:*

```
<header className="cabecalho container">
```

```

<div className="menu-hamburguinho">
  <span className="menu-hamburguinho_icone"></span>
</div>
<div className="cabecalho-container">
  <a href="/" className="flex--centro">
    <img className="cabecalho_logo" src={imagem} alt="Logo Catiorrinho"/>
    <h1 className="cabecalho_titulo">PetShop</h1>
  </a>
</div>

<nav className="menu-cabecalho">
  <ul className="menu-itens">
    <li><a href="#" className="menu-item menu-item--entrar">Entrar</a></li>
    <li><a href="#" className="menu-item">Produtos</a></li>
    <li><a href="/" className="menu-item">Blog</a></li>
    <li><a href="/sobre" className="menu-item">Sobre</a></li>
  </ul>
</nav>
<div className="menu-cabecalho-background"></div>
</header>

```

5. Para renderizar a página e direcionar a rota que agora irá compor todas as páginas, abra o arquivo ‘App.js’, aproveite para organizar o arquivo, manter as importações gerais no início, logo abaixo os componentes importados, e inclua a rota fora da estrutura ‘switch’, para que possa ser herdada por todos os componentes, como segue no código abaixo:

```

import React from 'react'
import './assets/css/base/base.css'
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom'

import Home from './paginas/Home';
import Sobre from './paginas/Sobre';
import Pagina404 from './paginas/Pagina404';
import Cabecalho from './components/cabecalho';

function App() {
  return (
    <Router>
      <Cabecalho />
      <Switch>
        <Route exact path='/'>
          <Home />
        </Route>
        <Route path='/sobre'>
          <Sobre />
        </Route>
        <Route path='*'>
          <Pagina404 />
        </Route>
      </Switch>
    </Router>
  )
}

export default App

```

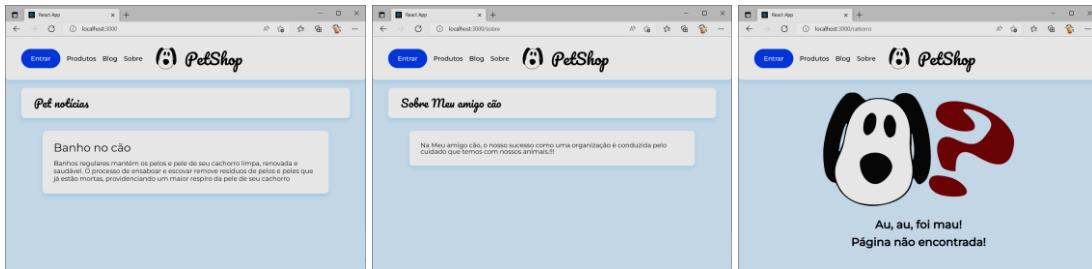
```

        )
}

export default App

```

6. O componente ‘Cabecalho’ foi atribuído a todas as páginas. Observe o resultado após a renderização:



7. Com isso finalizamos mais uma atividade;

Atividade 4 – Otimizando o carregamento de páginas na SPA

Objetivos:

- Otimizar o carregamento da SPA através de Link do React Router.

Mesmo sendo uma SPA, as páginas estão sofrendo ‘refresh’ quando carregadas, vamos fazer ajustes para que isso não ocorra.

1. Acesse o componente ‘Cabecalho’ e vamos utilizar um componente do React Router chamado ‘Link’, para correção, inicie fazendo a importação como abaixo:

```

import React from 'react';
import { Link } from 'react-router-dom';
import '../assets/css/components/cabecalho.css';

```

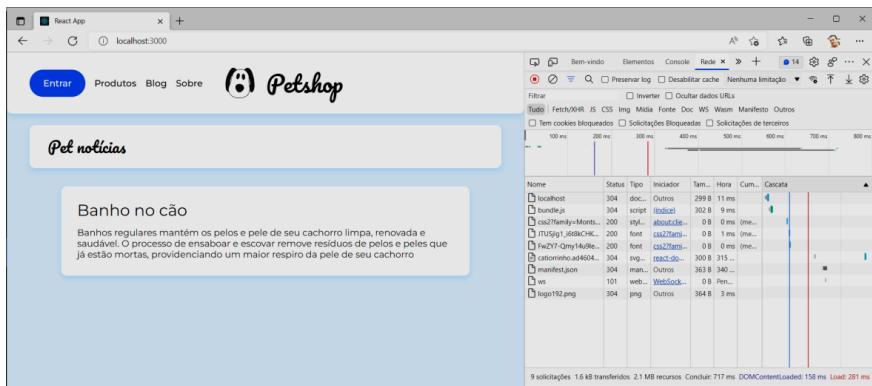
2. Na navegação, substitua a tag <a>, do ‘Sobre’, por ‘link’, e o parâmetro ‘href’, por ‘to’, desta forma:

```

<nav className="menu-cabecalho">
  ...
  <li><Link to="/sobre" className="menu-item">Sobre</Link></li>
</ul>
</nav>

```

3. Renderize a SPA, ative o ‘modo dev’, pressionando F12, no seu navegador e note que ao navegar para a página inicial ainda ocorre o ‘refresh’ da página, porém ao navegar para ‘Sobre’, a página não é recarregada.



4. Agora que ficou claro a funcionalidade da SPA, vamos aplicar o mesmo conceito aos outros elementos de navegação:

```
const Cabecalho = () => {
  return(
    <header className="cabecalho container">
      ...
      <div className="cabecalho-container">
        <Link to="/" className="flex flex--centro">
          <img className="cabecalho_logo" src={imagem} alt="Logo Catiorrinho"/>
          <h1 className="cabecalho_titulo">Petshop</h1>
        </Link>
      </div>

      <nav className="menu-cabecalho">
        <ul className="menu-itens">
          <li><Link to="#" className="menu-item menu-item--entrar">Entrar</Link></li>
          <li><Link to="#" className="menu-item">Produtos</Link></li>
          <li><Link to="/" className="menu-item">Blog</Link></li>
          <li><Link to="/sobre" className="menu-item">Sobre</Link></li>
        ...
      </ul>
    </header>
  )
}
```

5. Renderize novamente a SPA, ative o 'modo dev', pressionando F12, no seu navegador e note que ao navegar entre as páginas não ocorre mais o recarregamento. Atividade finalizada com sucesso.

Atividade Extra

Objetivos:

- Rever e aprimorar os conhecimentos construídos.

Vamos trabalhar algumas atividades extra para rever e aprimorar o que trabalhamos até aqui:

1. Faça a sugestão aos alunos que desenvolvam mais uma página para compor o app;

CAPÍTULO 4 – INTRODUÇÃO A API

Neste capítulo vamos entender o funcionamento de uma API - Application Programming Interface, ou seja, interface de programação de aplicativos que vai permitir a troca de informações entre a interface do usuário e um conjunto de informações, seja de um arquivo ou de um banco de dados.

Vamos implementar no projeto do Pet Shop uma API, tirar a simular as informações dos posts, e em seguida iremos trabalhar com essas informações.

Objetivos:

- Estabelecer uma conexão e consumir dados da API.
- Exibir posts através de dados dinâmicos.
- Elaborar estrutura para exibir posts através de dados dinâmicos.
- Criar estados de um componente.
- Visualizar dados dinâmicos da API renderizados na página;
- Renderizar um post específico;
- Redirecionar para página de erro caso um post não seja encontrado.

Atividade 1 – Conectando com a API

Objetivos:

- Estabelecer uma conexão e consumir dados da API.
- Exibir posts através de dados dinâmicos.

Atualmente os dados já estão inseridos dentro do HTML da página, porém isso seria complexo de ser atualizado, pois a cada dado novo teria que haver uma inserção no código. Vamos instalar um servidor JSON para estabelecer uma conexão com um arquivo que irá simular o banco de dados, e fazer a leitura direta das informações, e exibir nas páginas.

1. *Inicie a atividade, copiando do material o arquivo ‘cap04_arq_trabalho/db.json’, para o diretório do projeto, em seguida vamos instalar o JSON Server, abrindo o terminal e executando o seguinte comando:*

a. *Maiores informações em <https://www.npmjs.com/package/json-server> ;*

```
npm install -g json-server
```

```
npm i json-server
```

```
C:\Projetos\petshop>npm install -g json-server
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.
added 182 packages, and audited 183 packages in 16s

20 packages are looking for funding
  run `npm fund` for details

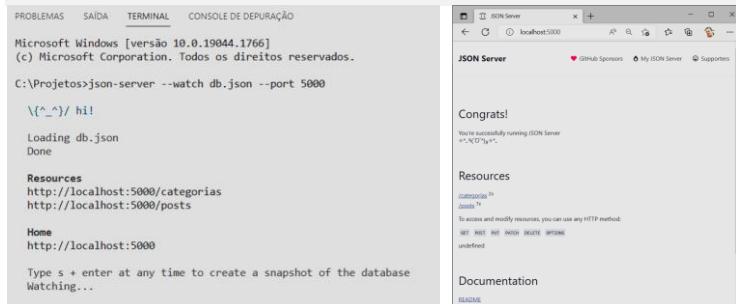
found 0 vulnerabilities
```

2. Verifique se 'JSON Server' foi integrado ao pacote de dependências do json abrindo o arquivo 'petshop/package.json', e verificando as dependências. Caso não esteja incluído, insira o código como abaixo:

```
{
  "name": "petshop",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    ...
    "@testing-library/user-event": "^13.5.0",
    "json-server": "^0.17.0",
    "react": "^18.2.0",
    ...
  },
}
```

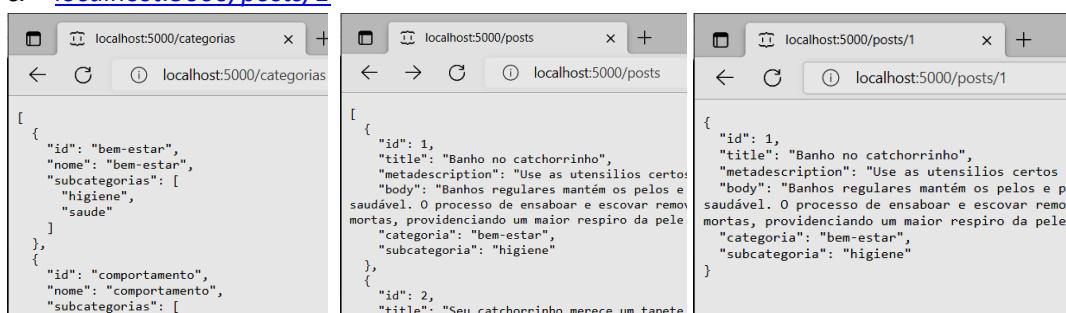
3. Ative o servidor React e depois o servidor de dados com o comando:

```
npm start
json-server --watch db.json --port 5000
```



4. Verifique o arquivo de dados acessando o url do JSON Server e os Arrays funcionam com tabelas, e os ids, como keys, para filtrar os dados, como no exemplos do links:

- localhost:5000/categorias
- localhost:5000/posts
- localhost:5000/posts/1



5. Conforme a sugestão do Fernando Jacinto da Silva, da Unidade Senac – Campinas; Insira no 'package.json', dentro dos scripts, o seguinte comando para facilitar a inicialização do Servidor Json "server": "json-server --watch db.json --port 5000"

```
{
  "name": "petshop",
  ...
  "scripts": {
```

```

    ...
      "eject": "react-scripts eject",
      "server": "json-server --watch db.json --port 5000"
    },
...

```

6. Pare o servidor e execute novamente com o comando:

```
npm run server
```

7. Insira mais algumas informações no arquivo e finalize a atividade confirmando a leitura dos dados.

Atividade 2 – Recebendo os dados da API

Objetivos:

- Elaborar estrutura para exibir posts através de dados dinâmicos.

Com os dados disponíveis, é preciso que sejam visualizados na aplicação, isto é, vamos construir uma estrutura para consumir os dados de um arquivo que simulará a nossa API.

1. Para dar suporte à conexão e aos dados dinâmicos vamos instalar o ‘Axios’, que é um cliente HTTP, para fazer requisições, tanto no Node.js ou qualquer serviço API. Para instalação abra o terminal e digite o seguinte comando:

```
npm install -g axios
```

```
npm i axios
```

```

PROBLEMAS SAÍDA TERMINAL CONSOLE DE DEPURAÇÃO
Microsoft Windows [versão 10.0.19044.1766]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Projetos>npm install -g axios
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.

added 8 packages, and audited 9 packages in 3s

1 package is looking for funding
  run `npm fund` for details

found 0 vulnerabilities

C:\Projetos>|

```

2. Verifique se o ‘Axios’ foi integrado ao pacote de dependências do json abrindo o arquivo ‘petshop/package.json’, e verificando as dependências. Caso não esteja incluído, insira o código como abaixo:

```
{
  "name": "petshop",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    ...
    "@testing-library/user-event": "^13.5.0",
    "axios": "^0.27.2",
    "json-server": "^0.17.0",
    "react": "^18.2.0",
    ...
  },
}
```

3. Crie um diretório dentro de 'petshop/src', chamado 'api', dentro dele um arquivo chamado 'api.js', importe o Axios, crie as exportações da API, para disponibilizar os dados para renderização no projeto. Com o código abaixo:

- a. O servidor de dados será no endereço: <http://localhost:5000>, inicialmente em uma busca genérica;
- b. As 'props', ficam definidas de modo assíncrono em relação a url e configuração de dados
- c. a 'busca' dos dados deve aguardar a resposta da API;

```
import axios from 'axios';

export const api = axios.create({
  baseURL: 'http://localhost:5000'
})

export const busca = async() => {
  const resposta = await api.get(url)
}
```

4. Dentro do diretório 'petshop/components' crie o arquivo 'ListaPost.jsx', para criarmos o componente, para consumir os dados da API, e exibir dentro de um '<section>', da seguinte forma:

```
import React from 'react';
import { busca } from '../api/api.js';

const ListaPost = () => {
  return(
    <section className="posts container">
      </section>
  )
}

export default ListaPost
```

5. Finalizamos a estrutura para receber os dados.

Atividade 3 – Estados de componente e visualização dinâmica de dados

Objetivos:

- Criar estados de um componente.
- Visualizar dados dinâmicos da API renderizados na página;

Com a estrutura disponível e o componente criado temos que 'alterar o estado', para leitura das informações. O estado de um componente pode ser alterado em resposta a manipuladores de eventos, respostas do servidor ou alterações de prop. Para isso usamos o método `useState()`, que 'enfileira' as atualizações no estado do componente e instrui o React a renderizar novamente o componente e seus filhos com o estado atualizado.

1. Abra o arquivo 'ListaPost.jsx', para que possamos configurar os estados do nosso componente, conforme o código abaixo:

- a. Configuramos o useState, para exibir o estado inicial;
- b. Estabelecer que a estrutura deve ser repetida para cada post;

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

const ListaPost = ( { url } ) => {

  const [posts, setPosts] = useState([])

  return(
    <section className="posts container">
      {
        posts.map((post)=> (
          )
        )
      }
    </section>
  )
}

export default ListaPost
```

2. Ainda no arquivo 'ListaPost.jsx', passamos configurar os dados do nosso componente, conforme o código abaixo:

- a. Tenha um compreensão clara e digite com calma as informações, pois no 'return', inserimos os dados no formato, 'post.informação'. Exemplo: 'post.categoria' e 'post.id';

```
import React, { useState } from 'react';
import { Link } from 'react-router-dom'

const ListaPost = ( { url } ) => {
  ...
  posts.map((post)=> (
    <Link className={`cartao-post cartao-post--${post.categoria}`}
to={`/posts/${post.id}`}
      <article key={post.id}>
        <h3 className="cartao-post__titulo">
          {post.title}
        </h3>
        <p className="cartao-
post__meta">{post.metadescription}</p>
      </article>
    </Link>
  )
)
</section>
}

export default ListaPost
```

3. É preciso que o componente busque as informações e as exiba nos campos que determinamos, então vamos incluir no código, useEffect para realizar essa busca:

```

import React, { useState, useEffect } from 'react'
import { Link } from 'react-router-dom'
import { busca } from '../api/api'

const ListaPost = ( { url } ) => {

const [posts, setPosts] = useState([])

useEffect(() => {
  busca(url, setPosts)
}, [])

return(
...
)
}

export default ListaPost

```

4. Acesse o arquivo ‘petshop/src/api/api.js’ para inserir os parâmetros de url e configuração de dados da busca:

```

import axios from 'axios';

export const api = axios.create({
  baseURL: 'http://localhost:5000'
})

export const busca = async(url, setDado) => {
  const resposta = await api.get(url)
  setDado(resposta.data)
}

```

5. Para renderização acesse o arquivo “petshop/src/paginas/Home.jsx”, e vamos apagar a <section>, para importar e inserir o componente que criamos:

```

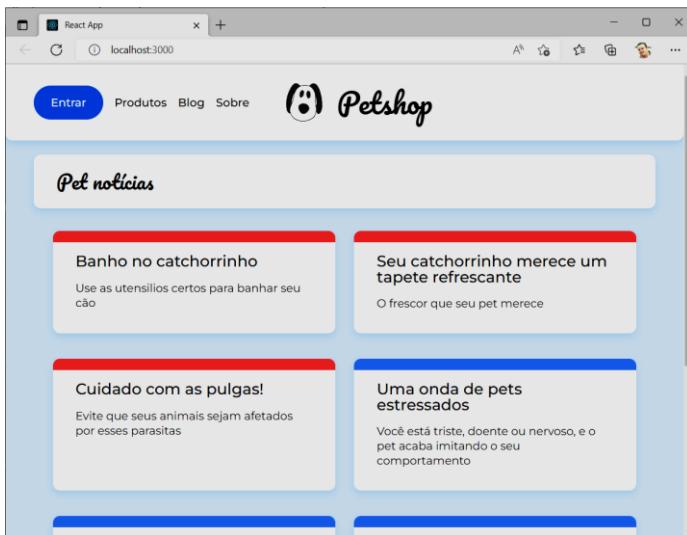
import React from "react";
import ListaPost from "../components/ListaPost";

const Home = () => {
  return(
    <main>
      <div className="container">
        <h2 className="titulo-pagina">Pet notícias</h2>
        </div>
        <ListaPost url={'/posts'} />
      </main>
  )
}

export default Home

```

6. Observe a página renderizada e finalize mais uma atividade;



Atividade 4 – Renderizando um componente específico

Objetivos:

- Renderizar um post específico;
- Redirecionar para página de erro caso um post não seja encontrado.

O projeto já está renderizando o componente com os todos os posts, porém vamos trabalhar para que ao clicar em um determinado post, as informações dele sejam exibidas em uma página específica.

1. *Dentro do diretório ‘petshop/src/paginas’ crie o arquivo ‘Post.jsx’, para criarmos o componente, para exibir um componente específico, com os dados de apenas um post da API. Revise alguns conceitos:*
 - a. *useState, é estado inicial;*
 - b. *useEffect é atualiza o estado de um componente segundo uma ação;*

```
import React, {useState, useEffect } from 'react'
import { busca } from '../api/api'

const Post = () => {
  const[post, setPost] = useState({})

  useEffect(() => {
    busca(`/posts/${id}`, setPost)
  }, [id])

  return(
  )
}

export default Post
```

2. Crie o arquivo 'post.css', dentro do diretório 'petshop/assets/css/', para inserirmos um pequeno código:

```
.post {  
  margin-top: 1.25rem;  
}
```

3. Construa o return para exibição dos dados específicos do post:

```
import React, {useState, useEffect } from 'react'  
import { busca } from '../api/api'  
import '../assets/css/post.css'  
  
const Post = () => {  
  const[post, setPost] = useState({})  
  
  useEffect(() => {  
    busca(`/posts/${id}`, setPost)  
  }, [id])  
  
  return(  
    <main className="container flex flex--centro">  
      <article className="cartao post">  
        <h2 className="cartao__titulo">{post.title}</h2>  
        <h3 className="cartao-post__meta">{post.metadescription}</h3>  
        <br />  
        <p className="carta__texto">{post.body}</p>  
      </article>  
    </main>  
  )  
}  
  
export default Post
```

4. Abra o arquivo 'petshop/src/App.js' para realizar o ajuste das rotas.

```
import React from 'react'  
...  
import Cabecalho from './components/cabecalho';  
import Post from './paginas/Post';  
  
function App() {  
  
  return (  
...  
    <Route path='/sobre'>  
      <Sobre />  
    </Route>  
    <Route path='/posts/:id'>  
      <Post />  
    </Route>  
    <Route path='*'>  
      <Pagina404 />  
    </Route>  
  </Switch>  
</Router>
```

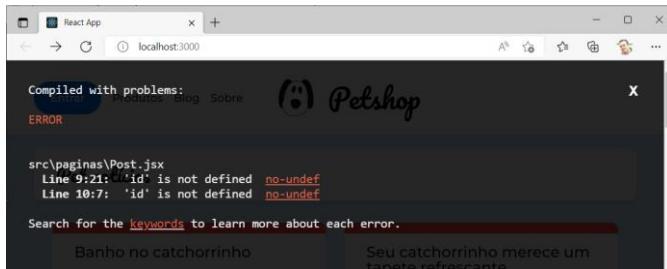
```

    )
}

export default App

```

5. Após as modificações de rota, renderize o projeto, e note que um erro foi exibido, o 'id' não está definido:



6. Os parâmetros precisam ser reconhecidos no envio e na recepção da url, no arquivo 'Post.jsx' inseri novos códigos:

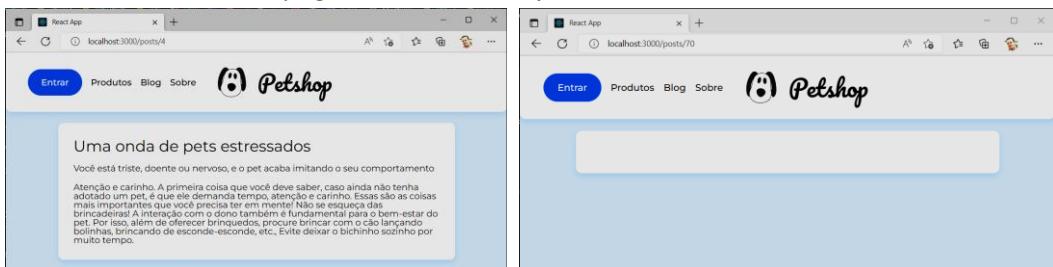
```

import React, {useState, useEffect } from 'react';
import { useParams } from 'react-router-dom';
import { busca } from '../api/api';

...
const Post = () => {
  const { id } = useParams()
  const[post, setPost] = useState({})
  ...
}

```

7. Observe que o resultado da renderização é positivo, porém se tentarmos acessar um post através da url, por exemplo, <http://localhost:3000/posts/70>, você não consegue, e mesmo que ele não exista, não é exibida a 'pagina404' e sim o layout vazio, como mostrado:



8. Acesse novamente o arquivo 'Post.jsx' e vamos utilizar o *catch*, que especifica uma resposta, caso haja uma exceção. O *Catch* vai acessar o histórico de navegação, captura a rota inválida e redireciona.

```

import React, {useState, useEffect } from 'react';
import { useHistory, useParams } from 'react-router-dom';
import { busca } from '../api/api';
import '../assets/css/post.css'

const Post = () => {
  let history = useHistory()
  const { id } = useParams()
  const[post, setPost] = useState({})

```

```

useEffect(() => {
  busca(`/posts/${id}`, setPost)
  .catch(()=>{
    history.push('/404')
  })
}, [id])

return(
...

```

9. Acesse novamente a url, <http://localhost:3000/posts/70>, e não consegue obter um post, porém agora será exibida a 'pagina404':



10. Atividade finalizada com sucesso.

Atividade Extra

Objetivos:

- Rever e aprimorar os conhecimentos construídos.

Vamos trabalhar algumas atividades extra para rever e aprimorar o que trabalhamos *até aqui*:

1. *Inclua mais posts no projeto;*
2. *Pesquise mais sobre o useState, stateful e Stateless;*

CAPÍTULO 5 – APERFEIÇOANDO O USO DA API

Agora que já sabemos os conceitos e a utilização de uma API, vamos aperfeiçoar nosso projeto exibindo os posts, filtrando as informações disponibilizadas, primeiramente por categoria, depois por subcategorias, abordando o uso de rotas aninhadas, o uso de endereços dinâmicos e o carregamento de vários componentes interdependentes.

Objetivos:

- Criar componentes derivados da API para categorizar Posts;
- Configurar Rotas derivadas de outras rotas, chamadas rotas aninhadas.
- Exibir Post inicialmente filtrados por categoria, de forma dinâmica, consultando a API
- Refinar os filtros em subcategorias;
- Renderizar um componente dentro de outro;
- Estabelecer Rotas dinâmicas.

Atividade 1 – Criando Links para exibição por categoria

Objetivos:

- Criar componentes derivados da API para categorizar Posts;

Os dados disponibilizados dividem os Posts em Categorias, e vamos fazer uma leitura desses dados e exibir as categorias existentes.

1. *Copie do material o arquivo ‘cap05_arq_trabalho/blog.css’, para o diretório do projeto, ‘petshop/src/assets/css/blog.css’, para a depois importar no arquivo que criaremos a seguir;*
2. *Dentro do diretório ‘petshop/src/components’ crie o arquivo ‘ListaCategorias.jsx’, para configurarmos o componente, com a finalidade de exibir os links que irão direcionar a página de posts por categoria:*

```
import React, { useState, useEffect } from "react";
import { busca } from '../api/api.js';
import './assets/css/blog.css';

const ListaCategorias = () => {
    const [categorias, setCategorias] = useState([])
    useEffect(() => {
        busca(`/categorias`, setCategorias)
    }, [])
    return(
    )
}
```

```
export default ListaCategorias
```

3. Ainda no arquivo 'ListaCategorias.jsx', passamos a configurar os dados do nosso componente, criando links para exibição por categoria, conforme o código abaixo:

```
import React, { useState, useEffect } from "react";
import { Link } from 'react-router-dom';
...
return(
    <ul className="lista-categorias container flex">
    {
        categorias.map((categoria) => (
            <Link to={`/categoria/${categoria.id}`}>
                <li className={`lista-categorias__categoria lista-categorias__categoria--${categoria.id}`}>
                    {categoria.nome}
                </li>
            </Link>
        ))
    }
    </ul>
)
...
...
```

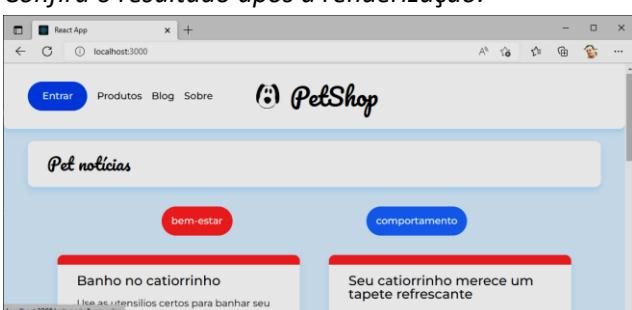
4. Abra o arquivo 'petshop/src/paginas/Home.jsx' para inserir o componente 'ListaCategorias', e assim colocar os botões de link acima dos posts, assim exibindo as categorias:

```
import React from "react";
import ListaCategorias from "../components/ListaCategorias";
import ListaPost from "../components/ListaPost";

const Home = () => {
    return(
        <main>
            <div className="container">
                <h2 className="titulo-pagina">Pet notícias</h2>
            </div>
            <ListaCategorias />
            <ListaPost url={'/posts'} />
        </main>
    )
}

export default Home
```

5. Confira o resultado após a renderização:



- Após a renderização, os links de categoria ainda não funcionam, mas essa será a tarefa da próxima atividade.

Atividade 2 – Rotas aninhadas por categoria

Objetivos:

- Configurar Rotas derivadas de outras rotas, chamadas rotas aninhadas.

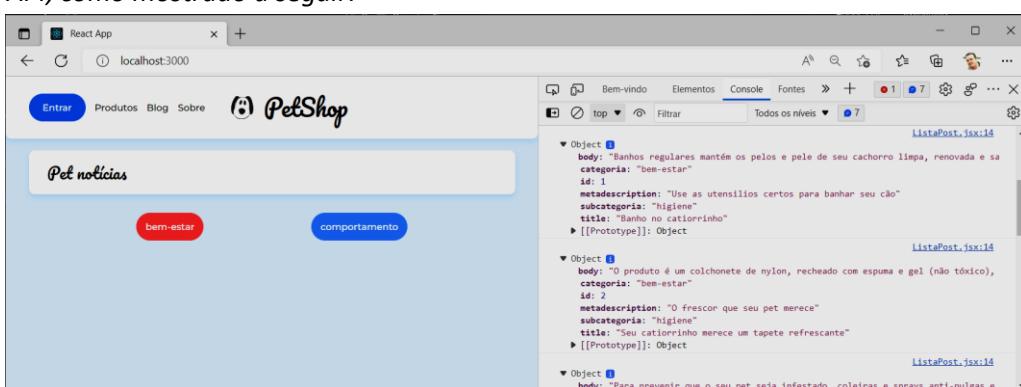
Nosso projeto tem um página de posts e já estão categorizados, então vamos construir rotas para filtrar os dados através das categorias;

- Altere o código abaixo para verificar a leitura dos dados da API, e observar a divisão por categorias;

```
import React, { useState, useEffect } from "react";
...
return(
    <section className="posts container">
    {
        posts.map((post)=> {
            console.log(post);
        ...
    })
    </section>
)
}

export default ListaPost
```

- Acesse o modo de desenvolvedor do navegador para visualizar o resultado, verifique a leitura da API, como mostrado a seguir:



- Retorne ao código anterior para continuarmos a atividade:

```
import React, { useState, useEffect } from "react";
import { busca } from '../api/api.js';
import { Link } from 'react-router-dom';

const ListaPost = ({ url }) => {
```

```

        const [posts, setPosts] = useState([])
        useEffect(() => {
            busca(url, setPosts)
        }, [url])
        return (
            <section className="posts container">
                {
                    posts.map((post) => (
                        //console.log(post);
                        <Link className={`cartao-post cartao-post--${post.categoria}`} to={`/posts/${post.id}`} >
                            <article key={post.id}>
                                <h3 className="cartao-post__titulo">
                                    {post.title}
                                </h3>
                                <p className="cartao-post__meta">
                                    {post.metadescription}
                                </p>
                            </article>
                        </Link>
                    ))
                }
            </section>
        )
    }

    export default ListaPost

```

4. Dentro do diretório 'petshop/src/paginas' crie o arquivo 'Categoria.jsx', para configurarmos o componente, com a finalidade de exibir os posts por categoria:

```

import React from 'react';
import '../assets/css/blog.css';
import ListaCategorias from '../components/ListaCategorias';

const Categoria = () => {
    return(
        <>
            <div className="container">
                <h2 className="titulo-pagina">Pet Notícias</h2>
            </div>
            <ListaCategorias />
        </>
    )
}

export default Categoria

```

5. Abra o arquivo 'petshop/src/App.js' para realizar o ajuste das rotas. Note que temos uma rota que verifica o 'id' da categoria:

```

import React from 'react';
...
import Post from './paginas/Post';
import Categoria from './paginas/Categoria';

```

```

function App() {
  return (
    ...
      <Route path='/sobre'>
        <Sobre />
      </Route>
      <Route path='/categoria/:id'>
        <Categoria />
      </Route>
    ...
  )
}

```

6. Ao renderizar a página não será exibida erro, porém os posts da categoria não serão exibidos.



7. Finalizamos a atividade com o desafio de exibir os posts dentro das categorias configuradas

Atividade 3 – Exibindo Posts por categoria

Objetivos:

- Exibir Post inicialmente filtrados por categoria, de forma dinâmica, consultando a API;

As rotas por categorias já estão definidas e configuradas, então vamos codificar a exibição dos Posts relacionados, e permitir a navegação entre as opções disponíveis.

1. Para exibir os posts é preciso determinar qual será a categoria, e definir os elementos do filtro no componente, para isso vamos incluir uma rota ao componente ‘Categoria’, utilizando o ‘id’ da categoria como parâmetro, e a princípio exibindo a categoria ‘bem-estar’ na renderização:

```

import React from 'react';
import { Route, useParams } from 'react-router-dom';
import '../assets/css/blog.css';
import ListaCategorias from '../components/ListaCategorias';
import ListaPost from '../components/ListaPost';

const Categoria = () => {
  const { id } = useParams()
  return(
    <>
      <div className="container">
        <h2 className="titulo-pagina">Pet Notícias</h2>
      </div>
      <ListaCategorias />
    </>
  )
}

```

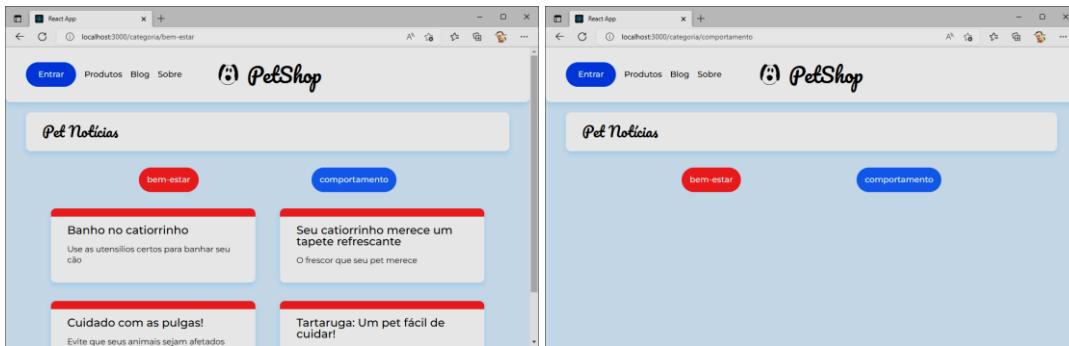
```

        <Route exact path={`/categoria/bem-estar`} >
            <ListaPost url={`/posts?categoria=${id}`} />
        </Route>
    </>
)
}

export default Categoria

```

2. Renderize a páginas e realize o teste clicando na categoria 'bem-estar' observe que os post são filtrados, porém ao clicar na categoria 'comportamento' nenhum post será exibido;



3. Para correção, do componente 'Categoria' utilize um 'hook', o 'useRouteMatch', para buscar o caminho de forma dinâmica e posterior utilização, conforme o código:

- a. Aproveite para lembrar que os 'Hooks' são funções que permitem "ligar-se" aos recursos de 'state' a partir de componentes funcionais. Eles permitem que você use React sem classes.

```

import React from 'react';
import { Route, useParams, useRouteMatch } from 'react-router-dom';
import '../assets/css/blog.css';
import ListaCategorias from '../components/ListaCategorias';
import ListaPost from '../components/ListaPost';

const Categoria = () => {
    const { id } = useParams()
    const { path } = useRouteMatch()
    return(
        <>
            <div className="container">
                <h2 className="titulo-pagina">Pet Notícias</h2>
            </div>
            <ListaCategorias />
            <Route exact path={`${path}/`}>
                <ListaPost url={`/posts?categoria=${id}`} />
            </Route>
        </>
    )
}

export default Categoria

```

4. Nesse momento estamos renderizando as duas categorias, porém não é possível navegar entre elas, pois o componente 'ListaPost.jsx', não foi programado para atualizar na mudança de url, da categoria, para ajustar é preciso apenas um pequeno ajuste no código.:

```
import React, { useState, useEffect } from "react";
...
const ListaPost = ( { url } ) => {
  const [posts, setPosts] = useState([])
  useEffect(() => {
    busca(url, setPosts)
  }, [url])
  return(
  ...
}
```

5. Finalizamos a exibição dos posts por categorias.

Atividade 4 – Refinando as categorias em subcategorias

Objetivos:

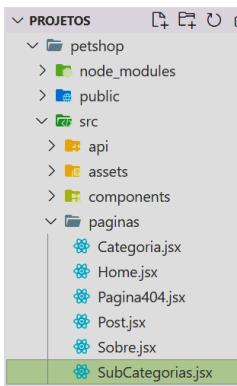
- Refinar os filtros em subcategorias;
- Renderizar um componente dentro de outro;
- Estabelecer Rotas dinâmicas.

O projeto já está renderizando categorias, porém vamos trabalhar para que ao clicar em uma categoria, sejam exibidas suas subcategorias, e ao escolher uma delas, sejam exibidos os posts relacionados.

1. Observe no arquivo 'petshop/db.json', que existem subcategorias:

```
{
  "categorias": [
    {
      "id": "bem-estar",
      "nome": "bem-estar",
      "subcategorias": ["higiene", "saude"]
    ...
  ],
  "posts": [
    {
      "id": 1,
      ...
      "categoria": "bem-estar",
      "subcategoria": "higiene"
    },
  ...
}
```

2. Vamos exibir as subcategorias na renderização da página de Posts. Crie um arquivo em 'petshop/src/paginas', com o nome de 'SubCategorias.jsx';



3. Esse é um componente bem simples, aproveite para revisar alguns conceitos, importação de parâmetros para o uso em urls personalizadas, fique atento enquanto digita o código:

```

import React from 'react';
import { useParams } from 'react-router-dom';
import ListaPost from '../components/ListaPost';

const SubCategoria = () => {
  const { subcategoria } = useParams()
  return(
    <ListaPost url={`/posts?subcategoria=${subcategoria}`} />
  )
}

export default SubCategoria

```

4. Para exibir as subcategorias é necessário alterar o estado do componente 'Categoria.jsx', para exibir as opções e posteriormente os posts relacionados, para tanto altere o código para determinar os estados:

```

import React, { useEffect, useState } from 'react';
import { Route, useParams, useRouteMatch } from 'react-router-dom';
import { busca } from '../api/api';
import '../assets/css/blog.css';
import ListaCategorias from '../components/ListaCategorias';
import ListaPost from '../components/ListaPost';

const Categoria = () => {
  const { id } = useParams()
  const { url, path } = useRouteMatch()
  const [subcategorias, setSubCategorias] = useState([])

  useEffect(() => {
    busca(`/categorias/${id}`, (categoria) => {
      setSubCategorias(categoria.subcategorias)
    })
  }, [id])
  return(
    ...
  )
}

export default Categoria

```

5. Vamos inserir no componente 'Categoria.jsx' a exibição das subcategorias, inserindo logo abaixo de 'ListaCategorias' o seguinte código:

```

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

```

```

import { Route, useParams, useRouteMatch, Link } from 'react-router-dom';
...
import ListaCategorias from '../components/ListaCategorias';
import SubCategoria from '../paginas/SubCategoria'
import ListaPost from '../components/ListaPost';

const Categoria = () => {
    const { id } = useParams()
    ...
        <ListaCategorias />
        <ul className="lista-categorias container flex">
            {
                subcategorias.map((subcategory) => (
                    <li className={`lista-categorias__categoria lista-categorias__categoria--${id}`}>
                        <Link to={`${url}/${subcategory}`}>
                            {subcategory}
                        </Link>
                    </li>
                ))
            }
        </ul>
    ...

```

6. Em seguida faremos os ajustes das rotas para a subcategoria, considerando os filtros recebidos da categoria e filtrando novamente:

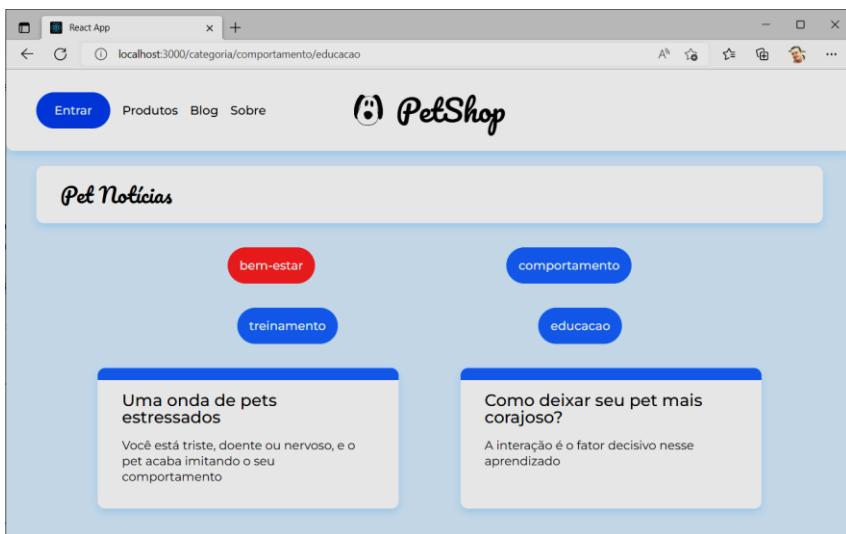
```

import React, { useEffect, useState } from 'react';
import { Route, useParams, useRouteMatch, Link, Switch } from 'react-router-dom';
...
        </ul>
        <Switch>
            <Route exact path={`${path}/`}>
                <ListaPost url={`/posts?categoria=${id}`} />
            </Route>
            <Route path={`${path}/:subcategory`}>
                <SubCategoria />
            </Route>
        </Switch>
    )
}

export default Categoria

```

7. Observe a página renderizada, filtre os posts por categorias e subcategorias para testar o funcionamento, corrija eventuais falhas e finalize mais um capítulo com sucesso;



Atividade Extra

Objetivos:

- Rever e aprimorar os conhecimentos construídos.

Vamos trabalhar algumas atividades extra para rever e aprimorar o que trabalhamos *até aqui*:

1. *Inclua mais posts, de novas categorias e/ou subcategorias no projeto;*
2. *Pesquise mais sobre o Hooks;*

CAPÍTULO 6 – INTRODUÇÃO AO CRUD COM REACT

Avançamos muito nos conhecimentos sobre o React, os Hooks e tantos outros elementos abordados nesse projeto, agora vamos a um passo importante, conhecer formas de interagir com os dados, inserindo e excluindo informações da nossa API.

Objetivos:

- Preparar os elementos básicos da área administrativa
- Instalar e utilizar uma biblioteca de estilos, a ‘MUI’
- Criar um formulário utilizando a biblioteca de estilos
- Realizar a configuração para componentes realizarem inserções na API
- Estabelecer os parâmetros de exclusão de dados da API

Atividade 1 – Preparando a área administrativa

Objetivos:

- Preparar os elementos básicos da área administrativa

Vamos preparar uma identificação e a lista de categorias que irão compor a área administrativa, e sem demora vamos a atividade.

1. Vamos então a criação da área administrativa. Crie uma pasta dentro do diretório de ‘petshop/src/paginas’, chamada ‘admin’, e dentro dela o arquivo ‘Admin.jsx’, com o código:

```
import React from "react";

const Admin = () => {
  return(
    <main>
      <div className="container">
        <h2 className="titulo-pagina">Administração</h2>
      </div>
    </main>
  )
}

export default Admin
```

2. Vamos acessar a pasta ‘src/components’ e abrir o componente ‘cabecalho.jsx’, para ajustar a navegação para direcionar para área administrativa

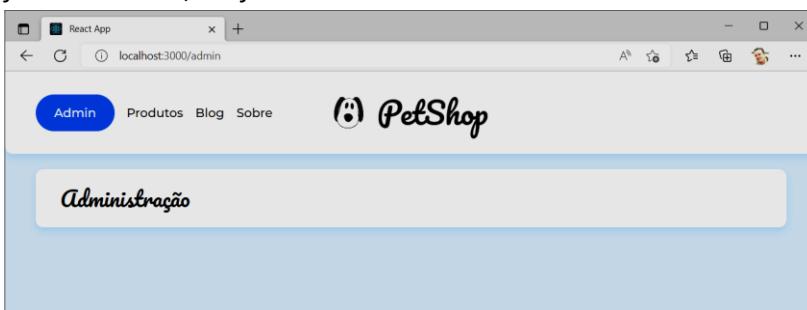
```
import React from "react";
...
<li>
  <Link to="/admin" className="menu-item menu-item--entrar">Admin</Link>
```

```
</li>  
...
```

3. Abra o arquivo 'App.js' para incluirmos a rota para a área administrativa:

```
import React from 'react';  
import './assets/css/base/base.css';  
...  
import Categoria from './paginas/Categoria';  
import Admin from './paginas/admin/Admin';  
...  
    <Route exact path='/'>  
        <Home />  
    </Route>  
    <Route exact path='/admin'>  
        <Admin />  
    </Route>  
...
```

4. Renderize o projeto e clique no botão de acesso a área administrativa para verificar o funcionamento, conforme o resultado:



5. Crie um diretório dentro da pasta 'admin', chamado 'components', e em seguida um arquivo com o nome 'ListaCatAdmin', para criarmos a visualização inicial da área administrativa. E realize a criação do componente como no código abaixo:

```
import React, { useEffect, useState } from "react";  
import { busca } from "../../api/api";  
  
const ListaCatAdmin = () => {  
    const [categorias, setCategorias] = useState([])  
    useEffect(() => {  
        busca(`/categorias`, setCategorias)  
    }, [])  
    return (  
    )  
}  
  
export default ListaCatAdmin
```

6. Acesse o arquivo disponível no material em 'cap06_arq_trabalho/tabela.css', e copie para pastas 'admin/components'. Em seguida importe para o componente 'ListaCatAdmin', e construa o return a princípio apenas utilizando código html:

```
import React, { useEffect, useState } from "react";  
import { Link } from "react-router-dom";
```

```

import "../components/tabela.css";

...
    return (
        <section className="container">
            <table className="tabela">
                <thead>
                    <tr>
                        <th className="tabela__coluna--g">Categoria</th>
                        <th colSpan="3" className="tabela__coluna--p">
                            tabela__alinhamento--direita</th><Link to="/admin/NovaCategoria">
                                Nova Categoria
                            </Link>
                        </th>
                    </tr>
                </thead>
                <tbody >
                    {
                        categorias.map((categoria) => (
                            <tr key={categoria.id}>
                                <td className="tabela__coluna--m">
                                    <Link
                                        to={`/categoria/${categoria.id}`}
                                    >{categoria.nome}</Link>
                                </td>
                                <td></td>
                                <td></td>
                            </tr>
                        ))
                    }
                </tbody>
            </table>
        </section>
    )
}

export default ListaCatAdmin

```

7. Insira o componente ‘ListaCatAdmin’, dentro do arquivo, ‘Admin.jsx’, conforme o código abaixo:

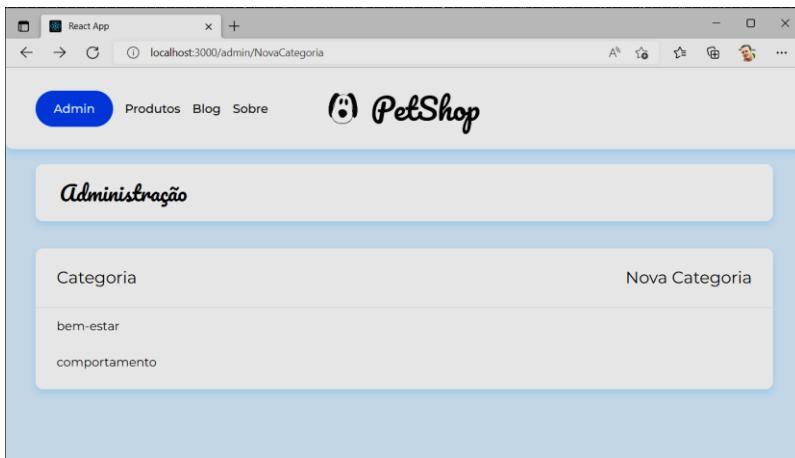
```

import React from "react";
import ListaCatAdmin from "./components/ListCatAdmin";

...
    <div className="container">
        <h2 className="titulo-pagina">Administração</h2>
    </div>
    <ListaCatAdmin />
</main>
...

```

8. Verifique o resultado da renderização e navegue pelas opções:



Atividade 2 – Utilizando a biblioteca de estilos ‘MUI’

Objetivos:

- Instalar e utilizar uma biblioteca de estilos, a ‘MUI’

Nosso projeto tem um visual bem bacana, mas existem elementos prontos que podemos usar com React, são as bibliotecas de estilo, vamos explorar uma e colocar alguns desses elementos no nosso projeto.

1. *Antes de iniciar as nossas atividades vamos instalar uma biblioteca de estilos chamada ‘mui’, acesse o site para maiores informações e o comando de instalação, aguarde um pouco levará um tempo até a instalação ser concluída, abra o terminal, inicie os servidores do React e o JSON server, então num novo terminal execute o comando:*
 - a. <https://mui.com/pt/> para a instalação;
 - b. <https://mui.com/pt/material-ui/react-table/> para a biblioteca de componentes.

```
npm install @mui/material @emotion/react @emotion/styled
```

```
PROBLEMAS SAIDA TERMINAL CONSOLAS DE DEPURAÇÃO

C:\Projetos\petshop>npm install @mui/material @emotion/react @emotion/styled
npm <v> config global '--global', '--local' are deprecated. Use '--location=global' instead.
npm <v> ERESOLVE overriding peer dependency
npm <v> While resolving: mini-create-react-context@0.4.1
npm <v> Found: react@19.2.0
npm <v> node_modules/react
npm <v>   peer react@"^18.0.0" from @testing-library/react@13.3.0
npm <v>   node_modules/@testing-library/react
npm <v>     @testing-library/react@"^13.3.0" from the root project
npm <v>     14 more (react-dom, react-router, react-router-dom, ...)
npm <v>
npm <v> Could not resolve dependency:
npm <v>   peer react@"^0.14.0 || ^15.0.0 || ^16.0.0 || ^17.0.0" from mini-create-react-context@0.4.1
npm <v>   node_modules/react-router/node_modules/mini-create-react-context
npm <v>     mini-create-react-context@"^0.4.0" from react-router@5.3.3
npm <v>     node_modules/react-router
npm <v>
npm <v>   Conflicting peer dependency: react@17.0.2
npm <v>   node_modules/react
npm <v>     peer react@"^0.14.0 || ^15.0.0 || ^16.0.0 || ^17.0.0" from mini-create-react-context@0.4.1
npm <v>     node_modules/react-router/node_modules/mini-create-react-context
npm <v>       mini-create-react-context@"^0.4.0" from react-router@5.3.3
npm <v>       node_modules/react-router
npm <v>
added 31 packages, and audited 1570 packages in 56s

206 packages are looking for funding
  run `npm fund` for details

11 vulnerabilities (5 moderate, 6 high)

To address all issues (including breaking changes), run:
  npm audit fix --force

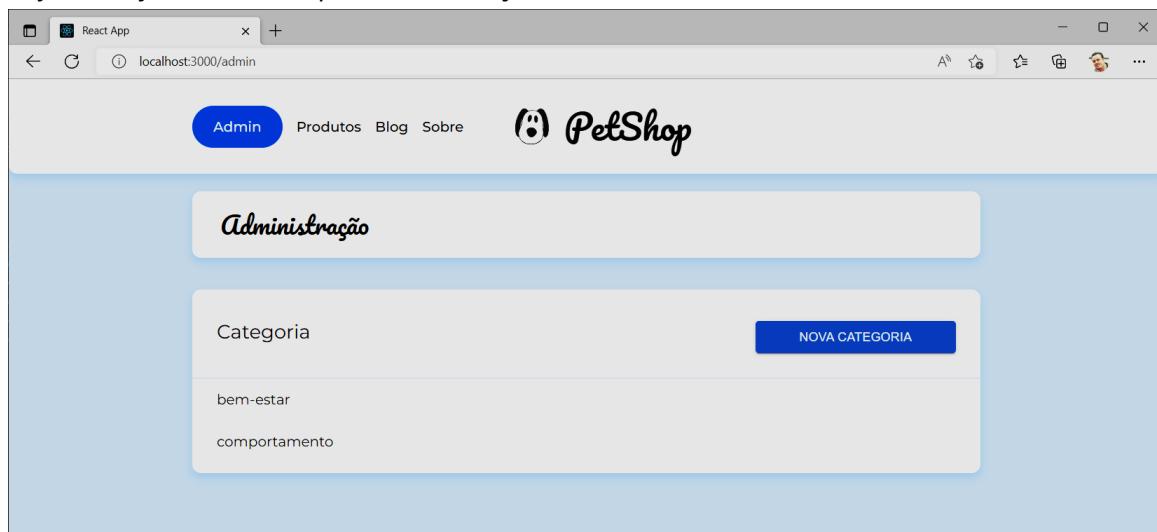
Run `npm audit` for details.
```

2. Vamos agregar opções da biblioteca 'mui' para estilizar os botões do componente 'ListaCatAdmin' e adicionar uma nota categoria e incluir botões para futuramente editar e excluir as categorias. Fique atento ao código:

```
import React, { useEffect, useState } from "react";
...
import './tabela.css';
import { Button } from "@mui/material";
...
<table className="tabela">
  <thead>
    <tr>
      <th className="tabela__coluna--g">Categoria</th>
      <th colSpan="3" className="tabela__coluna--p
tabela__alinhamento--direita"><Link to="/admin/NovaCategoria">
        <Button
          type="submit"
          variant="contained"
          fullWidth
          sx={{ marginTop: 1 }}>
          Nova Categoria
        </Button>
      </Link>
      </th>
    </tr>
  </thead>
  ...

```

3. Veja como ficou o botão após a renderização:



4. Em seguida aplica a biblioteca 'mui' também na criação dos botões de 'editar' e 'excluir', com o seguinte código:

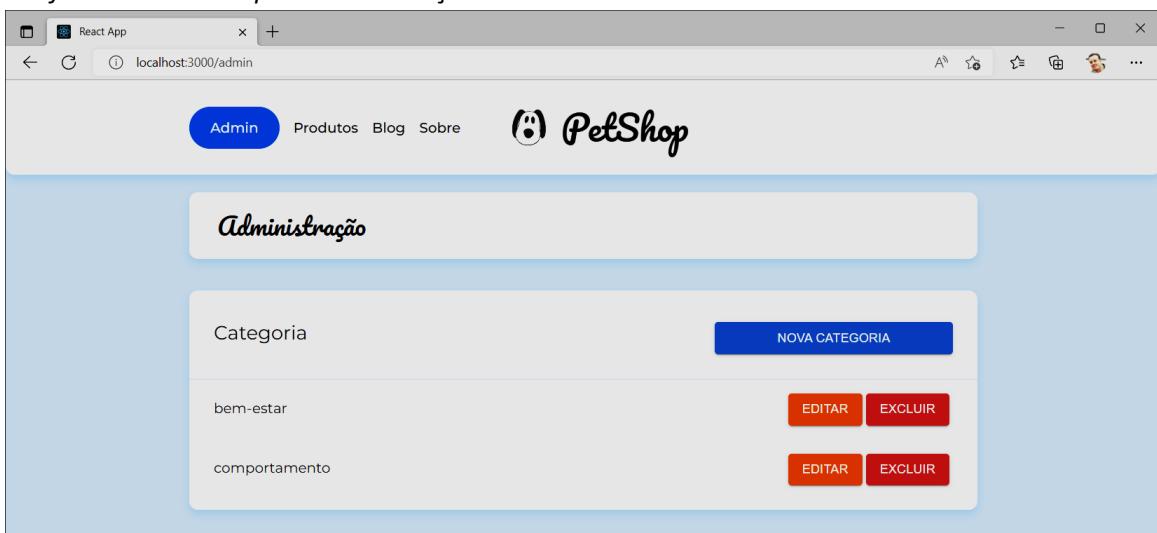
```
...
<tbody >
{
  categorias.map((categoria) => (
    <tr>
      <td className="tabela__coluna--m">
```

```

        <Link
      to={`/categoria/${categoria.id}`}>{categoria.nome}</Link>
    </td>
    <td colSpan="2" className="tabela__coluna--m
tabela__alinhamento--direita">
      <Link to={`/admin`} >
        <Button
          type="submit"
          variant="contained"
          color="warning"
          align="right"
        >
          Editar
        </Button>
      </Link>
      <Link to={`/admin`} >
        <Button
          type="submit"
          variant="contained"
          color="error"
          align="right"
          sx={{ margin: "0 0.25rem" }}
        >
          Excluir
        </Button>
      </Link>
    </td>
  </tr>
))
}
</tbody>
...

```

5. Confira o resultado após a renderização:



6. Após a renderização, o link para adicionar a categoria ainda não funciona, mas essa será a tarefa da próxima atividade.

Atividade 3 – Formulário e cadastro de novas Categorias

Objetivos:

- Criar um formulário utilizando a biblioteca de estilos
- Realizar a configuração para componentes realizam inserções na API

1. *Vamos criar um formulário simples para inserção de dados, para isso vamos utilizar a biblioteca de estilo, os States, e todos os recursos necessários para inserir dados em nossa API;*
2. *Crie um arquivo dentro de ‘admin/components’ com o nome de ‘FormCategoria’, e configure o componente com a codificação:*

- a. *Explore os componentes de input e button do ‘mui’;*

```
import React from "react";
import { Button, TextField } from "@mui/material";

const FormCategoria = () => {

    return (
        <main className="container flex flex--centro">
            <article className="cartao post">
                <h2 className="titulo-pagina">Cadastro de
Categorias</h2>
                <br />
                <form >
                    <TextField
                        id="standard-basic"
                        label="Categoria"
                        variant="filled"
                        fullWidth
                        required
                    />
                    <br />
                    <Button
                        type="submit"
                        variant="contained"
                        sx={{ marginTop: 1 }}
                        fullWidth
                    >
                        Cadastrar
                    </Button>
                </form>
            </article>
        </main>
    );
}

export default FormCategoria;
```

3. *Faça a indicação a rota no arquivo ‘App.js’:*

```
import React from 'react';
```

```

...
import Categoria from './paginas/Categoria';
import Admin from './paginas/admin/Admin';
import FormCategoria from './paginas/admin/components/FormCategoria';
...
<Route exact path='/'>
  <Home />
</Route>
<Route exact path='/admin'>
  <Admin />
</Route>
<Route exact path='/admin/NovaCategoria'>
  <FormCategoria />
</Route>
<Route path='/sobre'>
  <Sobre />
</Route>
...

```

4. Realizamos alguns testes antes de efetivamente enviar os dados para a API, para isso altere o código para captura dados e verificar via console do navegador:

```

import React, { useState } from "react";
import { Button, TextField } from "@mui/material";

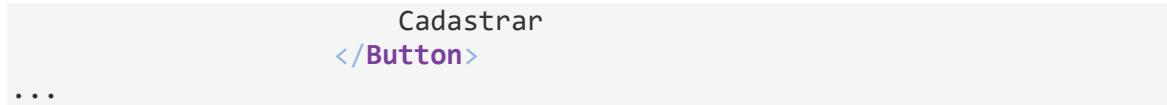
const FormCategoria = () => {
  const [nomeCategoria, setNomeCategoria] = useState('')

  const CadCategoria = (evento) => {
    evento.preventDefault()

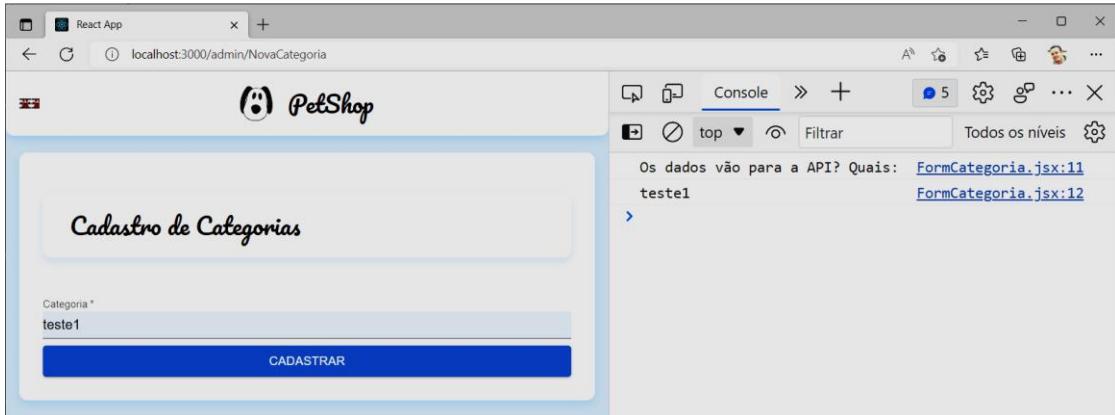
    console.log('Os dados vão para a API? Quais:')
    console.log(nomeCategoria)
  }

  return (
...
    <form onSubmit={CadCategoria}>
      <TextField
        value={nomeCategoria}
        onChange={evento =>
setNomeCategoria(evento.target.value)}
        id="standard-basic"
        label="Categoria"
        variant="standard"
        fullWidth
        required
      />
      <br />
      <Button
        type="submit"
        variant="contained"
        sx={{ margin: 1 }}
        fullWidth
      >

```



5. Abra o navegador, no modo desenvolvedor, aba 'console' e note que os dados estão sendo enviados, conforme na figura abaixo, se não houver o resultado esperado volte ao código para correções:



6. Faça agora os ajustes para que o formulário envie os dados para a API, modificando o código como abaixo:

```
import React, { useEffect } from "react";
import { Button, TextField } from "@mui/material";
import { useState } from "react";
import { api } from "../../api/api";
import { useHistory, useParams } from "react-router-dom";

const FormCategoria = () => {
    let history = useHistory()

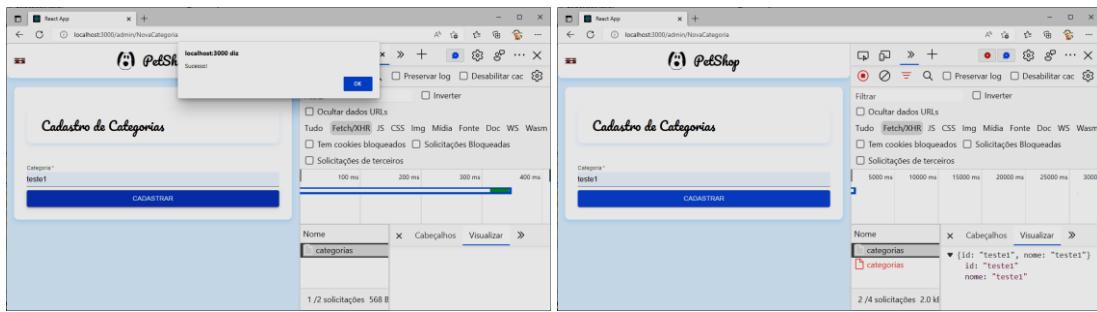
    const [nomeCategoria, setNomeCategoria] = useState('')

    const CadCategoria = (evento) => {
        evento.preventDefault()
        api.post(`categorias`, {
            id: nomeCategoria,
            nome: nomeCategoria,
            subcategorias: [subCategoria]

        })
        .then(() => {
            alert("Cadastro realizado com Sucesso!")
            history.push('/admin')
        })
    }

    return (
        ...
    )
}
```

7. Note que houve sucesso no envio das informações para a API:



8. Finalizamos a exibição dos posts por categorias.

Atividade 4 – Excluindo as categorias

Objetivos:

- Estabelecer os parâmetros de exclusão de dados da API

Após realizarmos inserções agora vamos para as exclusões de dados.

1. *Essa última atividade, finaliza de forma bem rápida e simples nosso capítulo, com a ativação do botão de exclusão da categoria. Abra o componente ‘ListaCatAdmin’, e ajuste o botão excluir:*

```
import React, { useEffect, useState } from "react";
import { Link } from "react-router-dom";
import { api, busca } from "../../../../../api/api";
...
{
  categorias.map((categoria) => (
    <tr key={categoria.id}>
      <td className="tabela_coluna--m">
        ...
        <Button
          type="submit"
          variant="contained"
          color="error"
          align="right"
          sx={{ margin: "0 0.25rem" }}
          onClick={() => excluir(categoria)}
        >
          Excluir
        </Button>
      ...
    </tr>
  ))
}
```

2. Agora vamos configurar os parâmetros de exclusão:

```
import React, { useEffect, useState } from "react";
...
const ListaCatAdmin = () => {
  ...
  const [categorias, setCategorias] = useState([])
  useEffect(() => {
    ...
  })
}
```

```

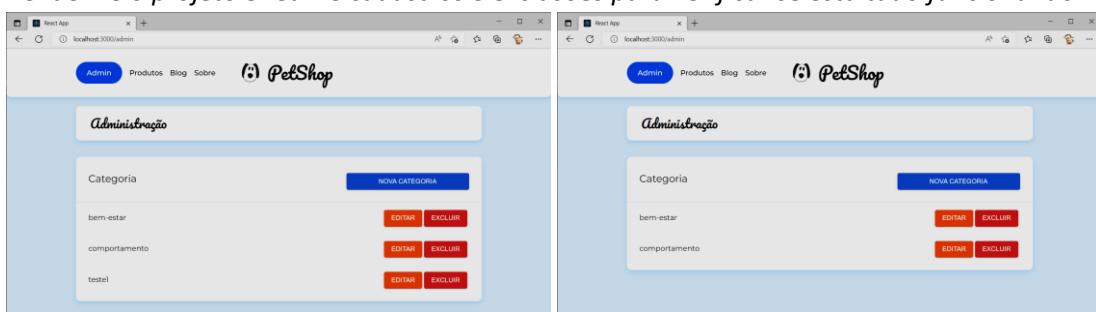
        busca(`/categorias`, setCategories)
    }, [])

    const excluir = (CategoriaDel) => {
        api.delete(`categorias/${CategoriaDel.id}`)
            .then(() => {
                const listaCategorias = categorias.filter(categoria =>
                    categoria.id !== CategoriaDel.id)
                setCategories([...listaCategorias])
            })
    }

    return (
        ...
    )
}

```

3. Renderize o projeto e realize cadastros e exclusões para verificar se está tudo funcionando:



4. Com isso finalizamos mais uma tarefa e um capítulo.

Atividade Extra

Objetivos:

- Rever e aprimorar os conhecimentos construídos.

Vamos trabalhar algumas atividades extra para rever e aprimorar o que trabalhamos *até aqui*:

1. *Como poderíamos aplicar esse mesmo conceito a outras áreas do projeto?*
2. *Pesquise mais sobre outras bibliotecas de estilos do React;*
3. *Encontre uma forma de exibir uma confirmação antes da exclusão de uma categoria;*

CAPÍTULO 7 – MAIS UM POUCO DE CRUD COM REACT

Já melhoramos as interfaces e simplificamos a construção de componente com as biblioteca de estilo e adicionamos e excluímos dados da API, utilizando React. Vamos continuar nessa linha de trabalho, agora alterando os dados das categorias.

Vamos expandir os nosso conhecimento, trabalhando com as subcategorias, relacionando e editando os dados, mantendo os conceitos de SPA e API, e melhorando aspectos do nosso CRUD React.

Objetivos:

- Alterar dados inseridos em uma API
- Relacionar dados de categorias e subcategorias;
- Exibir um componente relacional;
- Criar um formulário para edição de subcategorias;
- Relacionar dados em um formulário;
- Renderizar, visualizar e editar dados relacionados;

Atividade 1 – Alterando as Categorias

Objetivos:

- Alterar dados inseridos em uma API

Essa atividade vai nos permitir alterar os dados cadastrados nas Categorias. Vamos aos estudos?

1. Acesse o componente ‘petshop/src/paginas/admin/componentes>ListCatAdmin.jsx’, para ajustar o ‘Link’ do botão editar que criamos anteriormente:

```
...
<Link to={`/admin/${categoria.id}`}>
  <Button
    type="submit"
    variant="contained"
    color="warning"
    align="right"
  >
    Editar
  </Button>
</Link>
...
```

2. Abra o componente ‘petshop/src/paginas/admin/componentes/FormCategoria.jsx’, fique atento para configurar os elementos para atualização de dados, com o código:

```

import React, { useEffect } from "react";
...
import { useHistory, useParams } from "react-router-dom";

const FormCategoria = () => {
  let history = useHistory()
  const parametros = useParams()

  const [nomeCategoria, setNomeCategoria] = useState('')
  useEffect(() => {
    if (parametros.id) {
      api.get(`categorias/${parametros.id}`)
        .then(resposta => setNomeCategoria(resposta.data.nome))
    }
  }, [parametros])

  ...
  <Button
    type="submit"
    variant="contained"
    sx={{ marginTop: 1 }}
    fullWidth
  >
    Salvar
  </Button>
  ...
}

```

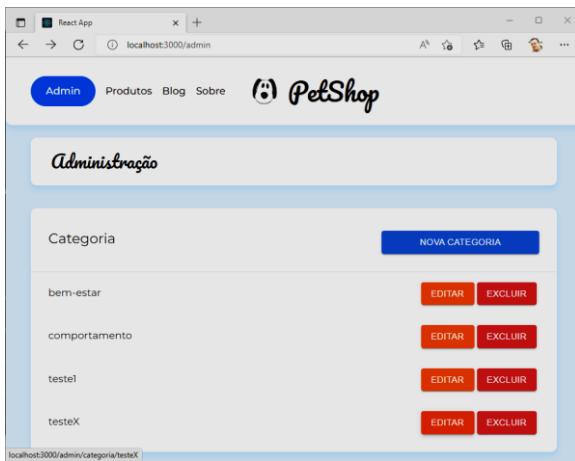
3. Acrescente ao arquivo 'App.js' a rota para um novo caminho, porém para o mesmo componente, o 'FormCategoria', que irá utilizar os recursos de state para realizar a alteração, para isso insira os códigos:

```

import React from 'react';
...
<Route exact path='/admin'>
  <Admin />
</Route>
<Route exact path='/admin/NovaCategoria'>
  <FormCategoria />
</Route>
<Route exact path='/admin/:id'>
  <FormCategoria />
</Route>
...

```

4. Renderiza o projeto e note que o botão editar disponibiliza o link com o id, quando passamos o mouse sobre ele:



5. Clique no botão editar e observe que o componente 'FormCategoria', ao ser carregado traz as informações da categoria inseridas no formulário, porém ao Salvar os dados eles não são atualizados, e sim inseridos como novos registros;

6. Para realizar a correção do componente 'FormCategoria', iremos estabelecer um parâmetro para realizar os métodos, através de um 'if', se o id já existir utilizaremos o put, que atualiza os dados, se não existir o state do form irá realizar uma inserção. Insira o código conforme o definido abaixo, relembrando os métodos:

- .get: listar;
- .post: inserir;
- .delete: excluir;
- .put: atualizar;

```
...
const CadCategoria = (evento) => {
  evento.preventDefault()

  if (parametros.id) {
    api.put(`categorias/${parametros.id}/`, {
      id: nomeCategoria,
      nome: nomeCategoria,
      subcategorias: []
    })
    .then(() => {
      alert("Sucesso na atualização!")
    })
  }
}
```

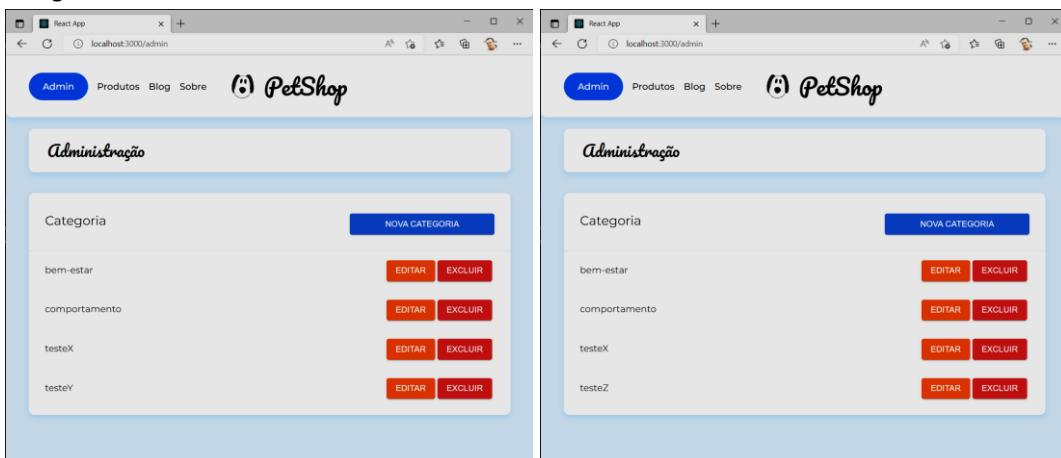
```

        history.push('/admin')
    })
} else {
    api.post(`categorias`, {
        id: nomeCategoria,
        nome: nomeCategoria,
        subcategorias: []
    })
    .then(() => {
        alert("Cadastro realizado com Sucesso!")
        history.push('/admin')
    })
}
}

return (
...

```

7. Após renderização realize a alteração dos itens para verificar o funcionamento, conforme a imagem abaixo:



8. Com isso finalizamos mais uma atividade.

Atividade 2 – Envio de dados do admin para subcategorias

Objetivos:

- Relacionar dados de categorias e subcategorias;
- Exibir um componente relacional;

Finalizamos as atividades de categorias, seguimos para as subcategorias que estão relacionadas e devem ser vinculadas e inseridas com os dados relacionados, esse será o desafio dessa atividade, incrementando pela ideia de exibir esses dados no mesmo espaço da SPA.

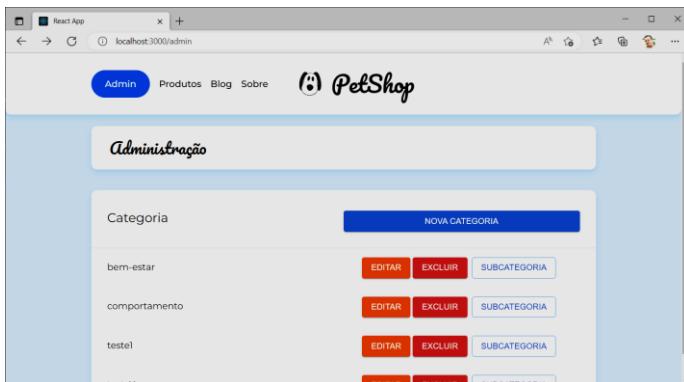
1. Abra o arquivo 'ListCatAdmin' e inicie a atividade criando o botão de SubCategoria logo abaixo do botão excluir, e fique atento aos detalhes do link, será importante no ajuste de rota:

```

<Link to={`/admin`} >
  <Button
    type="submit"
    variant="contained"
    color="error"
    align="right"
    sx={{ margin: "0 0.25rem" }}
    onClick={() => excluir(categoria)}
  >
    Excluir
  </Button>
</Link>
</td>
<td>
<Link to={`/admin/categoria/${categoria.id}`}>
  <Button
    type="submit"
    variant="outlined"
    color="primary"
    align="right"
    sx={{ margin: "0 0.25rem" }}
  >
    SubCategoria
  </Button>
</Link>

```

2. Renderize a página e faça um teste, por enquanto o botão irá direcionar a página 404 de categoria:



3. Vamos criar um componente para exibir as subcategorias, logo acima das categorias, vinculando o id, da categoria para exibição. Para essa tarefa, copie o arquivo 'src/paginas/Categoria.jsx' para a pasta 'src/paginas/admin', renomeando o arquivo como 'CatAdmin.jsx'. Fique atento aos ajustes de código, vamos iniciar pelas importações:

```

import { Button } from '@mui/material';
import { useEffect, useState } from 'react';
import { Link, Route, Switch, useParams, useRouteMatch } from 'react-router-dom';
import { busca } from '../../api/api';
import '../../assets/css/blog.css';

import ListaCatAdmin from './components/ListaCatAdmin';

```

...

4. Para os próximos ajustes no código procure exercitar as facilidades do VsCode, utilizando o recurso de alterar todas as instâncias:



5. Renomeie componente para 'CatAdmin' e faça os ajustes nos estados, conforme abaixo:

```
...
const CatAdmin = () => {
  const { id } = useParams()
  const { path } = useRouteMatch()
  const [subcategorias, setSubCategorias] = useState([])

  useEffect(() => {
    busca(`/categorias/${id}`, (categoria) => {
      setSubCategorias(categoria.subcategorias)
    })
  }, [id])
  return (
  ...
}
```

6. Crie uma tabela para inserir os elementos da subcategorias:

```
...
return (
<>
<div className="container">
  <h2 className="titulo-pagina">Administração</h2>
</div>
<div className='container'>
  <table className="tabela">
    <thead>
      <tr>
        <th colSpan="3" className="tabela__coluna--g">
          SubCategoria:
          <span className='cartao__titulo'>{id}</span>
        </th>
      </tr>
    </thead>
    <tbody>
      <tr> {
        subcategorias.map((subcategoria) => (
          <td className="tabela__coluna--m"
key={subcategoria.id}>
            {subcategoria}

```

```

        </td>))
    }
    <td>
        <Link to={`/admin/sub/${id}`}>
            <Button
                type="submit"
                variant="contained"
                fullWidth
                sx={{ marginTop: 1 }}
            >
                Editar SubCategorias
            </Button>
        </Link>
    </td>
</tr>
</tbody>
</table>
</div>
<ListaCatAdmin />
</>
)
}
}

export default CatAdmin

```

7. Se renderizar a página ela ainda não está exibindo as subcategorias:



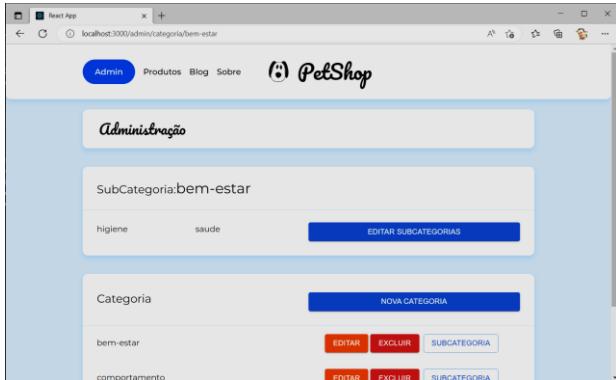
8. Para realizar as correções necessárias, abra o arquivo 'App.js' e defina a rota para ajustar a renderização:

```

import React from 'react';
...
import CatAdmin from './paginas/admin/CatAdmin';
...
<Route exact path='/admin/:id'>
    <FormCategoria />
</Route>
<Route path='/admin/categoria/:id'>
    <CatAdmin />
</Route>
<Route path='/sobre'>
    <Sobre />
</Route>
...

```

9. Verifique o resultado renderizando a página novamente, e escolhendo uma subcategoria. Faça os testes necessários e finalize mais uma atividade.



Atividade 3 – Formulário das SubCategorias

Objetivos:

- Criar um formulário para edição de subcategorias;
- Relacionar dados em um formulário;

Os Dados relacionados as subcategorias precisam transitar informações e permitir o vínculo as respectivas categorias esses é o próximo desafio.

1. Vamos criar um formulário para atualização das subcategorias, visto que elas estão ligadas as categorias. Crie um arquivo dentro de 'src/paginas/admin/componentes', e nomeie como 'FormSubCategoria'. Como de costume iniciamos pelos imports:

```
import React, { useEffect, useState } from 'react';
import { Button, TextField } from "@mui/material";
import { api } from "../../api/api";
import { useParams } from "react-router-dom";

...
```

2. Em seguida crie o componente 'FormSubCategoria' e alguns dos estados que serão utilizados, fique atento que para das elementos do formulário é necessários um componente para receber os dados:

```
const FormSubCategoria = () => {
  const parametros = useParams()

  const [nomeCategoria, setNomeCategoria] = useState([])
  const [subCategoria1, setSubCategoria1] = useState([])
  const [subCategoria2, setSubCategoria2] = useState([])

  useEffect(() => {
    if (parametros.id) {
      api.get(`categorias/${parametros.id}/`)
        .then(resposta => setNomeCategoria(resposta.data.nome))
    }
  }, [parametros])
```

```

        return (
    );
}

export default FormSubCategoria;

```

3. Crie o formulário para as subcategorias segue o código a seguir:

```

...
return (
<main className="container flex flex-centro">
  <article className="cartao post">
    <h3 className="titulo-pagina">
      Categoria: {parametros.id} / Subcategorias:
    </h3>
    <form>
      <TextField
        value={subCategoria1}
        onChange={(evento) =>
setSubCategoria1(evento.target.value)}
        id="outlined-basic"
        label="Subcategoria 1"
        variant="filled"
        fullWidth
        required
      />
      <br />
      <TextField
        value={subCategoria2}
        onChange={(evento) =>
setSubCategoria2(evento.target.value)}
        id="outlined-basic"
        label="Subcategoria 2"
        variant="filled"
        fullWidth
        sx={{ marginTop: 2 }}
      />
      <br />
      <Button
        type="submit"
        variant="contained"
        sx={{ marginTop: 2 }}
        fullWidth
      >
        Salvar
      </Button>
    </form>
  </article>
</main>
);
};

export default FormSubCategoria;

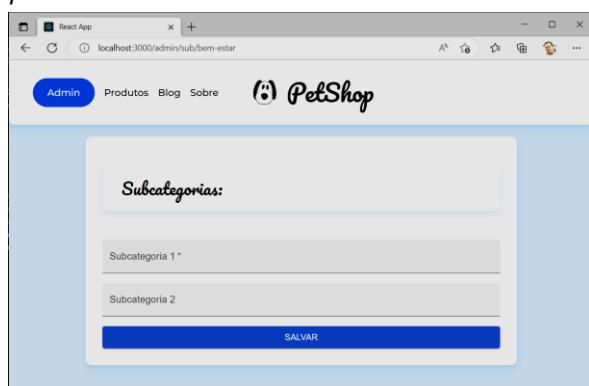
```

4. Ajuste a rota no arquivo 'App.js' para realizar a renderização do arquivo;

```
import React from 'react';
...
import CatAdmin from './paginas/admin/CatAdmin';
import FormSubCategoria from
'./paginas/admin/componentes/FormSubCategoria';
...
<Route path='/admin/categoria/:id'>
  <CatAdmin />
</Route>
<Route exact path='/admin/sub/:id'>
  <FormSubCategoria />
</Route>
<Route path='/sobre'>
  <Sobre />
</Route>
...

```

5. Renderize o projeto e verifique o visual do formulário, ele ainda não está funcional, essa será a próxima atividade:



6. Finalizamos o formulário, vamos aos dados.

Atividade 4 – Alterando os dados das subcategorias

Objetivos:

- Renderizar, visualizar e editar dados relacionados;

Nossa tarefa agora é fazer com que os dados que inserirmos no formulários sejam editados em sua respectiva categoria:

1. Abra o arquivo 'FormSubCategoria' e vamos codificar para exibir os dados da categoria e subcategorias que já estão na API, para que possamos identificar antes de fazer qualquer alteração. Fique atento e siga os códigos:

```
import React, { useEffect, useState } from "react";
...
import { api, busca } from "../../api/api";
```

```

const FormSubCategoria = () => {
  const parametros = useParams();

  const [ nomeCategoria, setNomeCategoria ] = useState([]);
  const [ subcategorias, setSubCategorias ] = useState([])
  const [ subCategoria1, setSubCategoria1 ] = useState([]);
  const [ subCategoria2, setSubCategoria2 ] = useState([]);

  useEffect(() => {
    busca(`/categorias/${parametros.id}`, (categoria) => {
      setSubCategorias(categoria.subcategorias)
    })
  }, [parametros])

  useEffect(() => {
    if (parametros.id) {
      api.get(`categorias/${parametros.id}`)
        .then((resposta) =>
      setNomeCategoria(resposta.data.nome));
    }
  }, [parametros]);
  return (
    <main className="container flex flex--centro">
      <article className="cartao post">
        <h3 className="titulo-pagina">Categoria:
{parametros.id} / Subcategorias:</h3>
        <form>
          ...

```

2. Renderize o projeto novamente, note que ainda não é possível realizar a edição das subcategorias:



3. Para realizar o cadastro ou edição das subcategorias, vamos inserir o “Put” e fazer os últimos ajustes no código. Realize a codificação como demonstrado:

```

import React, { useEffect } from "react";
...
const FormSubCategoria = () => {
  let history = useHistory()

```

```

const parametros = useParams()

useEffect(() => {
...
}, [parametros])

const CadCategoria = (evento) => {
    evento.preventDefault()

    if (parametros.id) {
        api.put(`/categorias/${parametros.id}/`, {
            id: nomeCategoria,
            nome: nomeCategoria,
            subcategorias: [subCategoria1, subCategoria2]
        })
        .then(() => {
            alert("Sucesso na atualização!")
            history.push('/admin/')
        })
    }
}

return (
...
)
<form onSubmit={CadCategoria}>
...

```

4. Renderize o projeto novamente, realize o cadastramento de algumas subcategorias, e outros testes, corrija possíveis falhas e note que por não trabalharmos com banco de dados, existem algumas limitações, em relação a relacionamentos, por exemplo.
5. Anote suas considerações, comente e debata com os colegas e procure se aprofundar nos estudos. Vale a pena! Com isso finalizamos mais uma tarefa e um capítulo.

Atividade Extra

Objetivos:

- Rever e aprimorar os conhecimentos construídos.

Vamos trabalhar algumas atividades extra para rever e aprimorar o que trabalhamos até aqui:

1. Como poderíamos aplicar esse mesmo conceito a outras áreas do projeto?
2. Pesquise mais sobre bancos de dados e os que estão relacionados ao React;
3. Desafie os seus conhecimento até aqui e encontre um meio de exibir as subcategorias em outra página;

CAPÍTULO 8 – FINALIZANDO O CRUD COM REACT

Na reta final vamos integrar uma barra de navegação a área administrativa, refazer o mapeamentos de rota, fazer a lista de posts, bem como a inserção, exclusão e atualização dos mesmo. Nossa quantas tarefas. Vamos lá!

Objetivos:

- Criar a barra de navegação da área administrativa utilizando rotas aninhadas
- Criar componentes para listar postagens revisando conceitos trabalhados;
- Criar um formulário para inserção e edição de posts;
- Utilizar as categorias como dados dinâmicos do select;
- Configurar formulário para inserir e editar posts;
- Finalizar o projeto PetShop.

Atividade 1 – Navegação aninhada

Objetivos:

- Criar a barra de navegação da área administrativa utilizando rotas aninhadas

Essa atividade vai nos permitir alterar os dados cadastrados nas Categorias. Vamos aos estudos?

1. *Copie o arquivo ‘cap08_arq_trabalho/catiorrinhoAdmin.svg’ do material didático, para o diretório, ‘petshop/src/assets/img/’;*
2. *Copie o arquivo ‘cap08_arq_trabalho/NavAdmin.jsx’ e ‘cap08_arq_trabalho/textarea.css’ do material didático, para o diretório, ‘petshop/src/paginas/admin/componentes/’ abra e faça uma pequena análise do código, que é bem familiar depois de tudo que já foi visto até aqui:*
3. *Revise e organize as rotas do arquivo ‘App.js’, ‘aninhando’ a área administrativa, para exibir a sua barra de navegação em todos os componentes relacionados:*

```
import React from 'react';
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';

import Home from './paginas/Home';
import Sobre from './paginas/Sobre';
import Pagina404 from './paginas/Pagina404';
import Cabecalho from './components/Cabecalho';
import Post from './paginas/Post';
import Categoria from './paginas/Categoria';
```

```

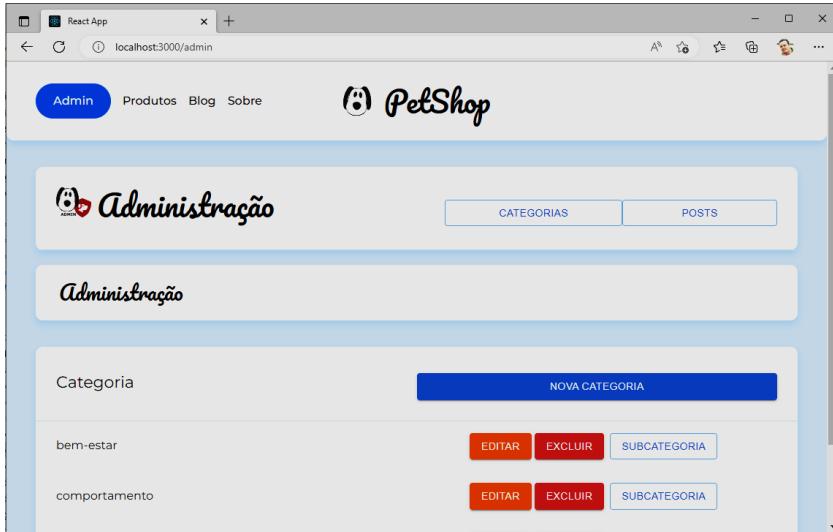
import Admin from './paginas/admin/Admin';
import FormCategoria from './paginas/admin/components/FormCategoria';
import CatAdmin from './paginas/admin/CatAdmin';
import FormSubCategoria from
'./paginas/admin/components/FormSubCategoria';
import NavAdmin from './paginas/admin/components/NavAdmin';

function App() {
  return (
    <Router>
      <Cabecalho />
      <Switch>
        <Route exact path='/'>
          <Home />
        </Route>
        <Route path='/sobre'>
          <Sobre />
        </Route>
        <Route path='/categoria/:id'>
          <Categoria />
        </Route>
        <Route path='/posts/:id'>
          <Post />
        </Route>
        <Route path='/admin/'>
          <NavAdmin />
          <Switch>
            <Route exact path='/admin'>
              <Admin />
            </Route>
            <Route exact path='/admin/NovaCategoria'>
              <FormCategoria />
            </Route>
            <Route exact path='/admin/categoria/:id'>
              <FormCategoria />
            </Route>
            <Route exact path='/admin/categoria/sub/:id'>
              <CatAdmin />
            </Route>
            <Route exact path='/admin/categoria/sub/form/:id'>
              <FormSubCategoria />
            </Route>
            <Route path='*'>
              <Pagina404 />
            </Route>
          </Switch>
        </Route>
        <Route path='*'>
          <Pagina404 />
        </Route>
      </Switch>
    </Router>
  );
}

```

```
export default App;
```

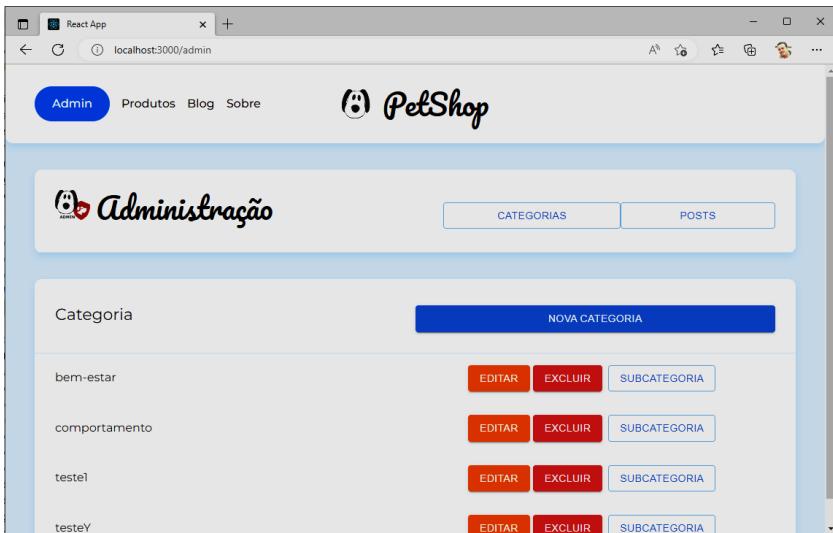
4. Renderiza o projeto e note que são necessárias algumas correções na composição da página e alguns links que não estão funcionando adequadamente:



5. No arquivo 'Admin.jsx', apague as linhas riscadas, como mostradas abaixo:

```
return (
  <main>
    <div className="container">
      <h2 className="titulo_pagina">Administração</h2>
    </div>
    <ListaCatAdmin />
  </main>
);
```

6. Observe o resultado, após a renderização:



7. No arquivo 'ListCatAdmin.jsx', corrija os link 'EDITAR' e 'SUBCATEGORIA', observando o código abaixo:

```
...
  <Link to={`/admin/categoria/${categoria.id}`}>
    <Button>
```

```

        type="submit"
        variant="contained"
        color="warning"
        align="right"
      >
    Editar
  </Button>
</Link>
...
<td>
  <Link to={`/admin/categoria/sub/${categoria.id}`}>
    <Button
      type="submit"
      variant="outlined"
      color="primary"
      align="right"
      sx={{ margin: "0 0.25rem" }}
    >
      SubCategoria
    </Button>
  </Link>
</td>
...

```

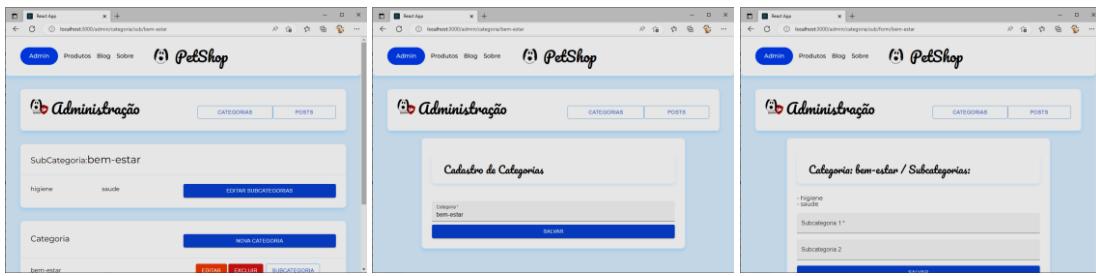
8. No arquivo 'CatAdmin.jsx', retire o código riscado, conforme demonstrado, e corrija também os links do botão, 'Editar SubCategorias', ajustando o endereçamento:

```

...
return (
<>
<div className="container">
  <h2 className="titulo-pagina">Administração</h2>
</div>
<div className='container'>
  <table className="tabela">
...
  <Link to={`/admin/categoria/sub/form/${id}`}>
    <Button
      type="submit"
      variant="contained"
      fullWidth
      sx={{ marginTop: 1 }}
    >
      Editar SubCategorias
    </Button>
  </Link>
...

```

9. Com isso criamos um barra de navegação com rotas aninhadas, corrigimos alguns endereçamentos e finalizamos mais uma atividade.



Atividade 2 – Lista dos Posts

Objetivos:

- Criar componentes para listar postagens revisando conceitos trabalhados;

O área do ‘CRUD’ de categorias está pronta e a barra de navegação funcional, vamos iniciar a criação dos elementos para manipular os posts. Foco na tarefa!

1. *Crie o arquivo ‘ListPostAdmin.jsx’, dentro do diretório ‘src/paginas/admin/components’ e inicie a atividade desenvolvendo o componente que irá listar os posts disponíveis no projeto:*

```
import { Button } from "@mui/material";
import { useEffect, useState } from "react";
import { api, busca } from "../../../../../api/api";
import { Link } from "react-router-dom";

const ListaPostAdmin = () => {
  const [posts, setPosts] = useState([]);

  useEffect(() => {
    busca(`posts`, setPosts);
  }, []);

  const excluir = (PostDel) => {
    api.delete(`posts/${PostDel.id}`).then(() => {
      const listaPosts = posts.filter((post) => post.id !== PostDel.id);
      setPosts([...listaPosts]);
    });
  };

  return (
    <>
      <div className="container">
        <section>
          <table className="tabela">
            <thead>
              <tr>
                <th className="tabela_coluna--p">Categoria</th>
                <th className="tabela_coluna--m">Título</th>
                <th className="tabela_coluna--m">Posts</th>
                <th
                  className="tabela_coluna--p"></th>
            </thead>
            <tbody>
              {posts.map((post) => (
                <tr key={post.id}>
                  <td>{post.categoria}</td>
                  <td>{post.titulo}</td>
                  <td>{post.posts}</td>
                  <td>
                    <Link to={`/admin/posts/editar/${post.id}`}>Editar</Link>
                    <span>|</span>
                    <Link to={`/admin/posts/deletar/${post.id}`}>Deletar</Link>
                  </td>
                </tr>
              ))}
            </tbody>
          </table>
        </section>
      </div>
    </>
  );
}
```

```

        */}{ " "}
            <th className="tabela__coluna--p tabela__alinhamento--direita">
                <Link to="/admin/posts/NovoPost">
                    <Button type="submit" variant="contained"
fullWidth>
                        Novo Post
                    </Button>
                </Link>
            </th>
        </tr>
    </thead>
    <tbody>
        {posts.map((post) => (
            <tr key={post.id}>
                <td>{post.categoria}</td>
                <td>
                    <Link to={`/posts/${post.id}`}>{post.title}</Link>
                </td>
                <td>{post.metadescription}</td>

                <td>
                    <Link to={`/admin/posts/${post.id}`}>
                        <Button
                            type="submit"
                            variant="contained"
                            color="warning"
                            align="right"
                            sx={{ margin: "0.25rem 0" }}
                            fullWidth
                        >
                            Editar
                        </Button>
                    </Link>
                    <br />
                    <Link to={`/admin/posts`}>
                        <Button
                            type="submit"
                            variant="contained"
                            color="error"
                            align="right"
                            sx={{ margin: "0.25rem 0" }}
                            onClick={() => excluir(post)}
                            fullWidth
                        >
                            Excluir
                        </Button>
                    </Link>
                </td>
            </tr>
        )))
    </tbody>
</table>
</section>
</div>

```

```

        </>
    );
};

export default ListaPostAdmin;

```

2. Em seguida, crie um arquivo, chamado 'PostAdmin.jsx', dentro do diretório 'src/páginas/admin', seguindo o código abaixo:

```

import React from 'react';
import '../assets/css/blog.css';

import ListaPostAdmin from './components/ListPostAdmin';

const PostAdmin = () => {

    return (
        <>
            <ListaPostAdmin />
        </>
    )
}

export default PostAdmin

```

3. Para realizar as configurações de rota necessárias, abra o arquivo 'App.js', defina mais uma rota para área administrativa, portanto aproveitando a barra de navegação na renderização, para isso adicione o código abaixo:

```

import React from 'react';
...
import CatAdmin from './paginas/admin/CatAdmin';
...
<Route exact path='/admin/categoria/:id'>
    <FormCategoria />
</Route>
<Route exact path='/admin/posts/'>
    <PostAdmin />
</Route>
<Route exact path='/admin/categoria/sub/:id'>
...

```

4. Verifique o resultado renderizando a página novamente, nesse momento apenas o botão excluir está funcionando, na próxima atividade ajustaremos os demais.

Atividade 3 – Formulário dos Posts com select dinâmico

Objetivos:

- Criar um formulário para inserção e edição de posts;
- Utilizar as categorias como dados dinâmicos do select;

Já temos todos os posts listados, mas que tal criar um componente para inserir mais, e alterar os existentes, e ainda como bônus, criar um ‘select’ que consulta as categorias. Vai ser muito legal essa atividade.

1. *Vamos criar um formulário para inserção e atualização dos posts, consultando as categorias. Crie um arquivo dentro de ‘src/paginas/admin/components’, e nomeie como ‘FormPost.jsx’. Como de costume iniciamos pelos imports:*

```
import { Box, Button, FormControl,InputLabel,MenuItem,Select,TextField } from "@mui/material";
import { useEffect, useState } from "react";
import { useParams } from "react-router-dom";
import { api } from "../../api/api";
import "../components/textarea.css";
```

2. *Crie o componente ‘FormPost’. Revise a consulta e inserção de dados através do uso de States:*

```
const FormPost = () => {
  const parametros = useParams();

  const [title, setTitle] = useState("");
  const [metadescription, setMetadescription] = useState("");
  const [body, setBody] = useState("");
  const [categoria, setCategoria] = useState("");

  const [categorias, setCategorias] = useState([]);
  useEffect(() => {
```

```

    api.get("categorias/").then((resposta) =>
setCategorias(resposta.data));
}, []);

return (
<>
<main className="container flex flex--centro">
<article className="cartao post">
<h2 className="titulo-pagina">Formulário de Posts</h2>
<Box
sx={{
    display: "flex",
    flexDirection: "column",
    alignItems: "center",
    flexGrow: 1,
}}
>
<Box sx={{ width: "100%" }}>
<form>
<TextField
    value={title}
    onChange={(evento) => setTitle(evento.target.value)}
    label="Título"
    variant="filled"
    fullWidth
    required
    margin="dense"
/>
<TextField
    value={metadescription}
    onChange={(evento) =>
setMetadescription(evento.target.value)}
    label="Subtítulo"
    variant="filled"
    fullWidth
    required
    margin="dense"
/>
<br />
<br />
<label className="" for="descricao">
    Descrição
</label>
<br />
<textarea
    className="textarea filled"
    name="descricao"
    id="descricao"
    placeholder="Descrição"
    rows="4"
    value={body}
    onChange={(evento) => setBody(evento.target.value)}
></textarea>
<FormControl margin="dense" fullWidth>

```

```

        <InputLabel id="select-
categoria">Categoria</InputLabel>
        <Select
            labelId="select-categoria"
            value={categoria}
            onChange={(evento) =>
setCategoria(evento.target.value)}
        >
            {categorias.map((categoria) => (
                <MenuItem key={categoria.id}>
                    {categoria.nome}
                </MenuItem>
            )));
        </Select>{" "}
    </FormControl>

    <br />

    <Button
        sx={{ marginTop: 1 }}
        type="submit"
        fullWidth
        variant="contained"
    >
        Salvar
    </Button>
    </form>
</Box>
</Box>
</article>
</main>
</>
);
};

export default FormPost;

```

3. Ajuste as rotas no arquivo 'App.js' para visualizar como está o layout que acabou de desenvolver:

```

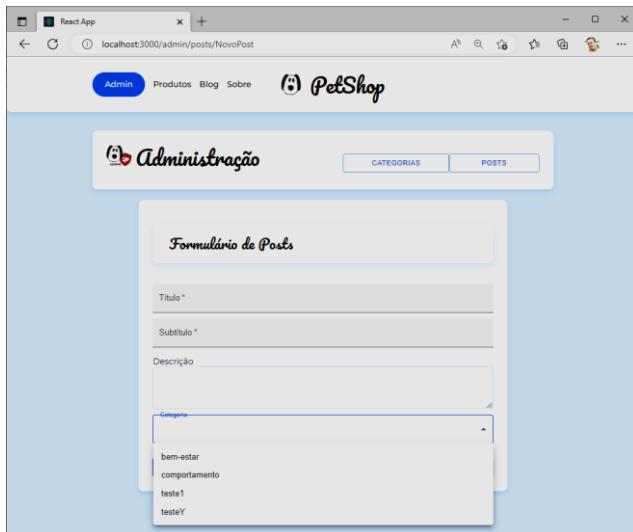
import React from 'react';
...
import CatAdmin from './paginas/admin/CatAdmin';
...
<Route exact path='/admin/categoria/:id'>
    <FormCategoria />
</Route>
<Route exact path='/admin/posts/NovoPost'>
    <FormPost />
</Route>
<Route exact path='/admin/posts/:id'>
    <FormPost />
</Route>
<Route exact path='/admin/posts/'>
    <PostAdmin />

```

```
</Route>
...

```

4. Renderize o projeto, verifique possíveis falhas e compare o resultado com a imagem a seguir:



5. Finalizamos o formulário, vamos aos dados na próxima atividade.

Atividade 4 – Finalizando o admin, os Posts e projeto

Objetivos:

- Configurar formulário para inserir e editar posts;
- Finalizar o projeto PetShop.

Falta pouco e já vai dando aquela vontade de aprender mais, pensar em outros cursos, mas ainda não acabou, vamos primeiro a edição, para trazer os dados par preencher o formulário e em seguida inserção para finalizar nosso projeto.

1. Na primeira etapa das atividades vamos exibir os dados dos posts que já estão na API, no componente ‘FormPost’, ligando cada elemento do formulário aos seus respectivos dados. Fique atento e siga os códigos:

```
import { Box, Button, FormControl,InputLabel,MenuItem,Select,TextField } from "@mui/material";
import { useEffect, useState } from "react";
import { useHistory, useParams } from "react-router-dom";
import { api } from "../../api/api";
import "../components/textarea.css";

const FormPost = () => {
  let history = useHistory();
  const parametros = useParams();

  const [title, setTitle] = useState("");
  ...
}
```

```

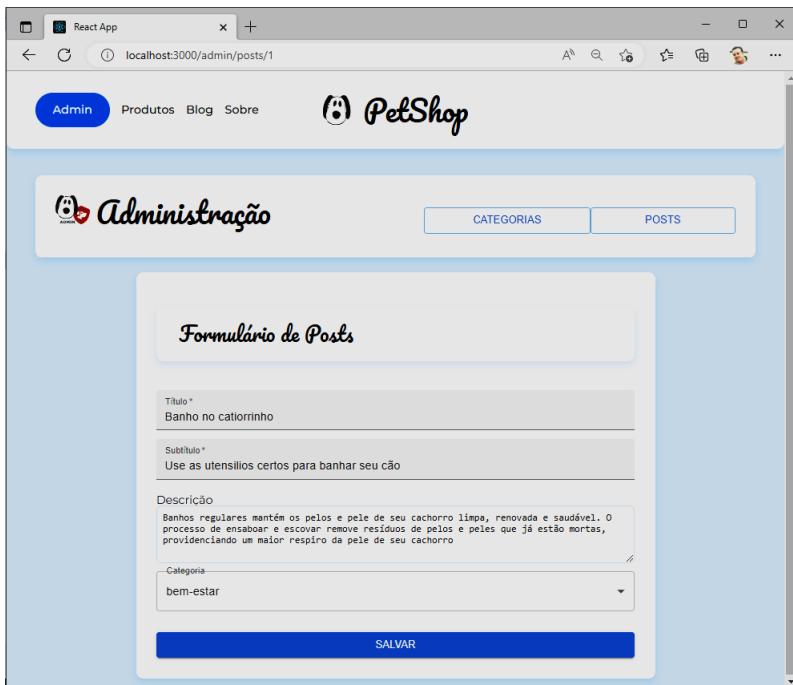
const [metadescription, setMetadescription] = useState("");
const [body, setBody] = useState("");
const [categoria, setCategoria] = useState("");
useEffect(() => {
  if (parametros.id) {
    api
      .get(`posts/${parametros.id}/`)
      .then((resposta) => setTitle(resposta.data.title));
    api
      .get(`posts/${parametros.id}/`)
      .then((resposta) =>
        setMetadescription(resposta.data.metadescription));
    api
      .get(`posts/${parametros.id}/`)
      .then((resposta) => setBody(resposta.data.body));
    api
      .get(`posts/${parametros.id}/`)
      .then((resposta) => setCategoria(resposta.data.categoria));
  }
}, [parametros]);

const [categorias, setCategorias] = useState([]);
useEffect(() => {
  api.get("categorias/").then((resposta) =>
    setCategorias(resposta.data));
}, []);

return (
  <>
    <main className="container flex flex--centro">
      <article className="cartao post">
        <h2 className="titulo-pagina">Formulário de Posts</h2>
        <Box
          sx={{
            display: "flex",
            flexDirection: "column",
            alignItems: "center",
            flexGrow: 1,
          }}
        >
          <Box sx={{ width: "100%" }}>
            <form>
              ...

```

2. Renderize novamente o projeto, navegue até a lista de posts, e clique em editar, para verificar seu código obteve sucesso e os dados do post estão sendo exibidos. Note que ainda não será possível atualizar nem inserir novos dados.



3. Para realizar a inserção e atualização vamos revisar os métodos 'post' e 'put', configurando o componente para essas ações. Realize os ajustes do código conforme a seguir:

```
import { Box, Button, FormControl,InputLabel,MenuItem,Select,TextField } from "@mui/material";
...
const FormPost = () => {
  let history = useHistory();
  const parametros = useParams();

  ...

  const CadPost = (evento) => {
    evento.preventDefault();

    if (parametros.id) {
      api
        .put(`/posts/${parametros.id}`, {
          title: title,
          metadescription: metadescription,
          body: body,
          categoria: categoria
        })
        .then(() => {
          alert("Sucesso na atualização!");
          history.push("/admin/posts");
        });
    } else {
      api
        .post(`/posts`, {
          title: title,
          metadescription: metadescription,
          body: body,
          categoria: categoria
        })
        .then(() => {
          alert("Sucesso na inserção!");
          history.push("/admin/posts");
        });
    }
  };
}
```

```

        })
        .then(() => {
            alert("Cadastro realizado com Sucesso!");
            history.push("/admin/posts/");
        });
    }
};

return (
    <>
    ...
    <Box sx={{ width: "100%" }}>
        <form onSubmit={CadPost}>
            <TextField
    ...

```

4. Realize os teste necessários, anote suas considerações, comente e debata com os colegas e procure se aprofundar nos estudos. Vale a pena! Com isso finalizamos mais uma tarefa, um capítulo, um projeto e um momento superimportante de aprendizado. #gratidão

Atividade Extra

Objetivos:

- Rever e aprimorar os conhecimentos construídos.

Vamos trabalhar algumas atividades extra para rever e aprimorar o que trabalhamos *até aqui*:

1. Que tal continuar estudando e desenvolver as páginas para exibir e administrar os produtos do PetShop?
2. Pesquise mais sobre React com acesso a banco de dados e React Native;
3. Desafie os seus conhecimento e encontre um meio de utilizar as suas habilidades para desenvolver outros projetos com os mesmo conceitos, uma agência de viagens, por exemplo;
4. Que legal seria que os posts tivessem imagens. Que tal estudar e descobrir como? Se descobrir me manda.

Deixo uma última mensagem nesse material. Que esse não seja o último desafio, que sigam sempre aprendendo, criando e evoluindo. Criando um futuro melhor!

Desde já agradeço a atenção, respeito e confiança no nosso trabalho.