

ReactJS

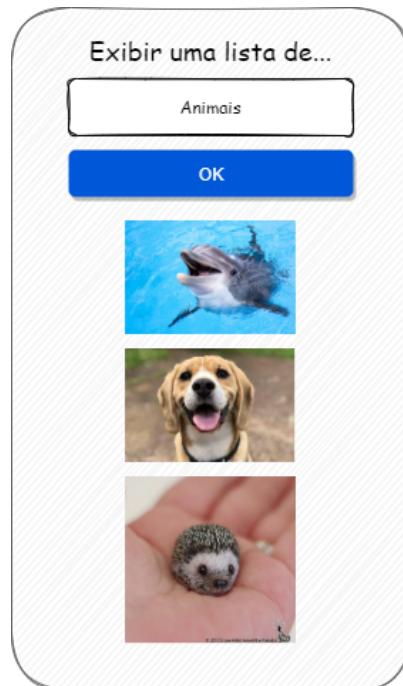
Forms, entrada de dados, eventos, acesso a APIs e listas

1 Introdução

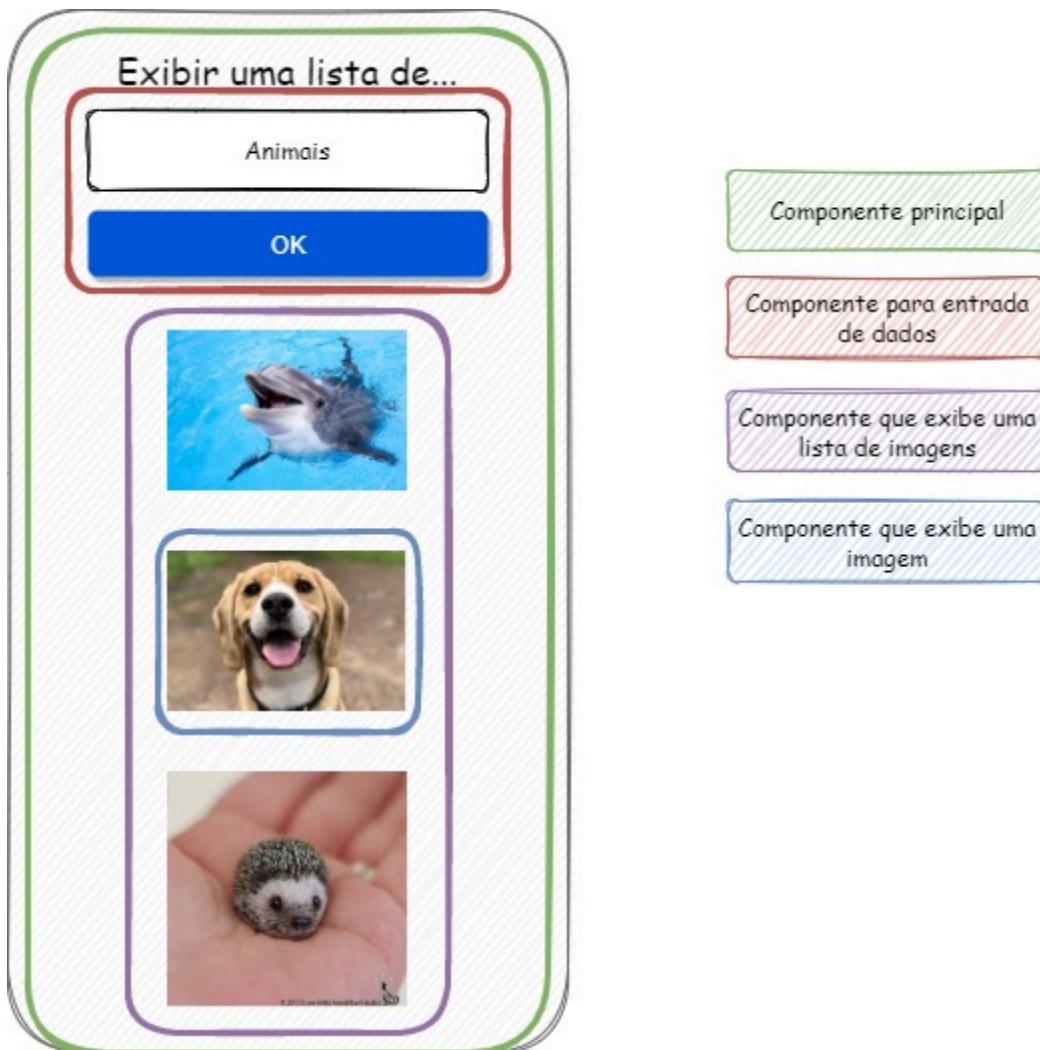
Neste material estudaremos os seguintes tópicos

- Forms & entrada de dados realizada pelo usuário
- Tratamento de eventos
- Acesso a APIs com requisições HTTP
- Exibição de coleções de dados

Veja a aplicação a ser desenvolvida. Ela oferece um campo em que o usuário pode digitar um termo de busca qualquer como "animais", "pessoas", "roupas". A seguir, se encarrega de exibir uma lista de figuras condizentes com a sua busca.



Os componentes que implementaremos são os seguintes.



2 Desenvolvimento

2.1 (Novo projeto e componente principal) Crie um projeto com

```
npx create-react-app nome-projeto
```

Use

```
cd nome-projeto
```

para navegar até o diretório em que se encontra o seu projeto. Use

```
code .
```

para obter uma instância do VS Code vinculada a esse diretório. No VS Code, clique **Terminal >> New Terminal** para obter um novo terminal interno do VS Code, o que simplifica o trabalho. Neste terminal, digite

```
npm start
```

para colocar a aplicação em funcionamento. Uma janela do seu navegador padrão deve ser aberta fazendo uma requisição a **localhost:3000**.

Apague todos os arquivos existentes na pasta **src**. A seguir, crie uma pasta chamada **components**, subpasta de **src**. Definiremos todos os componentes dessa aplicação em arquivos dentro dela.

Na pasta **components**, crie um arquivo chamado **App.js**. Veja seu conteúdo inicial. Optamos por um componente funcional pois não precisaremos de estado **inicialmente**.

Nota. Lembre-se que, até então, não estudamos sobre os Hooks do React, então, se precisarmos de componentes com estado, utilizaremos aqueles definidos por meio de classes.

```
import React from 'react'

export default () => {
    return (
        <div>
            <h1>Exibir uma lista de...</h1>
        </div>
    )
}
```

Crie um arquivo chamado **index.js** na pasta **src**, fora da pasta **components**. Veja seu conteúdo a seguir. Note que estamos “montando” o componente App tal qual feito em outras aplicações.

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

ReactDOM.render(
    <App />,
    document.querySelector('#root')
)
```

2.2 (Dependências) Utilizaremos componentes da biblioteca PrimeReact e os utilitários e grid system da PrimeFlex. As suas respectivas documentações podem ser visitadas a seguir.

PrimeReact
<https://www.primefaces.org/primereact/>

PrimeFlex
<https://www.primefaces.org/primeflex/>

Faça a instalação da PrimeReact com

```
npm install primereact
npm install primeicons
npm install react-transition-group
```

A PrimeFlex pode ser instalada com

```
npm install primeflex
```

A seguir, importe o CSS da PrimeReact, PrimeIcons, PrimeFlex e um dos temas descritos na documentação, na página **Get Started**. Isso pode ser feito no arquivo **index.js**. Veja.

```
import React from 'react'
import ReactDOM from 'react-dom'
import App from './components/App'
import 'primereact/resources/primereact.min.css'
import 'primeicons/primeicons.css'
import 'primeflex/primeflex.css'
import 'primereact/resources/themes/bootstrap4-light-purple/theme.css'
ReactDOM.render(
  <App />,
  document.querySelector('#root')
)
```

2.3 (Componente para a busca) Conforme o usuário digita, desejamos armazenar o conteúdo presente no campo de entrada para que possamos realizar a busca quando o botão for clicado. Esse conteúdo será armazenado no estado do componente, razão pela qual ele será definido por meio de uma classe. Crie um arquivo chamado **Busca.js** na pasta **src/components**. Veja seu conteúdo inicial. Repare na definição de estado.

```
import React, { Component } from 'react'
export default class Busca extends Component {
  state = {
    termoDeBusca: ''
  }
  render() {
    return (
      <div>

        </div>
      )
    }
}
```

A princípio, o componente de busca exibirá um componente textual e um botão, um abaixo do outro. Veja seu método render, os imports necessários e um valor padrão para o props utilizado na exibição da dica. Definimos, também, uma função que é executada toda vez que o usuário atualiza o campo de busca. Ela exibe o valor digitado no console do navegador.

```
import React, { Component } from 'react'
import { InputText } from 'primereact/inputtext'
import { Button } from 'primereact/button'
export default class Busca extends Component {
    state = {
        termoDeBusca: ''
    }
    onTermoAlterado = (event) => {
        console.log(event.target.value)
    }
    render() {
        return (
            // empilhando os filhos
            <div className="flex flex-column">
                {/* ícone à esquerda, largura máxima */}
                <span className="p-input-icon-left w-full">
                    <i className="pi pi-search"/>
                    <InputText
                        //largura máxima
                        className="w-full"
                        onChange={this.onTermoAlterado}
                        placeholder={this.props.dica}
                    />
                </span>
                <Button
                    label="OK"
                    className="p-button-outlined mt-2"
                />
            </div>
        )
    }
}

Busca.defaultProps ={
    dica: 'Digite algo que deseja ver...'
}
```

Cabe ao componente App exibir o conteúdo principal da aplicação e ele o fará utilizando o grid system da PrimeReact. Veja a forma como ele faz isso, incluindo o componente Busca.

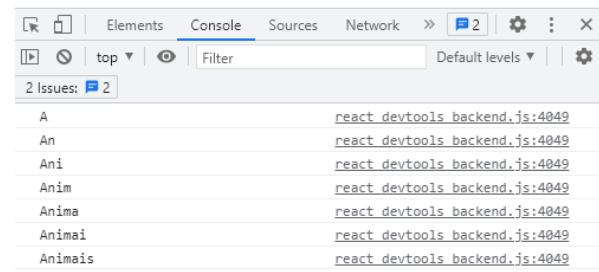
```
import React, { Component } from 'react'
import Busca from './Busca'

class App extends Component{

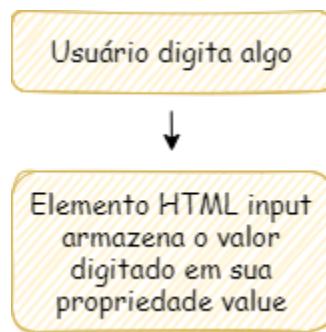
    render(){
        return (
            <div className="grid justify-content-center m-auto w-9 border-round border-1 border-400">
                <div className='col-12'>
                    <h1>Exibir uma lista de...</h1>
                </div>
                <div className="col-8">
                    <Busca />
                </div>
            </div>
        )
    }
}

export default App
```

Execute a aplicação e digite alguns caracteres no campo de texto. Visualize o resultado esperado no Chrome Dev Tools (CTRL+SHIFT+I), que deve ser parecido com esse aqui.



2.4 (Componentes controlados e não controlados) Componentes ReactJS podem ser classificados como controlados e não controlados. Para entender a diferença, considere o que acontece conforme o usuário digita no campo textual. Lembre-se de que ainda não estamos utilizando o estado do componente.



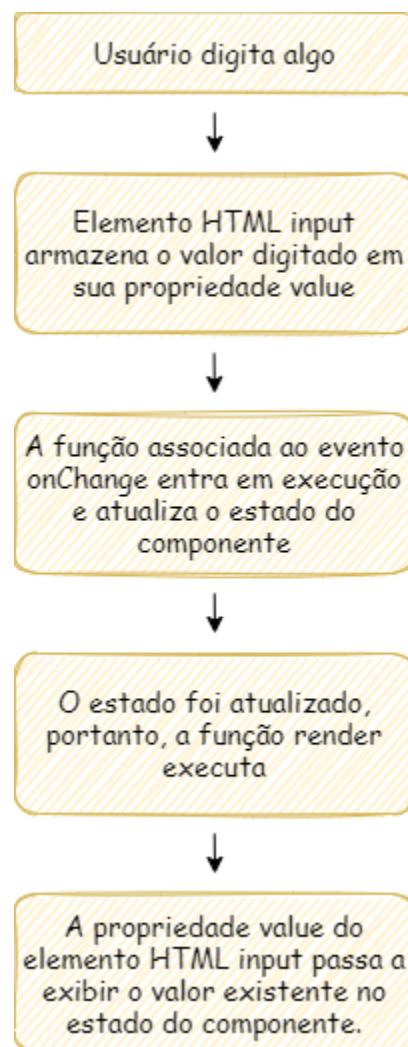
O fato importante a se observar é o local em que os dados digitados pelo usuário são armazenados: eles são armazenados no elemento HTML, na árvore DOM. Ou seja, não são armazenados ou controlados pelo componente React. Por essa razão, dizemos que este componente é **não controlado**. Neste caso, a **Source of Truth** é a própria árvore DOM, ou seja, a estrutura a partir da qual os dados podem ser obtidos.

Faça a seguinte alteração no componente **Busca**.

```
import React, { Component } from 'react'
import { InputText } from 'primereact/inputtext'
import { Button } from 'primereact/button'
export default class Busca extends Component {
    state = {
        termoDeBusca: ''
    }
    onTermoAlterado = (event) => {
        console.log(event.target.value)
        this.setState({termoDeBusca: event.target.value})
    }
    render() {
        return (
            // empilhando os filhos
            <div className="flex flex-column">
                {/* ícone à esquerda, largura máxima */}
                <span className="p-input-icon-left w-full">
                    <i className="pi pi-search"/>
                    <InputText
                        value={this.state.termoDeBusca}
                        //largura máxima
                        className="w-full"
                        onChange={this.onTermoAlterado}
                        placeholder={this.props.dica}>
                    />
                </span>
                <Button
                    label="OK"
                    className="p-button-outlined mt-2"
                />
            </div>
        )
    }
}
```

```
)  
}  
  
Busca.defaultProps ={  
  dica: 'Digite algo que deseja ver...'  
}
```

Note que o funcionamento permanece o mesmo. Entretanto, veja o funcionamento agora.



Embora o funcionamento seja o mesmo, a **Source of Truth** mudou. Os dados de interesse são agora armazenados por um componente React, em seu estado. O que a propriedade **value** do elemento HTML **input** armazena é determinado pelo componente. Neste caso, dizemos que o componente React é **controlado**. Considera-se uma boa prática utilizar componentes controlados.

Leia mais sobre componentes controlados aqui.

<https://reactjs.org/docs/forms.html#controlled-components>

Sobre componentes não controlados você pode ler aqui.

<https://reactjs.org/docs/uncontrolled-components.html>

Leia também sobre o conceito de **Single Source of Truth** aqui.

https://en.wikipedia.org/wiki/Single_source_of_truth

2.5 (Usando um form) Vamos ajustar a aplicação para que ela faça uso de um elemento HTML **form**. Veja.

```
export default class Busca extends Component {  
...  
render() {  
    return (  
        <form>  
            /* empilhando os filhos */  
            <div className="flex flex-column">  
                /* ícone à esquerda, largura máxima */  
                <span className="p-input-icon-left w-full">  
                    <i className="pi pi-search"/>  
                    <InputText  
                        value={this.state.termoDeBusca}  
                        //largura máxima  
                        className="w-full"  
                        onChange={this.onTermoAlterado}  
                        placeholder={this.props.dica}  
                    />  
                </span>  
                <Button  
                    label="OK"  
                    className="p-button-outlined mt-2"  
                />  
            </div>  
        </form>  
    )  
}
```

Digite algum texto e aperte Enter. Clique também no botão. Repare que o form tem comportamento padrão: o navegador tenta submetê-lo a um servidor e a tela “pisca”. Desejamos evitar esse funcionamento padrão e especificar uma função que será executada uma vez que o form seja submetido. Assim, temos a chance de executar código antes de a requisição acontecer. Além disso, podemos fazer uma requisição assíncrona e atualizar somente as partes da página que, de fato, tiverem algum conteúdo novo para exibir. Veja o ajuste a seguir.

```
export default class Busca extends Component {  
...  
    onFormSubmit = (event) => {  
        //não deixa o navegador submeter o form  
        event.preventDefault()  
  
    }  
...  
    render() {  
        return (  
            <form onSubmit={this.onFormSubmit}>  
...  
        )  
    }  
}
```

Teste novamente a aplicação digitando algo e apertando Enter. Clique também no botão. Repare que o navegador não mais tenta submeter o form.

2.6 (Função para realizar a busca) Digamos que a função que realiza a busca seja definida pelo componente App. Para que ela entre em execução no momento oportuno, precisamos passá-la - via props - ao componente Busca. Defina a função no componente App assim. Observe que ele foi redefinido, agora utilizando uma classe.

```
import React from 'react'
import Busca from './Busca'

export default class App extends React.Component {

    onBuscaRealizada = (termo) => {
        console.log(termo)
    }

    render(){
        return (
            <div className="grid justify-content-center m-auto w-9 border-round border-1 border-400">
                <div className="col-12">
                    <h1 className="text-center">Exibir uma lista de...</h1>
                </div>
                <div className="col-8">
                    <Busca onBuscaRealizada={this.onBuscaRealizada}/>
                </div>
            </div>
        )
    }
}
```

Agora, faça com que o componente Busca a coloque em execução no momento certo. Veja.

```
export default class Busca extends Component {
...
onFormSubmit = (event) => {
    //não deixa o navegador submeter o form
    event.preventDefault()
    this.props.onBuscaRealizada(this.state.termoDeBusca)
}

...
}
```

Teste a aplicação novamente. Após digitar algum texto e clicar no botão, o console do navegador deverá exibir aquilo que foi digitado.

2.7 (Pexels) Pexels é um site bastante utilizado para a obtenção de figuras de alta qualidade e gratuitas. É possível acessar a sua página oficial como um usuário regular e fazer buscas e obtenções de figuras. Também é possível acessar a base de dados por meio de uma API. O primeiro passo para isso é visitar a seguinte página.

<https://www.pexels.com/api/>

Será necessário criar uma conta ou fazer login com alguma rede social. Uma vez que esteja logado, visite a mesma página uma vez mais e clique **Your API Key** para visualizar a sua chave. A seguir, aprenderemos uma forma interessante de adicioná-la ao projeto.

2.8 (.env para projetos React) A chave para acesso ao site Pexels é um exemplo de "variável de ambiente". Trata-se de um valor que pode mudar conforme a aplicação muda de estágios (desenvolvimento, teste, produção etc.). Além disso, é uma chave que, evidentemente, não desejamos compartilhar em repositórios públicos. Em situações como essa, utilizamos um pacote clássico chamado **dotenv**. A ideia consiste em criar um arquivo chamado **.env** - daí o nome do pacote - responsável por abrigar as variáveis que têm características como essas descritas. O pacote se encarrega de configurar os valores para que eles possam ser acessados pelos componentes sem que tenham que ser mencionados explicitamente. Para aplicações React, utilizaremos o pacote chamado **react-dotenv**. Visite a sua página a seguir.

<https://www.npmjs.com/package/react-dotenv>

Instale o pacote com

`npm install react-dotenv`

Crie um arquivo chamado **.env** na raiz do seu projeto, ao lado das pastas **src**, **public** etc. Adicione a ele esse conteúdo aqui.

PEXELS_KEY=SUA CHAVE AQUI

Abra o arquivo **package.json** e faça os ajustes destacados a seguir.

```
{  
  "name": "pessoal_react_lista_de_figuras",  
  "version": "0.1.0",  
  "private": true,  
  "dependencies": {  
    "@testing-library/jest-dom": "^5.14.1",  
    "@testing-library/react": "^11.2.7",  
    "@testing-library/user-event": "^12.8.3",  
    "primeflex": "^3.0.1",  
    "primeicons": "^4.1.0",  
    "primereact": "^6.5.1",  
    "react": "^17.0.2",  
    "react-dom": "^17.0.2",  
    "react-dotenv": "^0.1.3",  
    "react-scripts": "4.0.3",  
    "react-transition-group": "^4.4.2",  
    "web-vitals": "^1.1.2"  
  },  
  "scripts": {  
    "start": "react-dotenv && react-scripts start",  
    "build": "react-dotenv && react-scripts build",  
    "serve": "react-dotenv && serve build",  
    "test": "react-scripts test",  
    "eject": "react-scripts eject"  
  },  
  "eslintConfig": {  
    "extends": [  
      "react-app",  
      "react-app/jest"  
    ]  
  }  
}
```

```
},
"browserslist": {
  "production": [
    ">0.2%",
    "not dead",
    "not op_mini all"
  ],
  "development": [
    "last 1 chrome version",
    "last 1 firefox version",
    "last 1 safari version"
  ]
},
"react-dotenv": {
  "whitelist": ["PEXELS_KEY"]
}
}
```

Se o projeto estiver em execução, poderá ser necessário reiniciá-lo. Neste caso, aperte **CTRL+C** - duas vezes, no Windows - no terminal em que ele estiver em execução para pará-lo. A seguir, use

npm start

para colocá-lo em execução uma vez mais.

As variáveis especificadas no arquivo **.env** podem, a partir de agora, ser acessadas de duas formas diferentes. Para testar ambas, abra o arquivo **App.js** e adicione o código a seguir.

```
import env from 'react-dotenv'  
...  
export default class App extends React.Component {  
...  
render(){  
    console.log(env.PEXELS_KEY)  
    console.log(window.env.PEXELS_KEY)  
...  
}
```

Ambas são as formas são equivalentes. Assim que a aplicação for colocada em execução, você deverá ver a sua chave - duas ocorrências - no console do seu navegador. Feitos os testes, você pode remover as linhas mencionadas.

2.9 (A documentação da Pexels e o pacote pexels-javascript) A documentação da Pexels pode ser acessada a partir deste link.

<https://www.pexels.com/api/documentation/>

Todos os recursos podem ser acessados por requisições HTTP comuns. Entretanto, também é possível utilizar uma biblioteca mantida oficialmente pela Pexels para simplificar as operações. Ela pode ser instalada com

`npm install pexels`

2.10 (Fazendo requisições) O primeiro passo para fazer requisições é importar a função `createClient` de `pexels`. Ela deve ser chamada com a chave de API como parâmetro e devolve um objeto capaz de fazer requisições. Vamos instanciá-lo utilizando o método do ciclo de vida `componentDidMount`. Veja.

```
import env from 'react-dotenv' //mantenha
import { createClient } from 'pexels'

export default class App extends React.Component {
...
pexelsClient = null

componentDidMount(){
    this.pexelsClient = createClient(env.PEXELS_KEY)
}
```

As requisições serão realizadas pelo método `onBuscaRealizada`, já que ele é chamado a cada clique no botão. No exemplo a seguir, fazemos uma busca e exibimos o resultado no console.

```
export default class App extends React.Component {

    onBuscaRealizada = (termo) => {
        this.pexelsClient.photos.search({
            query: termo
        })
        .then(pics => console.log(pics))
    }
...
}
```

Digite algo que deseje buscar e clique OK.

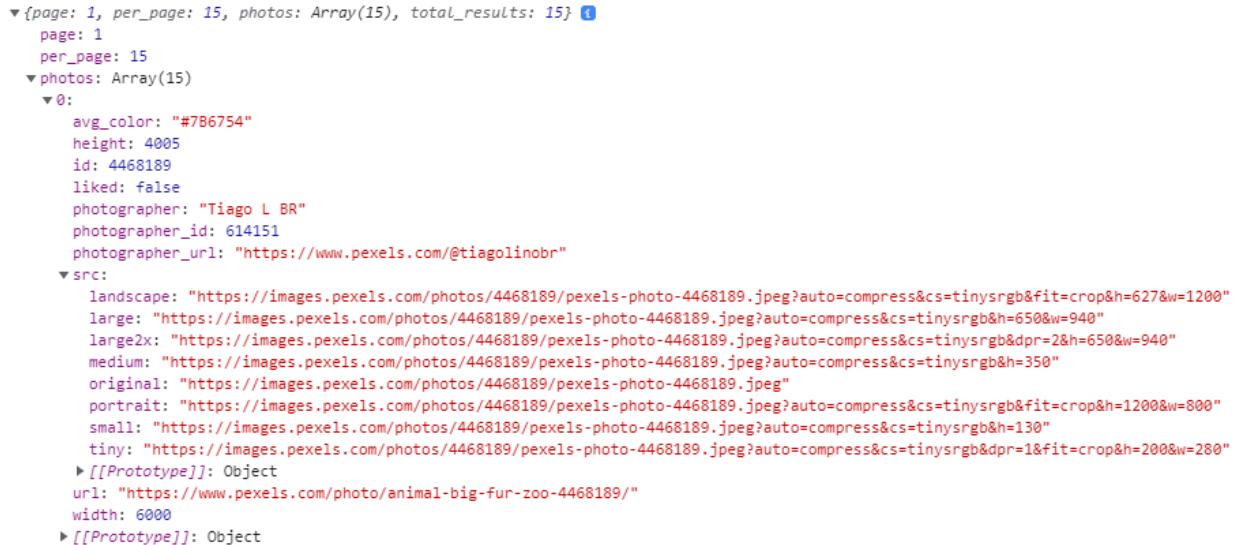
O resultado esperado é parecido com esse aqui.



The screenshot shows a browser's developer tools with the 'Console' tab selected. A search bar at the top contains the query 'Animals'. Below it is a button labeled 'OK'. The console output displays the following JSON object:

```
▼ {page: 1, per_page: 15, photos: Array(15), total_results: 15} ⓘ
  page: 1
  per_page: 15
  ▼ photos: Array(15)
    ▷ 0: {id: 4468189, width: 6000, height: 4005, url: 'https://www.pexels.com/photo/animal-big-fur-zoo-4468189/'}
    ▷ 1: {id: 4393616, width: 3456, height: 5184, url: 'https://www.pexels.com/photo/animal-pet-cute-grey-4393616/'}
    ▷ 2: {id: 4393613, width: 3456, height: 5184, url: 'https://www.pexels.com/photo/animal-pet-cute-grey-4393613/'}
    ▷ 3: {id: 4393637, width: 3456, height: 5184, url: 'https://www.pexels.com/photo/animal-pet-cute-grey-4393637/'}
    ▷ 4: {id: 436135, width: 5859, height: 3805, url: 'https://www.pexels.com/photo/street-animal-pet-cute-436135/'}
    ▷ 5: {id: 5919712, width: 5120, height: 5120, url: 'https://www.pexels.com/photo/nature-animal-cute-grass-5919712/'}
    ▷ 6: {id: 7283069, width: 3526, height: 2351, url: 'https://www.pexels.com/photo/city-road-beach-people-7283069/'}
    ▷ 7: {id: 4393615, width: 3456, height: 5184, url: 'https://www.pexels.com/photo/animal-pet-cute-grey-4393615/'}
    ▷ 8: {id: 9392997, width: 4000, height: 6000, url: 'https://www.pexels.com/photo/landscape-nature-people-country-side-9392997/'}
    ▷ 9: {id: 4468188, width: 6000, height: 4005, url: 'https://www.pexels.com/photo/black-and-white-bird-on-brick-tree-branch-4468188/'}
    ▷ 10: {id: 4600882, width: 2215, height: 3172, url: 'https://www.pexels.com/photo/flight-bird-water-winter-4600882/'}
    ▷ 11: {id: 4229167, width: 3402, height: 2202, url: 'https://www.pexels.com/photo/wood-nature-bird-animal-4229167/'}
    ▷ 12: {id: 4315288, width: 5845, height: 3897, url: 'https://www.pexels.com/photo/landscape-nature-field-animal-4315288/'}
    ▷ 13: {id: 4903759, width: 3873, height: 5809, url: 'https://www.pexels.com/photo/animal-country-side-agriculture-farm-4903759/'}
    ▷ 14: {id: 5056633, width: 5616, height: 3159, url: 'https://www.pexels.com/photo/landscape-man-people-woman-5056633/'}
  length: 15
  ▷ [[Prototype]]: Array(0)
  total_results: 15
  ▷ [[Prototype]]: Object
```

Repare que o resultado é um objeto JSON que tem uma propriedade chamada **photos**. Analise a figura a seguir para entender melhor os detalhes de cada objeto JSON pertencente à coleção associada à chave **photos**.



The screenshot shows a detailed view of one of the photo objects from the previous JSON response. The object has the following properties:

```
▼ {page: 1, per_page: 15, photos: Array(15), total_results: 15} ⓘ
  page: 1
  per_page: 15
  ▼ photos: Array(15)
    ▷ 0:
      avg_color: "#786754"
      height: 4005
      id: 4468189
      liked: false
      photographer: "Tiago L BR"
      photographer_id: 614151
      photographer_url: "https://www.pexels.com/@tiagolinobr"
      ▶ src:
        landscape: "https://images.pexels.com/photos/4468189/pexels-photo-4468189.jpeg?auto=compress&cs=tinysrgb&fit=crop&h=627&w=1200"
        large: "https://images.pexels.com/photos/4468189/pexels-photo-4468189.jpeg?auto=compress&cs=tinysrgb&h=650&w=940"
        large2x: "https://images.pexels.com/photos/4468189/pexels-photo-4468189.jpeg?auto=compress&cs=tinysrgb&dpr=2&h=650&w=940"
        medium: "https://images.pexels.com/photos/4468189/pexels-photo-4468189.jpeg?auto=compress&cs=tinysrgb&h=350"
        original: "https://images.pexels.com/photos/4468189/pexels-photo-4468189.jpeg"
        portrait: "https://images.pexels.com/photos/4468189/pexels-photo-4468189.jpeg?auto=compress&cs=tinysrgb&fit=crop&h=1200&w=800"
        small: "https://images.pexels.com/photos/4468189/pexels-photo-4468189.jpeg?auto=compress&cs=tinysrgb&h=130"
        tiny: "https://images.pexels.com/photos/4468189/pexels-photo-4468189.jpeg?auto=compress&cs=tinysrgb&dpr=1&fit=crop&h=200&w=280"
      ▷ [[Prototype]]: Object
      url: "https://www.pexels.com/photo/animal-big-fur-zoo-4468189/"
      width: 6000
      ▷ [[Prototype]]: Object
```

Assim, dado um objeto pertencente à coleção associada à chave **photos** podemos acessar URLs associadas a **src.landscape**, **src.original**, **src.small** etc. Os nomes das chaves são descritivos. Veja também os parâmetros em cada URL.

A fim de exibir as fotos, vamos armazenar a coleção **photos** no estado do componente. Veja.

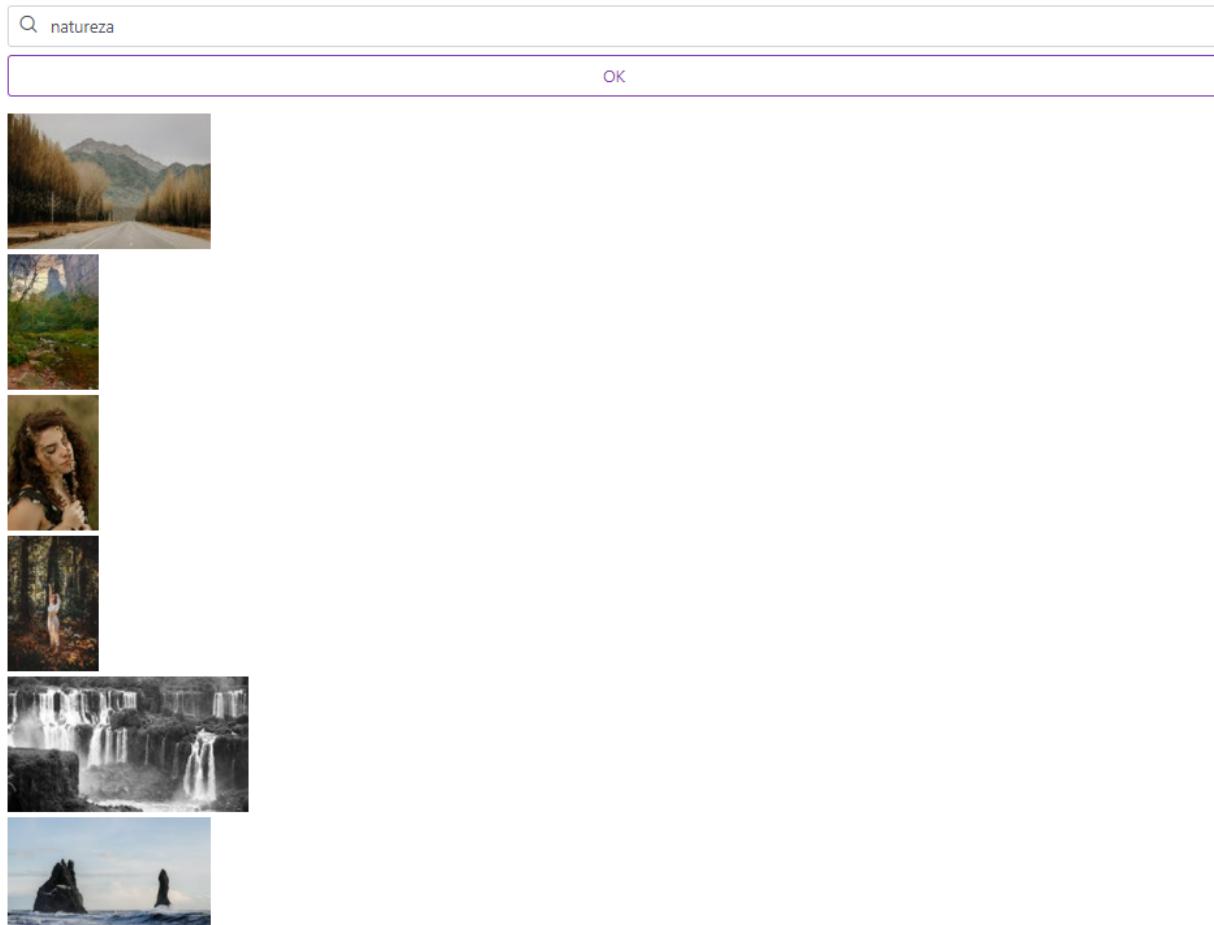
```
export default class App extends React.Component {  
  
  state = {pics: []}  
  
  onBuscaRealizada = (termo) => {  
    this.pexelsClient.photos.search({  
      query: termo  
    })  
    .then(pics => this.setState({pics: pics.photos}))  
  }  
}
```

A seguir, ajustamos o método **render** para que a JSX resultante inclua um elemento para cada imagem. Para tal, utilizamos a função **map** do Javascript para, como o nome sugere, mapear cada elemento da lista à JSX de interesse. Veja.

```
export default class App extends React.Component {  
...  
  render(){  
    return (  
      <div className="grid justify-content-center m-  
auto w-9 border-round border-1 border-400">  
        <div className="col-12">  
          <h1 className="text-center">  
            Exibir uma lista de...  
          </h1>  
        </div>  
        <div className="col-8">  
          <Busca  
            onBuscaRealizada={this.onBuscaRealizada}/>  
        </div>  
        <div className="col-8">  
          {  
            this.state.pics.map((pic, key) => (  
              <div key={key}>  
                <img src={pic.src.small}/>  
              </div>  
            ))  
          }  
        </div>  
      </div>  
    )  
  }  
}
```

Faça uma consulta e visualize o resultado assim.

Exibir uma lista de...



Repare que as fotos estão alinhadas à esquerda e que cada uma tem um tamanho específico. Em breve faremos com que elas sejam dispostas de uma maneira interessante na tela, aplicando o efeito “tile”.

2.11 (Componentes para exibição de uma imagem e da lista de imagens) Em nossa análise inicial optamos por definir um componente próprio para a exibição de uma imagem e um outro componente próprio para fazer a exibição de uma lista de imagens. Passamos, portanto, à sua implementação.

Crie um arquivo chamado **Imagen.js** na pasta **components**. Veja a sua definição.

```
import React from 'react'

const Imagem = ({pic}) => {
    return (
        <div>
            <img src={pic} />
        </div>
    )
}

export default Imagem
```

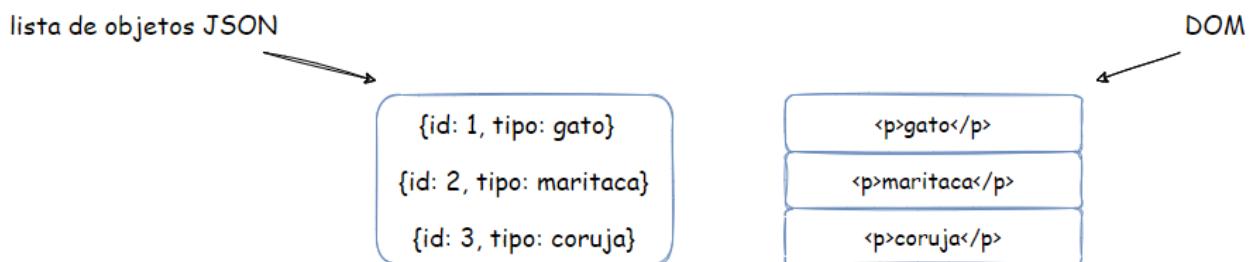
Crie um arquivo chamado **ListaImagens.js** na pasta **components**. A sua definição é a seguinte.

```
import React from 'react'
import Imagem from './Imagen'

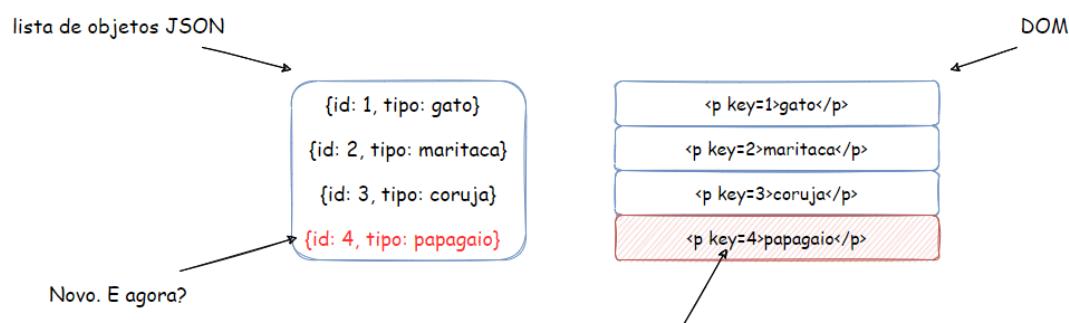
const ListaImagens = ({pics}) => {
    return (
        pics.map((pic, key) => (
            <Imagen
                pic={pic.src.small}
                key={key}
            />
        ))
    )
}

export default ListaImagens
```

O segundo parâmetro entregue para a arrow function especificada na lista de parâmetros da função map é simplesmente o índice - começando de zero - do elemento da vez. Dessa forma, seu valor não se repete para elementos diferentes. Repare que o valor foi utilizado como "key" de cada elemento gerado. A razão de ser desse atributo, idealmente existente em cada elemento de uma lista, é uma lista: desempenho. Para entender isso melhor, considere uma lista de objetos JSON que deverão ser exibidos em uma lista. Cada um deles terá a sua própria expressão JSX. Veja.



A pergunta a ser respondida é a seguinte. Como a árvore DOM é atualizada caso o usuário decida adicionar um novo animal à lista de objetos JSON? O React é capaz de fazer essa atualização de maneira muito eficiente, desde que os elementos envolvidos possuam uma key. Com esse valor em mãos, o React pode comparar elemento por elemento, verificando aqueles que já existem na árvore e adicionando somente aqueles que ainda não existirem. Veja.



Novo. E agora?

Elementos têm chave. Assim, o React pode comparar e somente incluir aquele que não existia anteriormente. Não é o caso de os demais serem substituídos por outros elementos idênticos.

Passe a exibir a lista de figuras utilizando o componente criado para tal. Isso deve ser feito no arquivo **App.js**. Veja.

```
export default class App extends React.Component {  
...  
  render(){  
    return (  
      <div className="grid justify-content-center m-  
      auto w-9 border-round border-1 border-400">  
        <div className="col-12">  
          <h1 className="text-  
          center">Exibir uma lista de...</h1>  
        </div>  
        <div className="col-8">  
          <Bus-  
          ca onBuscaRealizada={this.onBuscaRealizada}/>  
        </div>  
        <div className="col-8">  
          <ListaImagens pics={this.state.pics}/>  
        </div>  
      </div>  
    )  
  }  
...  
}
```

2.12 (Componente para exibir o logo Pexels) Estamos utilizando conteúdo gratuito disponibilizado pela Pexels. A sua documentação, que pode ser encontrada a seguir, sugere que mostremos um link ou figura atribuindo-lhes crédito pelo conteúdo, o que obviamente é justo.

<https://www.pexels.com/api/documentation/>

Veja essa parte da documentação que fala sobre isso.

Guidelines

Whenever you are doing an API request make sure to show a **prominent link to Pexels**. You can use a text link (e.g. "Photos provided by Pexels") or a link with our logo.

Always credit our photographers when possible (e.g. "Photo by John Doe on Pexels" with a link to the photo page on Pexels).

You may not copy or replicate core functionality of Pexels (including making Pexels content available as a wallpaper app).

Do not abuse the API. By default, the API is rate-limited to 200 requests per hour and 20,000 requests per month. **You may contact us to request a higher limit**, but please include examples, or be prepared to give a demo, that clearly shows your use of the API with attribution. If you meet our API terms, you can get unlimited requests for free.



Desta forma, vamos criar um componente para exibir seu logo. Na pasta **components**, clique com o direito e crie um arquivo chamado **PexelsLogo.js**. Veja seu conteúdo logo a seguir.

```
import React from 'react'

const PexelsLogo = () => {
  return (
    <div>
      /* target=_blank abre a página em nova aba */
      <a href="https://www.pexels.com" target="_blank">
        
      </a>
    </div>
  )
}

export default PexelsLogo
```

Para exibi-lo, adicione o seguinte conteúdo ao componente definido no arquivo **App.js**.

```
...
import PexelsLogo from './PexelsLogo'
export default class App extends React.Component {
...
render(){
    return (
        <div className="grid justify-content-center m-auto
                    w-9 border-round border-1 border-400">
            <div className="col-12">
                <PexelsLogo/>
            </div>
            <div className="col-12">
                <h1 className="text-
center">Exibir uma lista de...</h1>
            </div>
            <div className="col-8">
                <Bus-
ca onBuscaRealizada={this.onBuscaRealizada}/>
            </div>
            <div className="col-8">
                <ListaImagens pics={this.state.pics}/>
            </div>
        </div>
    )
}
}
```

O resultado esperado é o seguinte.



2.13 (Fazendo requisições HTTP de maneira mais genérica, usando a Axios)
As requisições HTTP que temos feito estão encapsuladas na biblioteca oferecida pela Pexels. Embora isso seja bastante conveniente, é fundamental aprender sobre formas como requisições HTTP em geral podem ser realizadas, sem o uso de bibliotecas tão específicas como essa da Pexels. Claro, faremos uso de uma biblioteca de requisições HTTP. A ideia, no entanto, é que faremos construções genéricas, que poderão ser utilizadas para fazer requisições a diferentes servidores. A biblioteca que utilizaremos se chama **axios**.

Veja a sua página no npm registry aqui.

<https://www.npmjs.com/package/axios>

Sua página no Github aqui.

<https://github.com/axios/axios>

E uma página de documentação aqui.

<https://axios-http.com/>

A documentação da Pexels nos mostra que a URL base para todas as requisições é

<https://api.pexels.com/v1/>

Já a seguinte página, também da documentação, nos mostra que o “endpoint” desejado, ou seja, apropriado para buscar fotos, é esse aqui.

/search

<https://www.pexels.com/api/documentation/?#photos-search>

Para começar a utilizar a axios, clique com o direito na pasta **src** e crie uma pasta chamada **utils**. Nesta nova pasta, crie um arquivo chamado **pexelsClient.js**. Veja o conteúdo do arquivo em que utilizamos a axios.

```
import axios from "axios"
import env from 'react-dotenv'
export default axios.create({
  baseURL: 'https://api.pexels.com/v1/',
  headers: {
    Authorization: env.PEXELS_KEY
  }
})
```

Para utilizar o novo cliente, ajuste o arquivo **App.js** assim. Repare que comentamos a implementação do método **onBuscaRealizada** e fizemos uma nova. Também foi comentada a definição do objeto **pexelsClient**, que utilizávamos anteriormente.

```
...
import pexelsClient from '../utils/pexelsClient'
export default class App extends React.Component {
// onBuscaRealizada = (termo) => {
//     this.pexelsClient.photos.search({
//         query: termo
//     })
//     .then(pics => this.setState({pics: pics.photos}))
// }
// pexelsClient = null

onBuscaRealizada = (termo) => {
    pexelsClient.get('/search', {
        params: { query: termo}
    })
    .then(result => {
        console.log(result)
        //data é um atributo definido pela axios
        //o conteúdo da resposta vem associado a essa chave
        this.setState ({pics: result.data.photos})
    })
}
...
}
```

Execute novamente a aplicação. Ela deve estar operando como antes.

2.14 (Exibindo as figuras como um grid) A aplicação exibe as figuras uma abaixo da outra. O resultado não é muito agradável visualmente. Vamos utilizar algumas classes da PrimeFlex para fazer a disposição das figuras como em um grid.

- I. Se a tela for extragrande, exibimos quatro figuras lado a lado.
- II. Se a tela for grande, exibimos três figuras lado a lado.
- III. Se a tela for média, exibimos duas figuras lado a lado.
- IV. Se a tela for pequena ou menos, exibimos todas as figuras empilhadas.

Começamos ajustando o componente **App**, no arquivo **App.js**. Tanto a div que abriga o componente **Busca** quanto a div que abriga o componente **ListaImagens** terão direito a 12 colunas. O segundo, em particular, terá um grid da PrimeFlex aninhado. Ele se encarrega de enviar, via props, os nomes das classes que serão utilizadas na distribuição de colunas, como descrito. Veja.

```
...
export default class App extends React.Component {
...
render(){
    return (
        <div className="grid justify-content-center m-auto w-9 border-round border-1 border-400">
            <div className="col-12">
                <PexelsLogo/>
            </div>
            <div className="col-12">
                <h1 className="text-
center">Exibir uma lista de...</h1>
            </div>
            <div className="col-12">
                <Busca onBuscaRealizada={this.onBuscaRealizada}/>
            </div>
            <div className="col-12">
                <div className="grid">
                    <ListaImagens imgStyle={'col-12 md:col-6 lg:col-
4 xl:col-3'} pics={this.state.pics}/>
                </div>
            </div>
        </div>
    )
}
}
```

O componente **ListaImagens** - arquivo **ListaImagens.js** - recebe as classes que determinam o número de colunas de acordo com o tamanho da tela mas não as utiliza. Ele simplesmente as repassa para seus filhos: as imagens. Veja.

```
import React from 'react'
import Imagem from './Imagen'

const ListaImagens = ({pics, imgStyle}) => {
    console.log(pics)
    return (
        pics.map((pic, key) => (
            <Imagen
                imgStyle={imgStyle}
                pic={pic.src.small}
                key={key}
            />
        ))
    )
}

export default ListaImagens
```

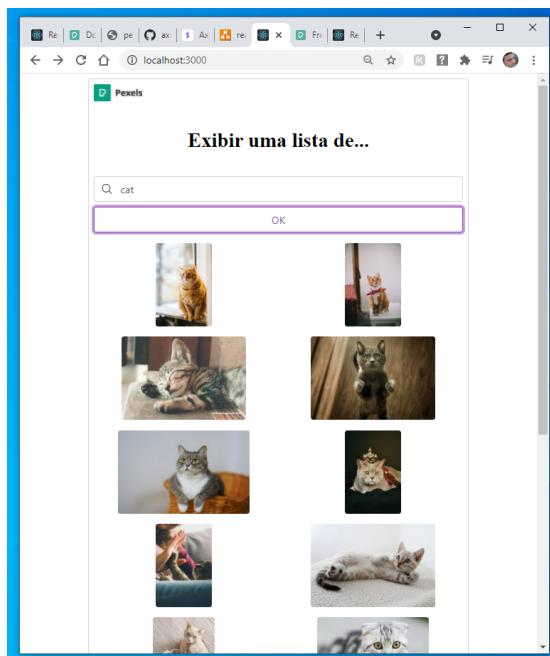
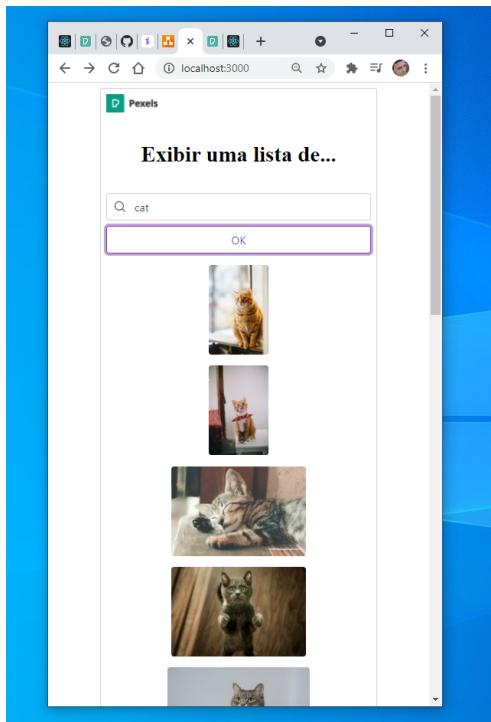
O componente **Imagen** - arquivo **Imagen.js** - passa a utilizá-las, além de centralizar as imagens e aplicar borda. Veja.

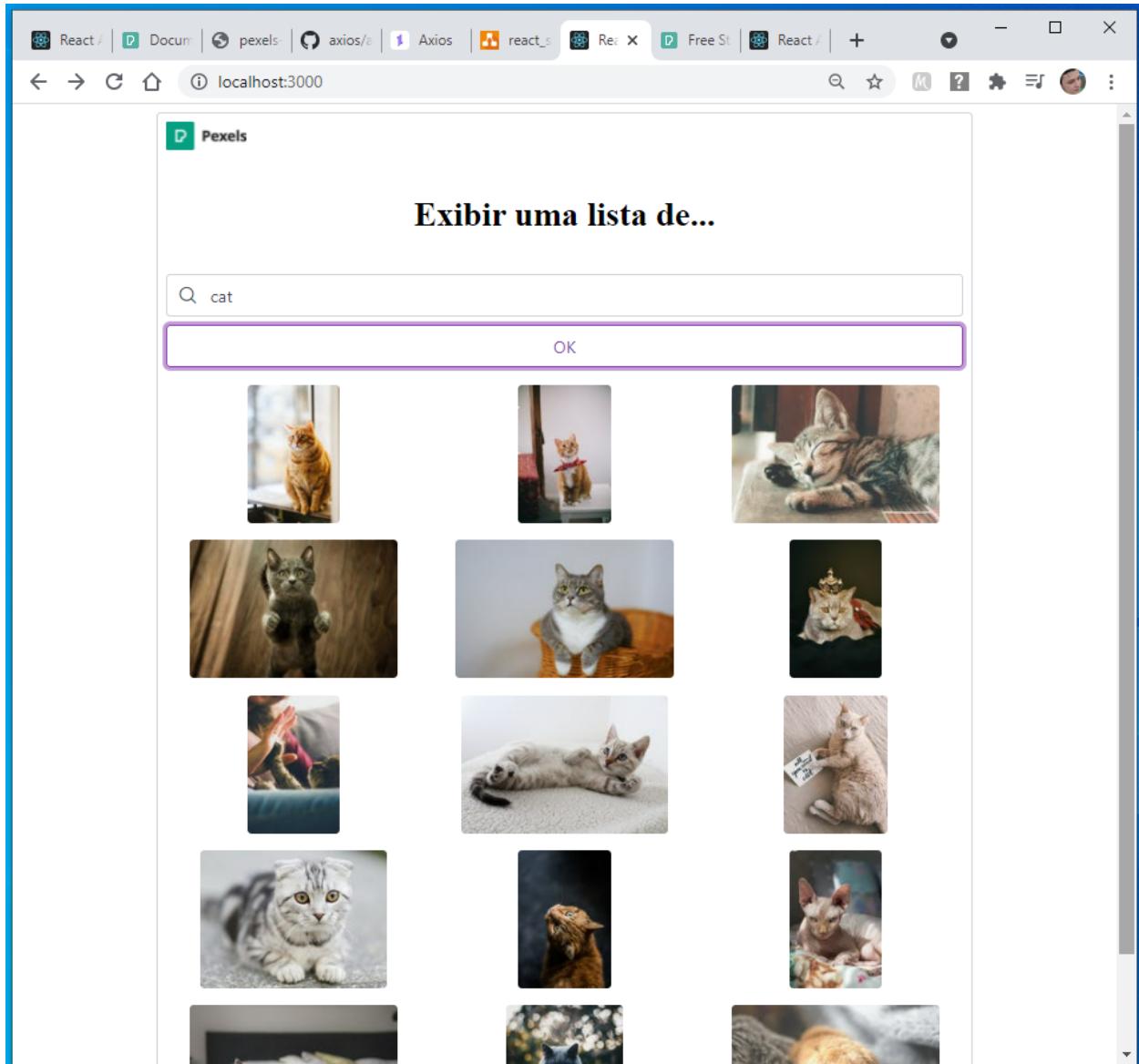
```
import React from 'react'

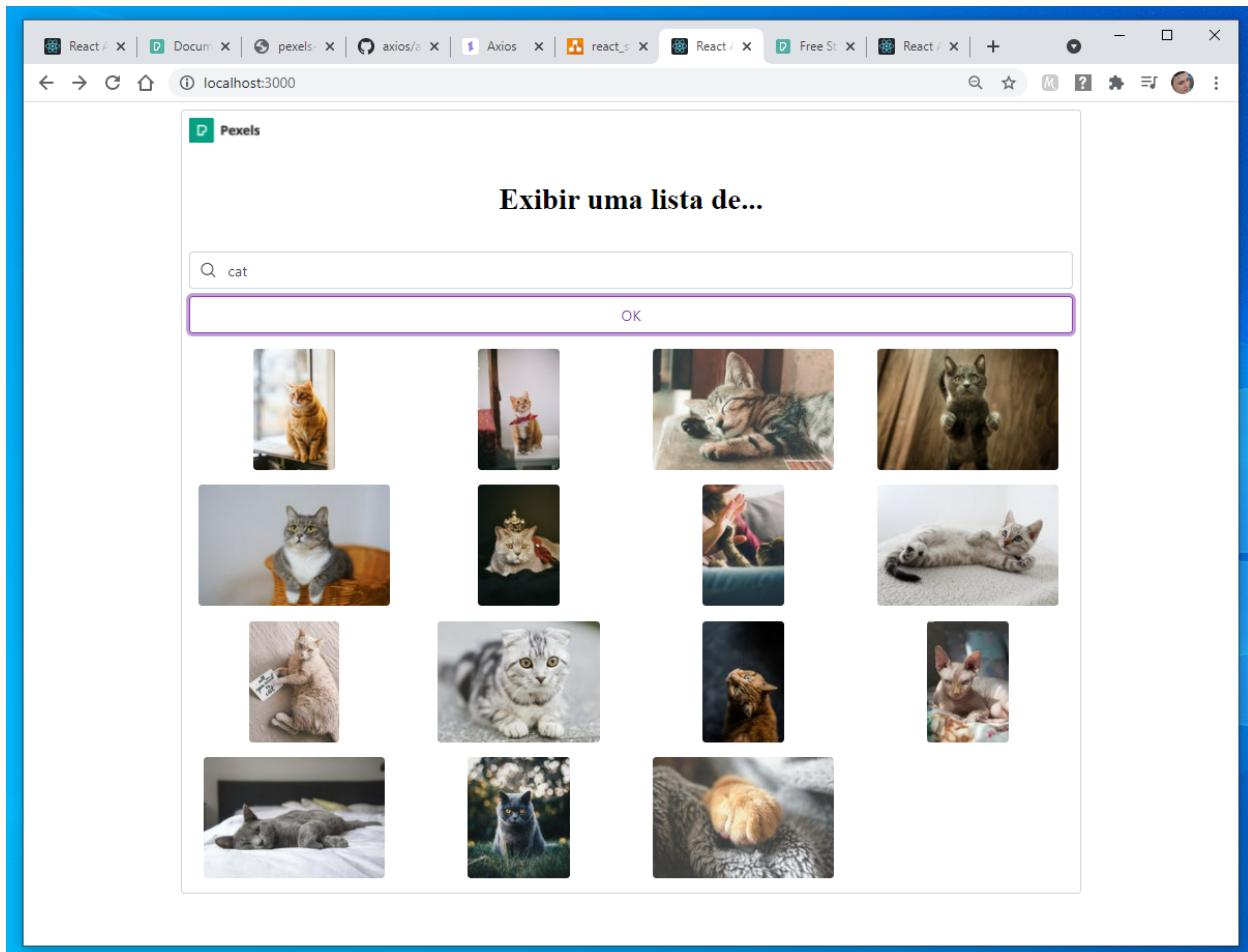
const Imagem = ({pic, imgStyle}) => {
    return (
        <div className={`${imgStyle} flex justify-content-center`}>
            <img className="border-round" src={pic} />
        </div>
    )
}

export default Imagem
```

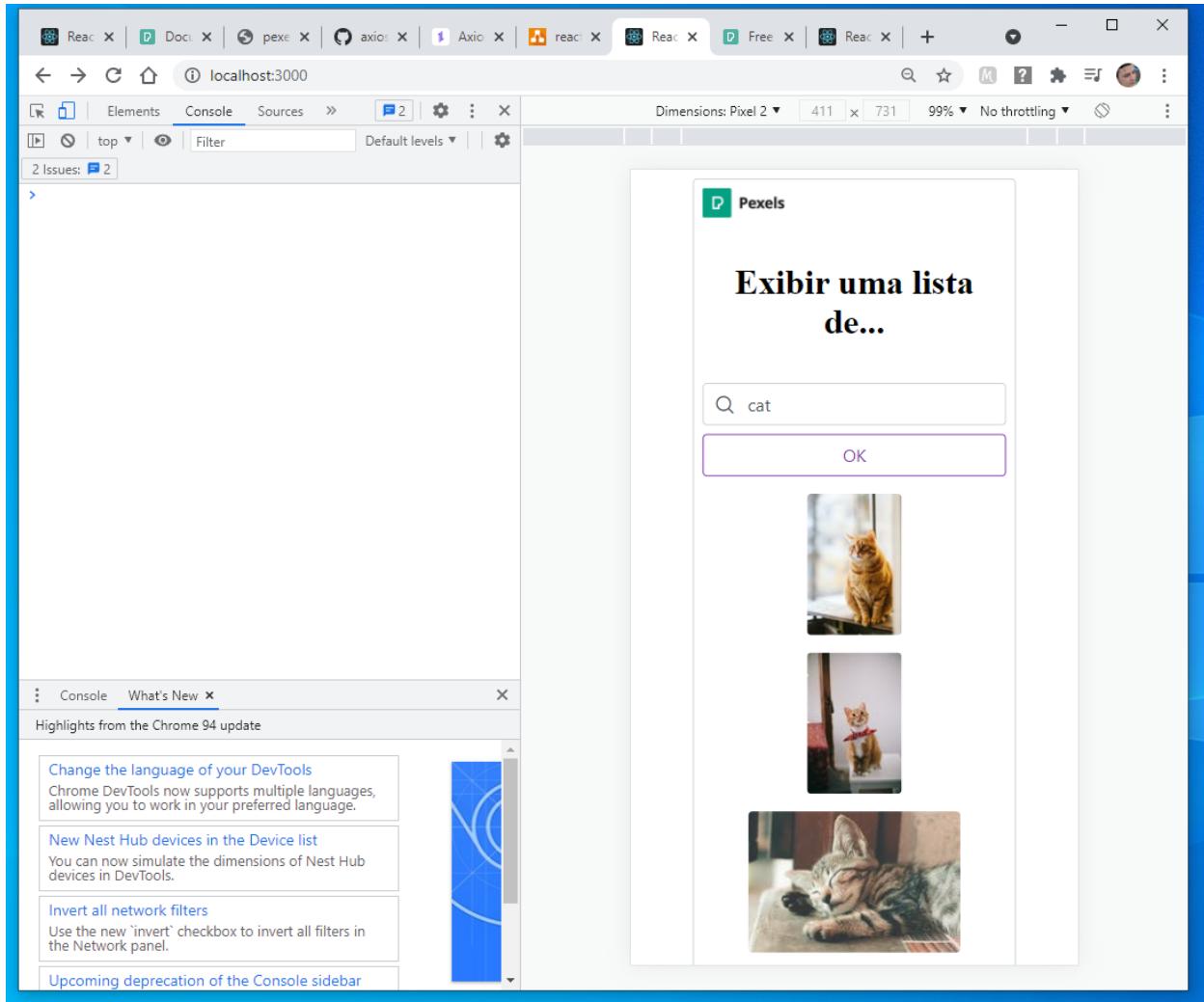
Pronto. Faça novos testes. Faça uma busca e redimensione a janela do navegador horizontalmente para obter resultados como os seguintes.

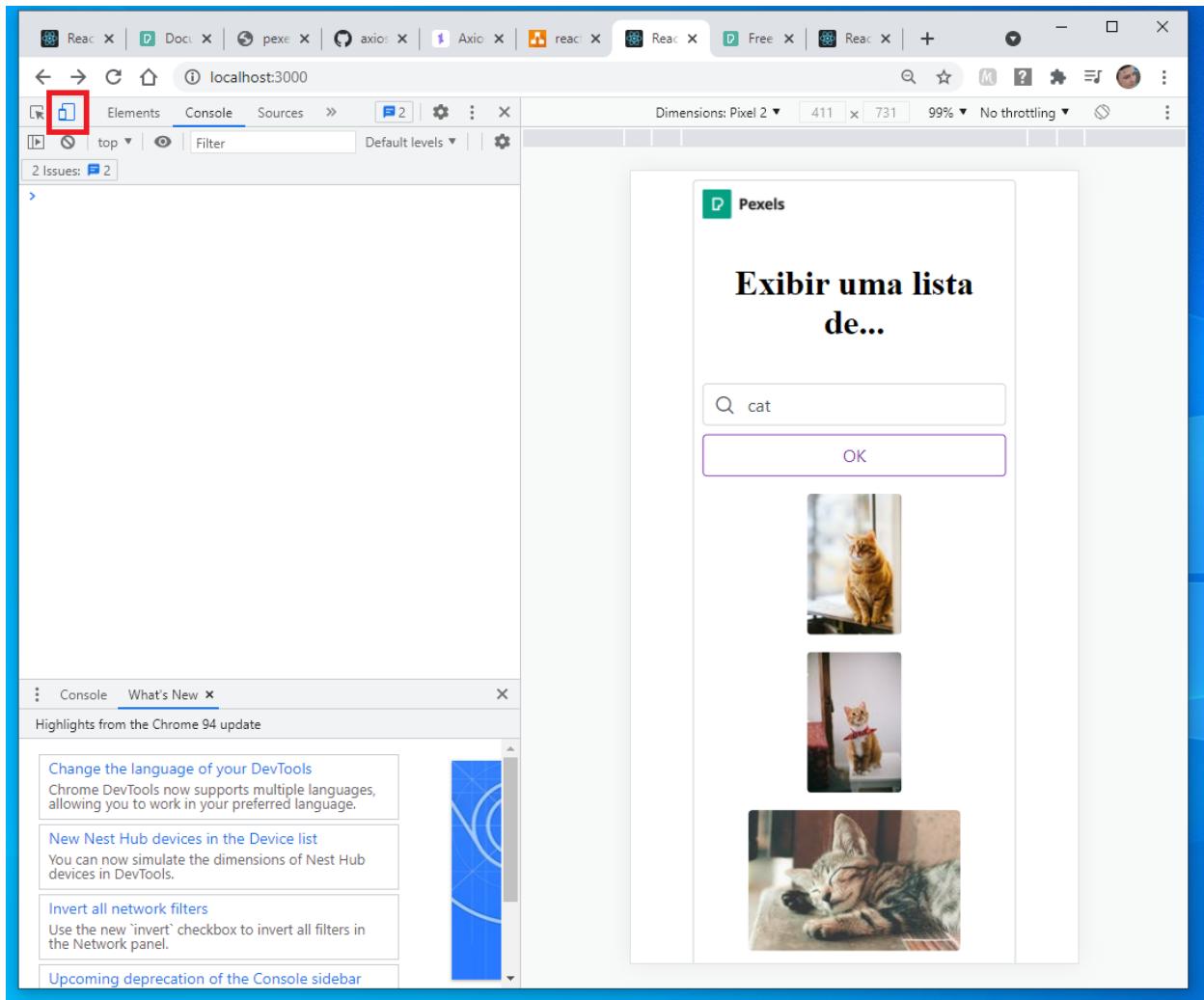




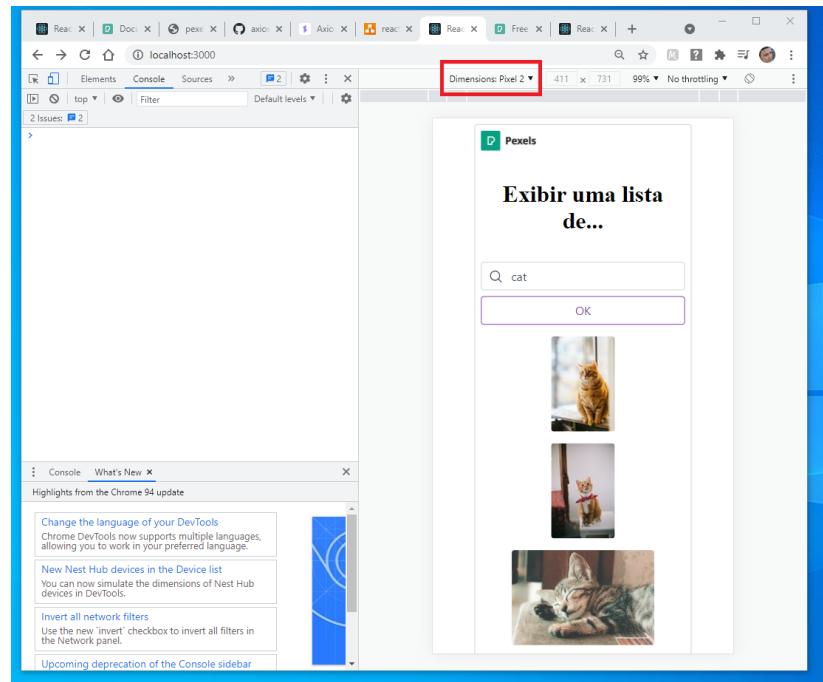


Um outro teste interessante envolve o uso do simulador de dispositivos móveis embutido no Google Chrome. Para utilizá-lo, aperte CTRL + SHIFT + I (ou seja, abra o Chrome Dev Tools) e clique no botão destacado a seguir.

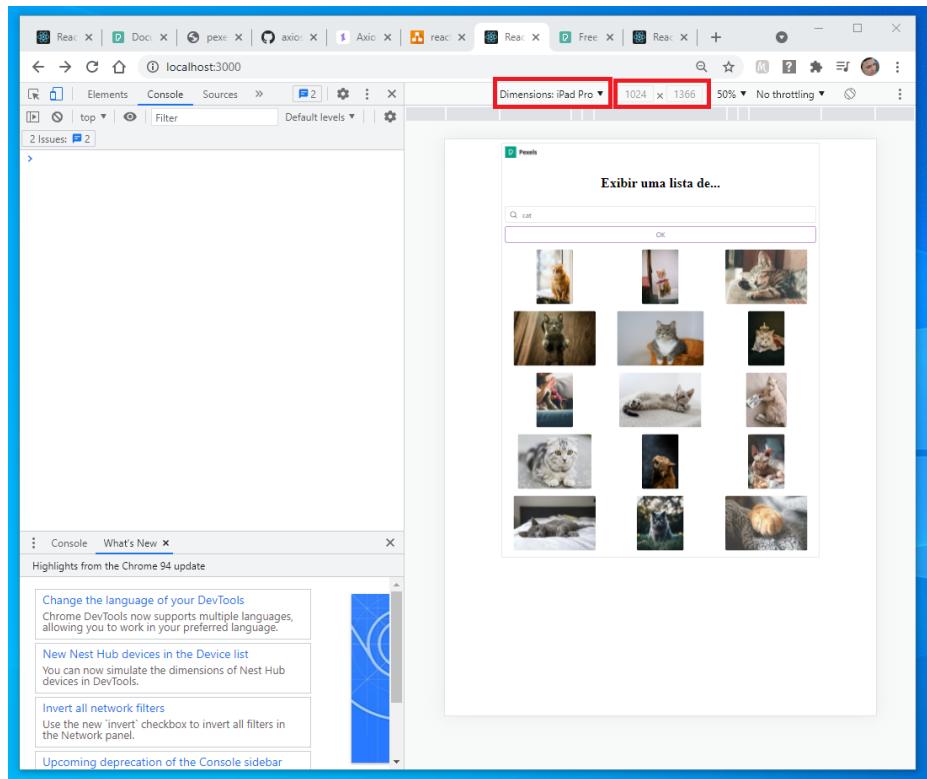




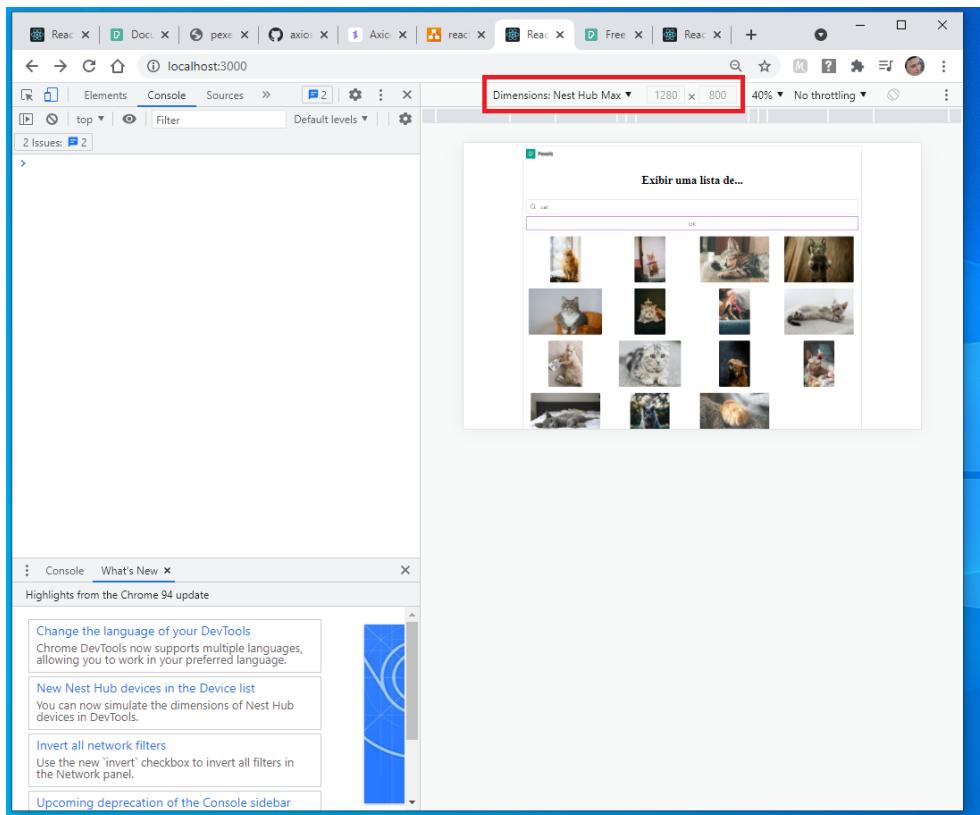
Repare que agora o navegador exibe uma espécie de dispositivo móvel ao lado. Você pode selecionar entre diferentes modelos, com diferentes quantidades de pixels, para visualizar os resultados. Para isso, clique como a seguir.



Veja o resultado com um Ipad Pro.



Ele aparece com 1024 pixels de largura, por isso é considerado “grande”. Três figuras são exibidas lado a lado, portanto. Por outro lado, veja o resultado do **Nest Hub Max**. Ele tem 1280px de largura e é considerado **extragrande**. Mostra, portanto, quatro figuras lado a lado.



OBS: As medidas de breakpoint da PrimeFlex são as seguintes:

xs: largura < 576px
sm: 576px ≤ largura < 768px
md: 768px ≤ largura < 992px
lg: 992px ≤ largura < 1200px
xl: largura ≥ 1200px

OBS: Os dispositivos ilustrados são apenas exemplos e as opções que o Chrome disponibiliza variam em função do tempo.

Referências

React - A JavaScript library for building user interfaces. 2021. Disponível em <<https://reactjs.org/>>. Acesso em agosto de 2021.