

Solicitações POST com Axios

A maneira mais fácil de fazer uma solicitação POST com o [Axios](#) é a `axios.post()` função. O primeiro parâmetro `axios.post()` é o URL e o segundo é o corpo da solicitação HTTP.

```
const res = await axios.post('https://httpbin.org/post', { hello: 'world' });
```

```
res.data.json; // { hello: 'world' }
```

Por padrão, se o 2º parâmetro for `axios.post()` um objeto, o Axios serializa o objeto para JSON usando a `JSON.stringify()` função. Se o segundo parâmetro for um objeto, o Axios também define o `content-type` cabeçalho como `application/json`, portanto, a maioria dos frameworks da Web, como o [Express](#), poderá converter automaticamente o corpo da solicitação em um objeto JavaScript para você.

```
const res = await axios.post('https://httpbin.org/post', { hello: 'world' });
```

```
res.data.headers['Content-Type']; // application/json; charset=utf-8
```

Para substituir o `content-type` cabeçalho no Axios, você deve usar o terceiro parâmetro para `axios.post()`: o `options` parâmetro. Defina a `options.header['content-type']` opção para definir o `content-type` cabeçalho.

```
const res = await axios.post('https://httpbin.org/post', { hello: 'world' }, {
  headers: {
    // 'application/json' is the modern content-type for JSON, but some
    // older servers may use 'text/json'.
    // See: http://bit.ly/text-json
    'content-type': 'text/json'
  }
});
```

```
res.data.headers['Content-Type']; // text/json
```

Corpos de solicitação codificados em formulário

Se você passar uma string como `body` parâmetro para `axios.post()`, o Axios definirá o `content-type` cabeçalho como `application/x-www-form-urlencoded`. Isso significa que o corpo da solicitação deve ser um grupo de pares chave/valor separados por `&`, como `key1=value1&key2=value2`.

```
const res = await axios.post('https://httpbin.org/post', 'hello=world');
res.data.form; // { hello: 'world' }
res.data.headers['Content-Type']; // application/x-www-form-urlencoded
```

Solicitações PUT com Axios

A maneira mais fácil de fazer uma solicitação PUT com o `Axios` é a `axios.put()` função. O primeiro parâmetro `axios.put()` é o URL e o segundo é o corpo da solicitação HTTP.

```
const res = await axios.put('https://httpbin.org/put', { hello:
'world' });

res.data.json; // { hello: 'world' }
```

Corpos de solicitação codificados em formulário

Se você passar uma string como `body` parâmetro para `axios.put()`, o Axios definirá o `content-type` cabeçalho como `application/x-www-form-urlencoded`. Isso significa que o corpo da solicitação deve ser um grupo de pares chave/valor separados por `&`, como `key1=value1&key2=value2`.

```
const res = await axios.put('https://httpbin.org/put', 'hello=world');

res.data.form; // { hello: 'world' }
res.data.headers['Content-Type']; // application/x-www-form-urlencoded
```

Para atualizar os dados da coluna, precisaremos dos ID respectivos, os quais obteremos na API.

Crie uma função chamada `setData`. Vincule essa função ao botão Update.

```
<Button onClick={() => setData()}>Update</Button>
```

Depois disso, precisamos passar os dados como parâmetro para a função superior.

```
<Button onClick={() => setData(data)}>Update</Button>
```

Na função acima, registramos esses dados no console:

```
const setData = (data) => {  
  console.log(data);  
}
```

Clique no botão Update na tabela e confira o console. Você obterá os dados do campo da tabela respectivo.

Vamos definir esses dados no localStorage.

```
const setData = (data) => {  
  
  let { id, firstName, lastName, checkbox } = data;  
  
  localStorage.setItem('ID', id);  
}
```

```
localStorage.setItem('First Name', firstName);

localStorage.setItem('Last Name', lastName);

localStorage.setItem('Checkbox Value', checkbox)

}
```

Aprenda a programar — currículo gratuito de 3 mil horas

29 DE MARÇO DE 2022/[#REACT](#)

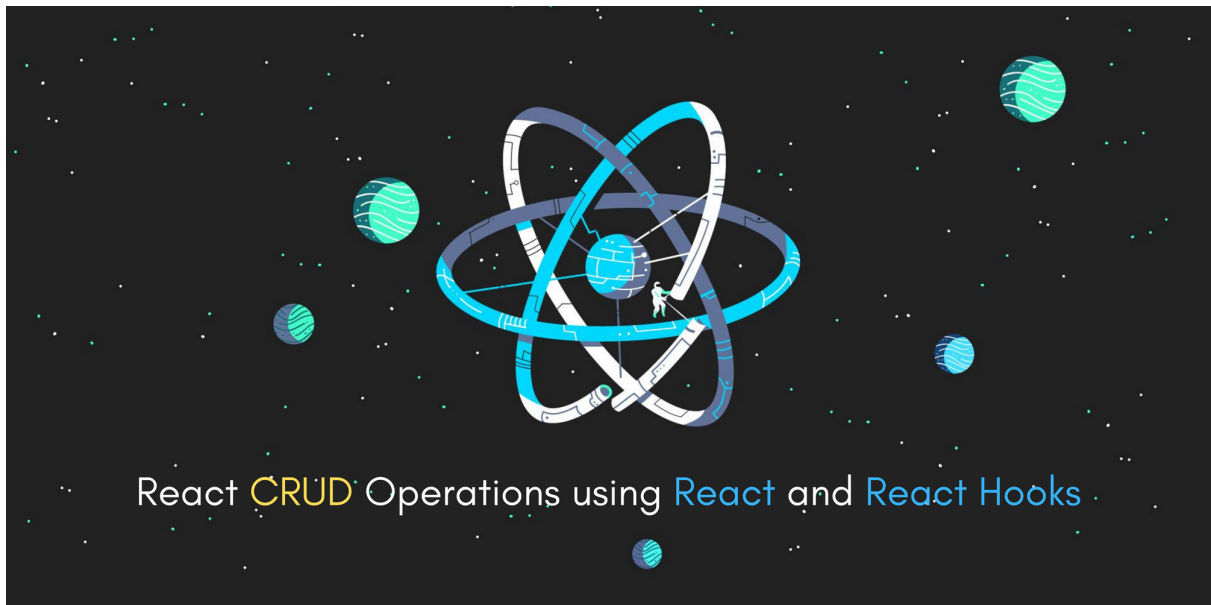
Como realizar operações de CRUD usando React, hooks do React e Axios



Tradutor: Daniel Rosa



Autor: Nishant Kumar (em inglês)



Artigo original: [How to Perform CRUD Operations using React, React Hooks, and Axios](#)

Se você está trabalhando com React, pode ser difícil entender e implementar solicitações de API.

Neste artigo, aprenderemos como tudo funciona implementando operações de CRUD usando React, hooks do React, React Router e Axios.

Vamos lá.

Como instalar o Node e o npm

Primeiramente, vamos instalar o Node em nosso sistema. Vamos usá-lo, primeiramente, para executar nosso código em JavaScript.

Para baixar o Node, acesse <https://nodejs.org/en/>.

Você também precisará do **node package manager**, ou npm, que já vem integrado ao Node. Você pode usá-lo para instalar os pacotes para seus apps em JavaScript.

Felizmente, como ele vem com o Node, não é necessário baixá-lo separadamente.

Depois de baixá-los e instalá-los, abra seu terminal ou o prompt de comando e digite `node -v`. Assim, você poderá conferir a versão do Node que você tem.

Como criar sua aplicação em React

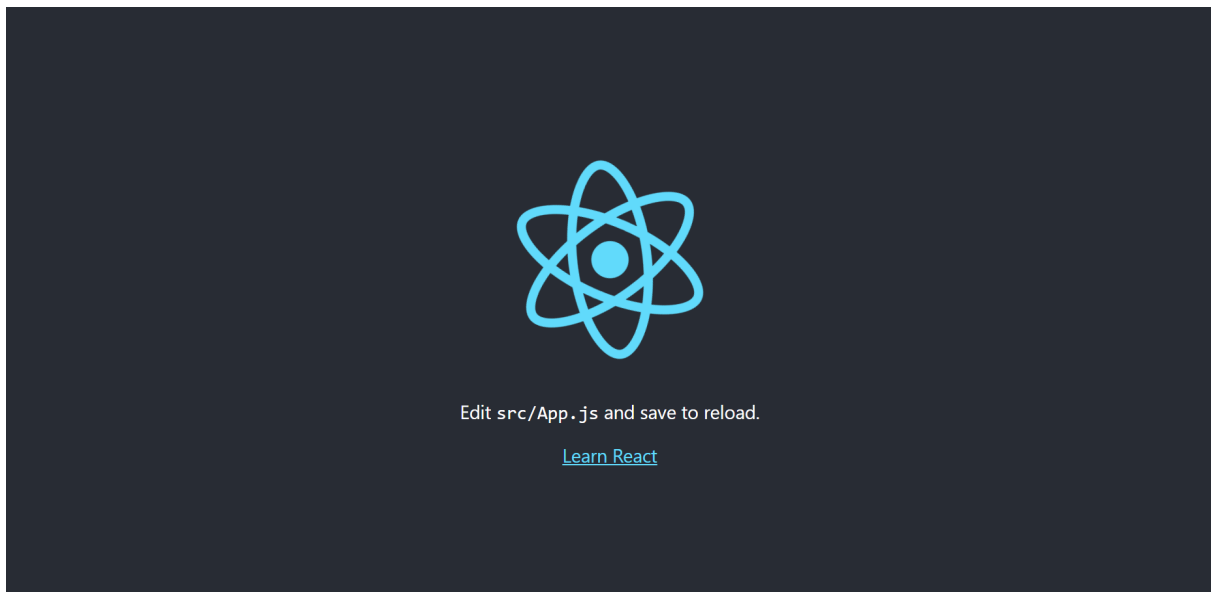
Para criar sua aplicação em React, digite

`npx-create-react-app <nome-do-seu-app>` no seu terminal ou `npx-create-react-app react-crud`, neste caso.

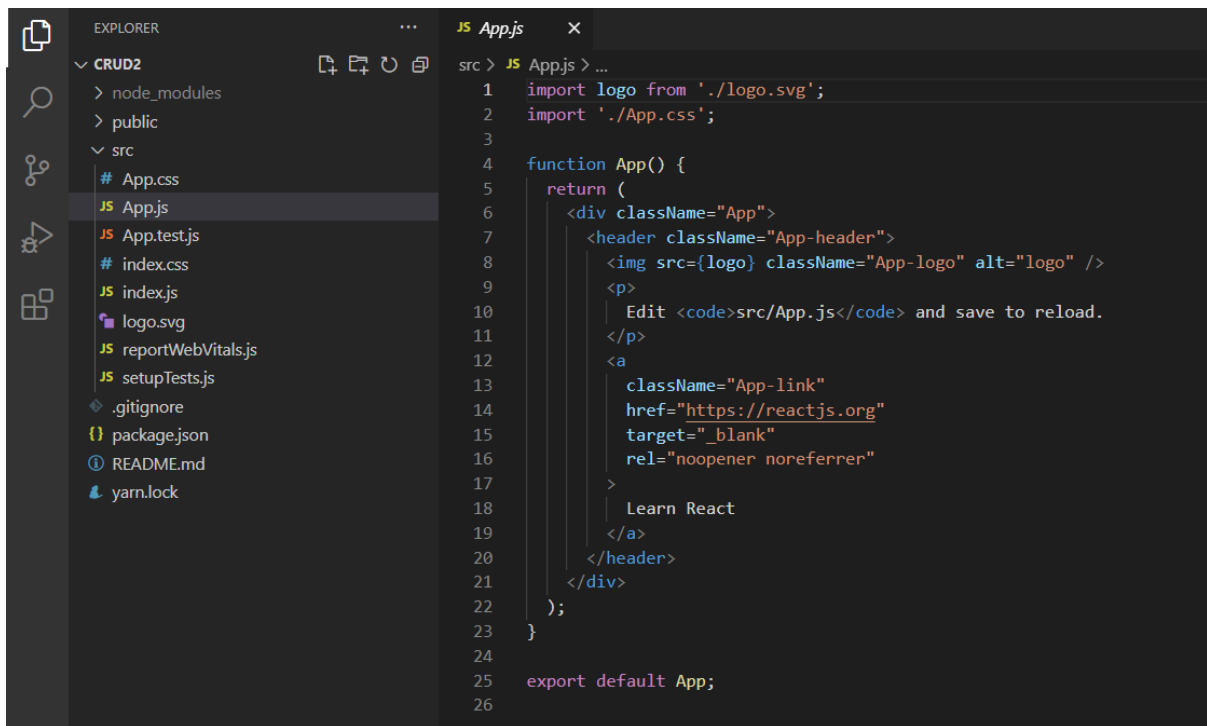
Você verá os pacotes serem instalados.

Ao terminar de baixar os pacotes, vá até a pasta do projeto e digite `npm start`.

Você verá o modelo padrão do React, que tem essa aparência:



O boilerplate padrão do React



```
src > JS App.js > ...
1  import logo from './logo.svg';
2  import './App.css';
3
4  function App() {
5    return (
6      <div className="App">
7        <header className="App-header">
8          <img src={logo} className="App-logo" alt="logo" />
9          <p>
10             Edit <code>src/App.js</code> and save to reload.
11          </p>
12          <a
13            className="App-link"
14            href="https://reactjs.org"
15            target="_blank"
16            rel="noopener noreferrer"
17          >
18            Learn React
19          </a>
20        </header>
21      </div>
22    );
23  }
24
25  export default App;
26
```

Nosso arquivo App.js

Como instalar o pacote Semantic UI para o React

Vamos instalar o pacote Semantic UI para o React em nosso projeto. Semantic UI é uma biblioteca de UI (interface de usuário) feita para o React que tem componentes de UI pré-construídos, como tabelas, botões e muitos outros recursos.

Você pode instalar o pacote usando um dos comandos abaixo, dependendo do seu gerenciador de pacotes.

```
yarn add semantic-ui-react semantic-ui-css
```

Para o gerenciador de pacotes do Yarn

```
npm install semantic-ui-react semantic-ui-css
```

Para o gerenciador de pacotes do Node, o NPM

Além disso, importe a biblioteca em seu arquivo de entrada principal do app, chamado index.js.

```
import 'semantic-ui-css/semantic.min.css'
```

Cole isto no seu arquivo index.js

Como criar sua aplicação com CRUD

Agora, vamos começar a criar nossa aplicação com CRUD usando o React.

Primeiro, vamos adicionar um título à nossa aplicação.

Em nosso arquivo app.js, teremos o seguinte:

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

```
function App() {  
  
  return (  
  
    <div>  
  
      React Crud Operations  
  
    </div>  
  
  );  
  
}  
  
export default App;
```

Adicionando um título à nossa aplicação

Agora, vamos garantir que ela esteja centralizada.

Dê à `div` pai a `classname main`. No arquivo `App.css`, usaremos `Flexbox` para centralizar o título.

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

```
function App() {
```

```
  return (
```

```
    <div className="main">
```

```
      React Crud Operations
```

```
    </div>
```

```
  );
```

```
}
```

```
export default App;
```

app.js com a className main na div pai

```
.main{
```

```
display: flex;
```

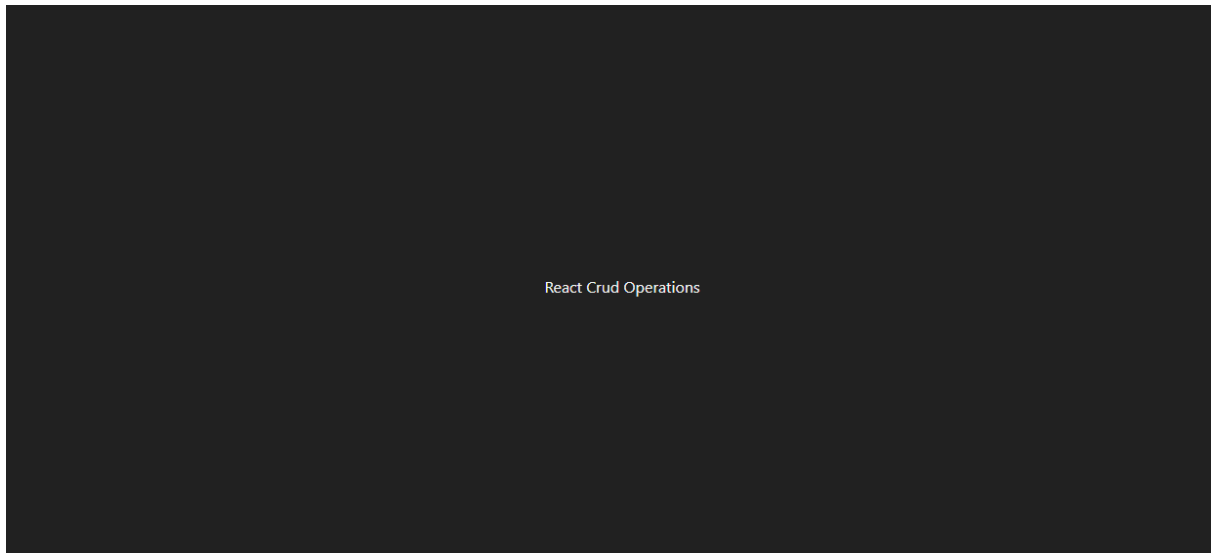
```
justify-content: center;
```

```
align-items: center;
```

```
height: 100vh;
```

```
}
```

Nosso arquivo app.css



Nosso título, agora, está centralizado com perfeição.

Agora que a aparência está melhor, precisamos colocá-lo em realce e adicionar umas fontes legais. Para fazer isso, usaremos as tags ao redor do título, assim:

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

```
function App() {
```

```
return (  
  
  <div className="main">  
  
    <h2 className="main-header">React Crud  
Operations</h2>  
  
  </div>  
  
);  
  
}  
  
export default App;
```

Vamos importar uma Google Font. Acesse
<https://fonts.google.com/> para escolhermos uma.

Selecione a fonte de sua preferência. Para o exemplo, usaremos a família de fontes Montserrat.

Importe a sua fonte preferida no arquivo App.css, assim:

```
@import  
url('https://fonts.googleapis.com/css2?family  
=Montserrat&display=swap');
```

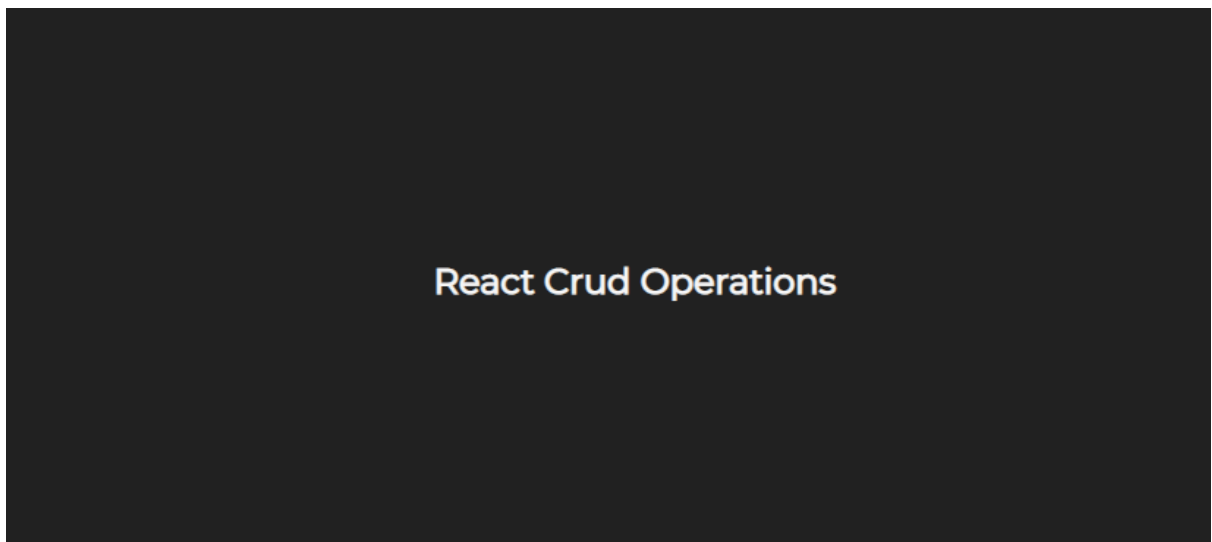
Agora, vamos mudar a fonte do título.

```
<div className="main">  
  
    <h2 className="main-header">React Crud  
Operations</h2>  
  
</div>
```

Dê à tag do título a `className main-header`, assim.

Em seguida, no seu App.css, adicione a família da fonte:

```
.main-header{  
  
  font-family: 'Montserrat', sans-serif;  
  
}
```

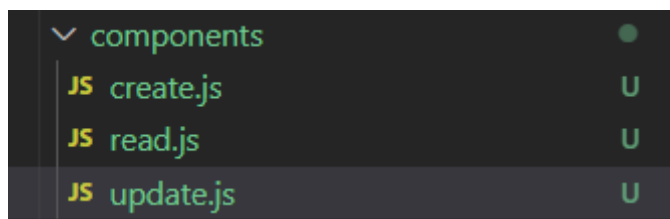


Agora, você verá o título modificado.

Como criar seus componentes de CRUD

Vamos criar quatro componentes de CRUD – Create, Read, Update e Delete (criar, ler, atualizar e excluir, respectivamente).

Na nossa pasta *src*, criamos uma pasta chamada *components*. Dentro dela, criamos três arquivos – *create.js*, *read.js* e *update.js*. Para a exclusão (*delete*), não precisamos de um componente adicional.



Agora, vamos implementar a operação de criação.

Para isso, precisamos usar APIs *mock* ou de simulação. Essas APIs enviarão dados ao servidor falso que criaremos, apenas para fins didáticos.

Assim, acesse <https://mockapi.io/> e crie sua conta.

mockapi.io

The easiest way to mock REST APIs! (Check out [docs](#))

GET STARTED

DEMO PROJECT

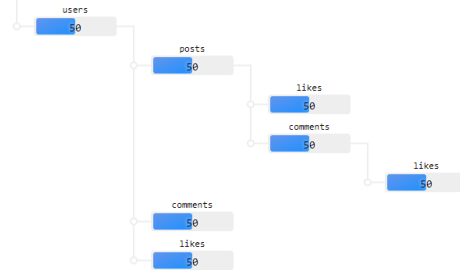
API endpoint

`https://SECRET.mockapi.io/:endpoint`

NEW RESOURCE

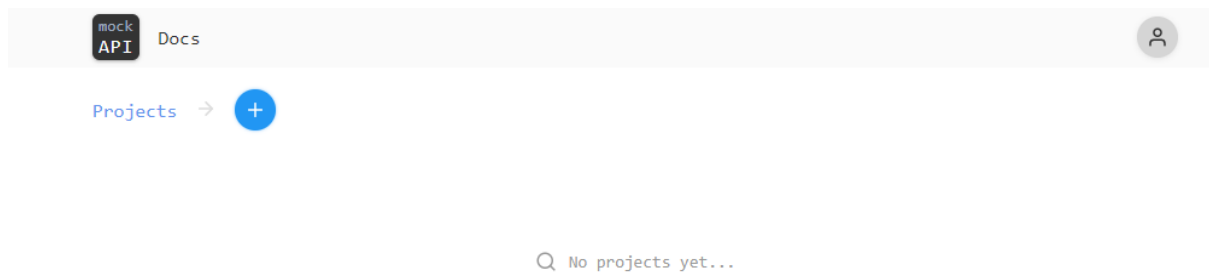
GENERATE ALL

RESET ALL



MockAPI




Criar um projeto clicando no botão +.



Clique no botão + para criar um novo projeto

The modal dialog box is titled 'Project name' and has a close button (X) in the top right corner. It contains three sections: 1. 'Project name' with a text input field containing 'Example: Todo App, Project X...'. 2. 'API Prefix (optional)' with a description 'Add API prefix to all endpoints in this project.' and a text input field containing 'Example: /api/v1'. 3. 'Collaborators (optional)' with a description 'Collaborators can create, update, and delete resources in this project.' and a text input field containing 'Search by name...'. At the bottom, there are two buttons: a blue 'CREATE' button and a grey 'CANCEL' button.

Adicione o nome do projeto e clique no botão Create (Criar).

Projects →  CRUD  

API endpoint
https://60fbca4591156a0017b4c8a7.mockapi.io/:endpoint

[NEW RESOURCE](#) [GENERATE ALL](#) [RESET ALL](#)

🔍 No resources yet...

Em seguida, crie um novo recurso clicando no botão NEW RESOURCE (Novo recurso).

Endpoint
//60fbca459
RESOURCE

Resource name

Enter meaningful resource name, it will be used to generate API endpoints.

Example: users, comments, articles...

Schema (optional)

Define Resource schema, it will be used to generate mock data.

id

Object ID

createdAt

Faker.js

Recent

name

Faker.js

Find name

avatar

Faker.js

Avatar

+

Object template (optional)

To define more complex structure for your data use JSON template. You can reference Faker.js methods using `\$.`.

EXAMPLE

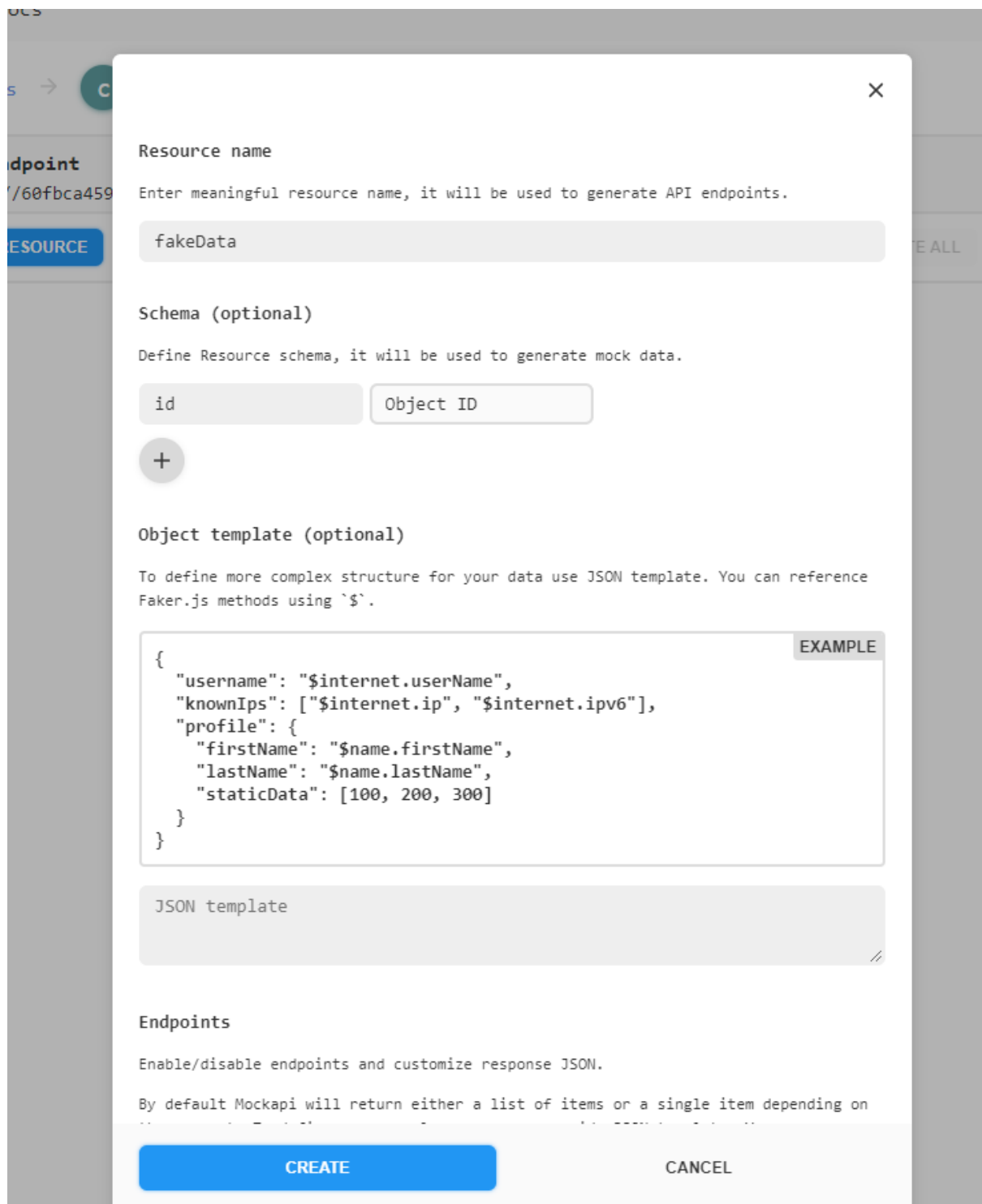
```
{
  "username": "$internet.userName",
  "knownIps": ["$internet.ip", "$internet.ipv6"],
  "profile": {
    "firstName": "$name.firstName",
    "lastName": "$name.lastName",
    "staticData": [100, 200, 300]
  }
}
```

JSON template

CREATE

CANCEL

Será solicitado o nome do recurso (em *Resource Name*).
Coloque ali o nome que achar apropriado.

A screenshot of a web application showing a modal dialog for creating a new resource in Mockapi. The dialog has a close button (X) in the top right corner. It is divided into several sections: 'Resource name' with a text input containing 'fakeData'; 'Schema (optional)' with a text input containing 'id' and a dropdown menu set to 'Object ID', plus a plus button to add more fields; 'Object template (optional)' with a text area containing a JSON template and an 'EXAMPLE' button; and 'Endpoints' with a text area containing 'JSON template'. At the bottom are 'CREATE' and 'CANCEL' buttons.

Resource name

Enter meaningful resource name, it will be used to generate API endpoints.

fakeData

Schema (optional)

Define Resource schema, it will be used to generate mock data.

id Object ID

+

Object template (optional)

To define more complex structure for your data use JSON template. You can reference Faker.js methods using `\$.`.

```
{
  "username": "$internet.userName",
  "knownIps": ["$internet.ip", "$internet.ipv6"],
  "profile": {
    "firstName": "$name.firstName",
    "lastName": "$name.lastName",
    "staticData": [100, 200, 300]
  }
}
```

EXAMPLE

JSON template

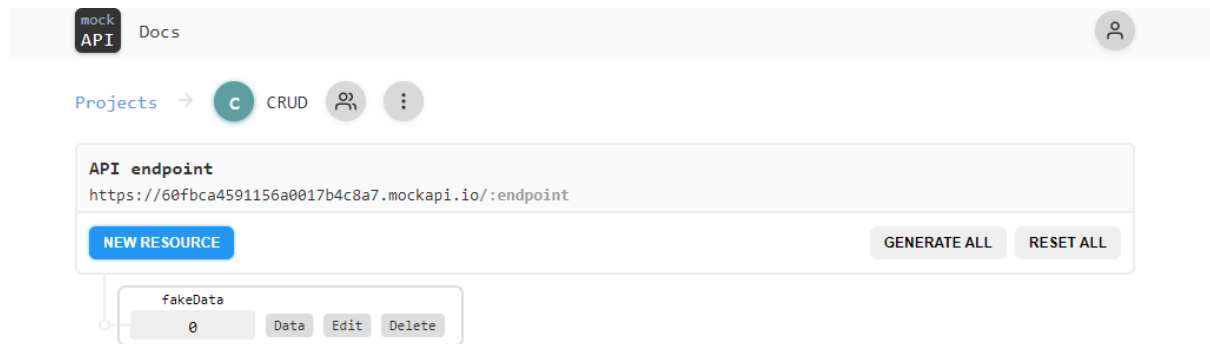
Endpoints

Enable/disable endpoints and customize response JSON.

By default Mockapi will return either a list of items or a single item depending on

CREATE CANCEL

Remova os campos adicionais, como name, avatar e createdAt, pois não precisaremos deles. Então, clique em Create (Criar).



Criamos nossa API falsa, que chamei de fakeData (dados falsos).

Clique em fakeData e você verá a API sendo aberta em uma nova guia. O banco de dados agora está vazio.

Como criar um formulário para o componente Create

Vamos usar um formulário da biblioteca Semantic UI.

Vá até Semantic UI, e procure por *Form* na barra de pesquisa à esquerda.

Form

A form.

Try it

CodeSandbox

Maximize

Permalink

Forms also have a robust shorthand props API for generating controls wrapped in `FormFields`. See shorthand examples below.

First Name

First Name

Last Name

Last Name

☐ I agree to the Terms and Conditions

Submit

Você verá um formulário assim. Clique em Try it (Experimente) na parte superior esquerda para obter o código.

Form

A form.

Try it

CodeSandbox

Maximize

Permalink

Forms also have a robust shorthand props API for generating controls wrapped in `FormFields`. See shorthand examples below.

First Name

First Name

Last Name

Last Name

☐ I agree to the Terms and Conditions

Submit

```
1 import React from 'react'
2 import { Button, Checkbox, Form } from 'semantic-ui-react'
3
4 const FormExampleForm = () => (
5   <Form>
6     <Form.Field>
7       <label>First Name</label>
8       <input placeholder='First Name' />
9     </Form.Field>
10    <Form.Field>
11      <label>Last Name</label>
12      <input placeholder='Last Name' />
13    </Form.Field>
14    <Form.Field>
15      <Checkbox label='I agree to the Terms and Conditions' />
16    </Form.Field>
17    <Button type='submit'>Submit</Button>
18  </Form>
19 )
20
21 export default FormExampleForm
22
```

Copie este código e cole-o no seu arquivo `Create.js`, assim:

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

```
import { Button, Checkbox, Form } from  
'semantic-ui-react'
```

```
const Create = () => (
```

```
  <Form>
```

```
    <Form.Field>
```

```
      <label>First Name</label>
```

```
      <input placeholder='First Name'
```

```
    />
```

```
    </Form.Field>
```

```
    <Form.Field>
```

```
      <label>Last Name</label>
```

```
<input placeholder='Last Name' />
```

```
</Form.Field>
```

```
<Form.Field>
```

```
<Checkbox label='I agree to the  
Terms and Conditions' />
```

```
</Form.Field>
```

```
<Button type='submit'>Submit</Button>
```

```
</Form>
```

```
)
```

```
export default Create;
```

Importe o componente Create no seu arquivo app.js.

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

```
import Create from './components/create';
```

```
function App() {
```

```
  return (
```

```
    <div className="main">
```

```
      <h2 className="main-header">React Crud  
Operations</h2>
```

```
      <div>
```

```
        <Create/>
```

```
</div>
```

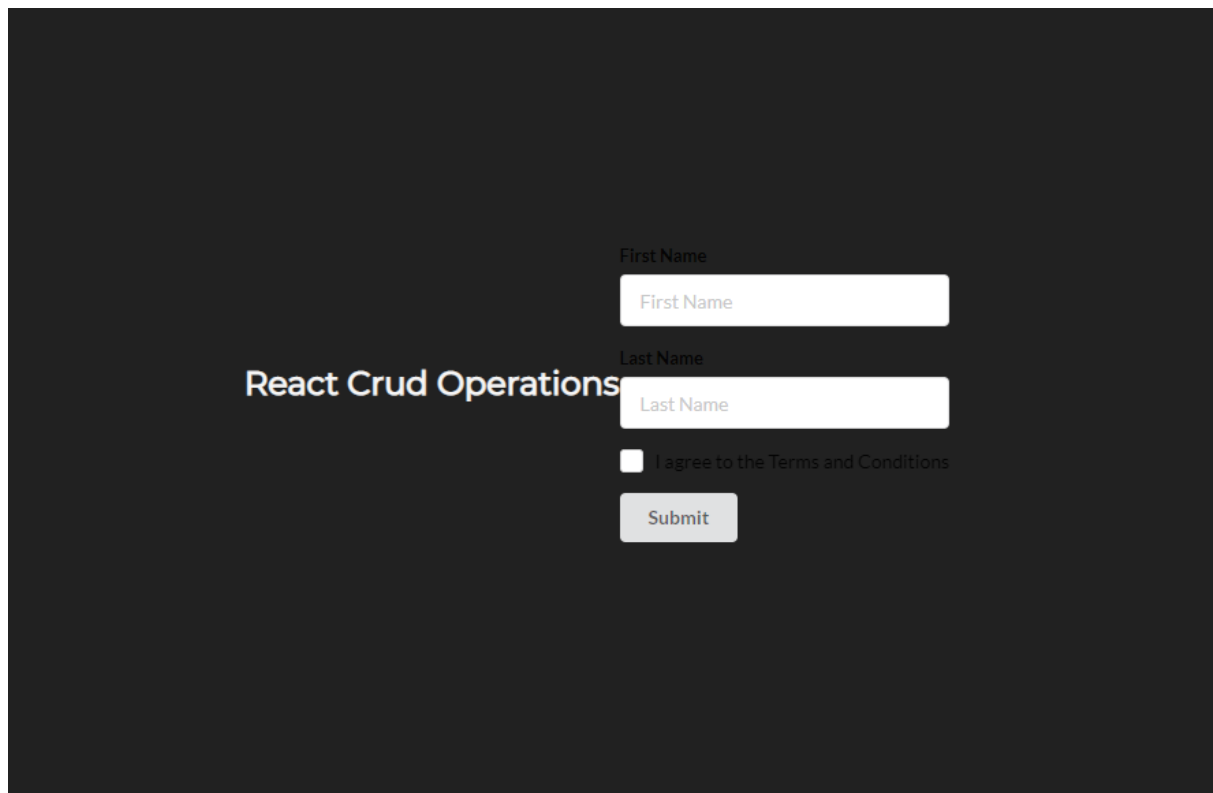
```
</div>
```

```
);
```

```
}
```

```
export default App;
```

Este é o resultado:



Temos um problema agora – os itens não estão alinhados adequadamente e as cores dos labels para os inputs de texto estão em preto. Vamos mudar isso.

No arquivo Create.js, dê à **Form** a className de `create-form`.

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

```
import { Button, Checkbox, Form } from  
'semantic-ui-react'
```

```
const Create = () => (  
  
  <Form className="create-form">  
  
    <Form.Field>  
  
      <label>First Name</label>  
  
      <input placeholder='First Name'  
/>  
  
    </Form.Field>  
  
    <Form.Field>  
  
      <label>Last Name</label>  
  
      <input placeholder='Last Name' />  
    </Form.Field>  
  </Form>  
)
```

```
</Form.Field>
```

```
<Form.Field>
```

```
    <Checkbox label='I agree to the  
Terms and Conditions' />
```

```
</Form.Field>
```

```
<Button type='submit'>Submit</Button>
```

```
</Form>
```

```
)
```

```
export default Create;
```

app.js

E adicione a classe abaixo ao seu arquivo App.css:


```
.create-form label{  
  
    color: whitesmoke !important;  
  
    font-family: 'Montserrat', sans-serif;  
  
    font-size: 12px !important;  
  
}
```

App.css

Esta classe terá como alvo todos os labels dos campos do formulários e aplicará a eles a cor whitesmoke. Ela também mudará a fonte e aumentará o tamanho da fonte.

Em nosso `className main`, adicione a propriedade `flex-direction`. Essa propriedade definirá a orientação como `column`, de modo que cada elemento na `className main` seja alinhado na horizontal.

```
.main{
```

```
display: flex;
```

```
justify-content: center;
```

```
align-items: center;
```

```
height: 100vh;
```

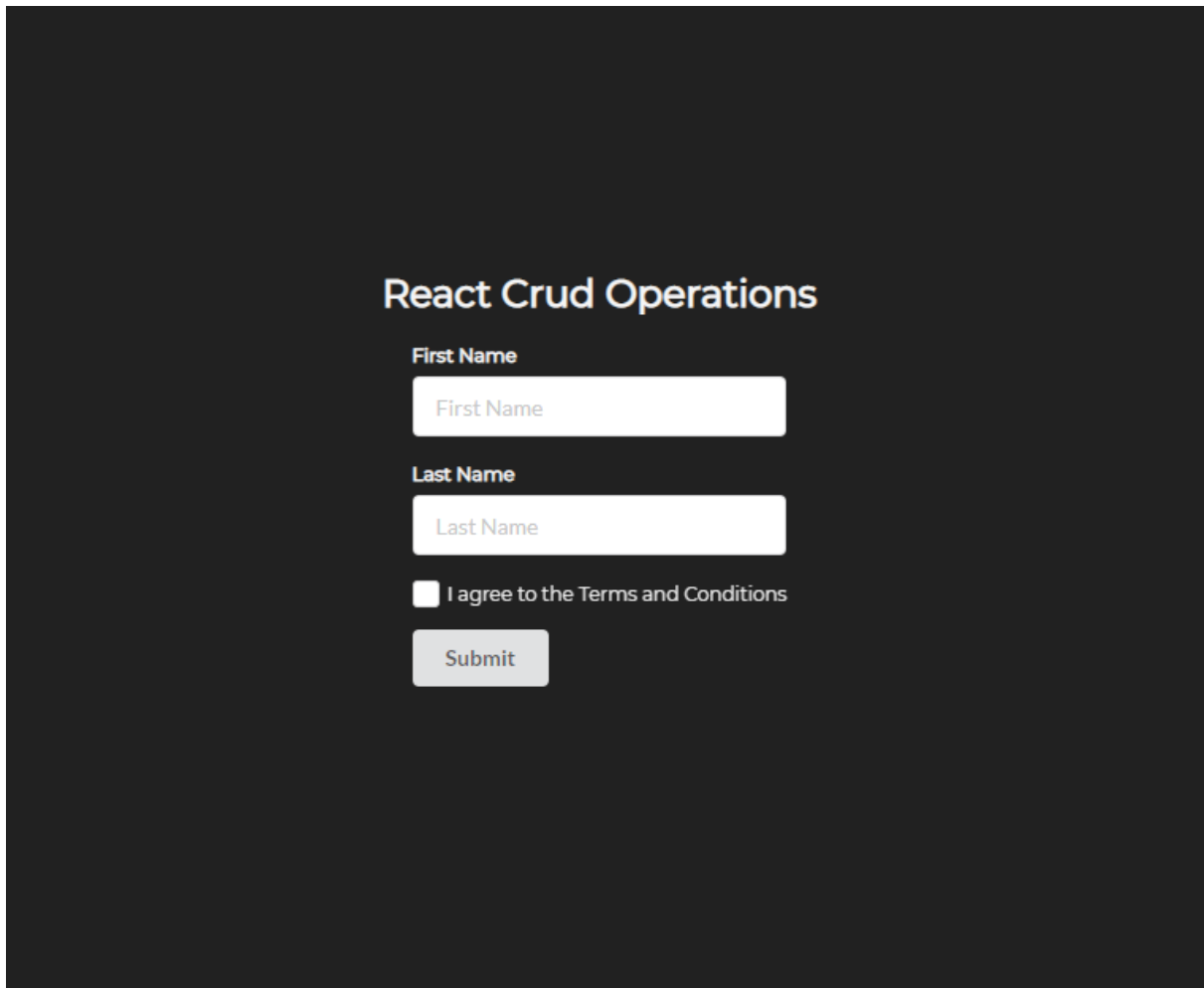
```
background-color: #212121;
```

```
color: whitesmoke;
```

```
flex-direction: column;
```

```
}
```

App.css



React Crud Operations

First Name

Last Name

☐ I agree to the Terms and Conditions

Submit

O formulário, agora, tem uma aparência bem melhor.

Em seguida, vamos obter os dados dos campos do formulário e colocá-los em nosso console. Para isso, usaremos o hook `useState` em React.

No arquivo `Create.js`, importe `useState` de React.

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

Depois, crie states para *first name*, *last name* (nome e sobrenome, respectivamente) e para a caixa de seleção. Inicializaremos os states como vazios ou *false*.

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

```
import { Button, Checkbox, Form } from  
'semantic-ui-react'
```

```
export default function Create() {
```

```
    const [firstName, setFirstName] =  
    useState('');
```

```
    const [lastName, setLastName] =  
    useState('');
```

```
    const [checkbox, setCheckbox] =  
    useState(false);
```

```
return (  
  
  <div>  
  
    <Form className="create-form">  
  
      <Form.Field>  
  
        <label>First Name</label>  
  
        <input placeholder='First  
Name' />  
  
      </Form.Field>  
  
      <Form.Field>  
  
        <label>Last Name</label>  
  
        <input placeholder='Last  
Name' />  

```

```
</Form.Field>
```

```
<Form.Field>
```

```
    <Checkbox label='I agree  
to the Terms and Conditions' />
```

```
</Form.Field>
```

```
    <Button  
type='submit'>Submit</Button>
```

```
</Form>
```

```
</div>
```

```
)
```

```
}
```

Você poderá ver agora que o código está agindo como um componente funcional. Por isso, precisamos transformar o componente em um componente funcional . Somente poderemos usar hooks com esse tipo de componente.

Vamos configurar *first name*, *last name* e a caixa de seleção usando as propriedades `setFirstName`, `setLastName` e `setCheckbox`, respectivamente.

```
<input placeholder='First Name' onChange={(e)
=> setFirstName(e.target.value)}/>
```

```
<input placeholder='Last Name' onChange={(e)
=> setLastName(e.target.value)}/>
```

```
<Checkbox label='I agree to the Terms and
Conditions' onChange={(e) =>
setCheckbox(!checkbox)}/>
```

Agora, estamos capturando os *states* de *first name*, *last name* e da caixa de seleção.

Crie uma função chamada `postData`, que usaremos para enviar dados para a API. Dentro da função, escreveremos este código:

```
const postData = () => {  
  
    console.log(firstName);  
  
    console.log.lastName);  
  
    console.log.checkbox);  
  
}
```

Estamos imprimindo no console o conteúdo de *first name*, *last name* e da caixa de seleção.

No botão Submit, atribua essa função usando o evento `onClick` para que, quando o botão Submit for pressionado, essa função seja chamada.

```
<Button onClick={postData}  
type='submit'>Submit</Button>
```

Aqui temos o código completo para o arquivo *create.js*:

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

```
import { Button, Checkbox, Form } from  
'semantic-ui-react'
```

```
export default function Create() {
```

```
    const [firstName, setFirstName] =  
    useState('');
```

```
    const [lastName, setLastName] =  
    useState('');
```

```
    const [checkbox, setCheckbox] =  
    useState(false);
```

```
    const postData = () => {
```

```
        console.log(firstName);
```

```
        console.log(lastName);
```

```
        console.log(checkbox);
```

```
    }
```

```
    return (
```

```
        <div>
```

```
            <Form className="create-form">
```

```
<Form.Field>
```

```
  <label>First Name</label>
```

```
  <input placeholder='First  
Name' onChange={(e) =>  
setFirstName(e.target.value)}/>
```

```
</Form.Field>
```

```
<Form.Field>
```

```
  <label>Last Name</label>
```

```
  <input placeholder='Last  
Name' onChange={(e) =>  
setLastName(e.target.value)}/>
```

```
</Form.Field>
```

```
<Form.Field>
```

```
        <Checkbox label='I agree  
to the Terms and Conditions' onChange={(e) =>  
setChecked(!checkbox)}/>
```

```
    </Form.Field>
```

```
        <Button onClick={postData}  
type='submit'>Submit</Button>
```

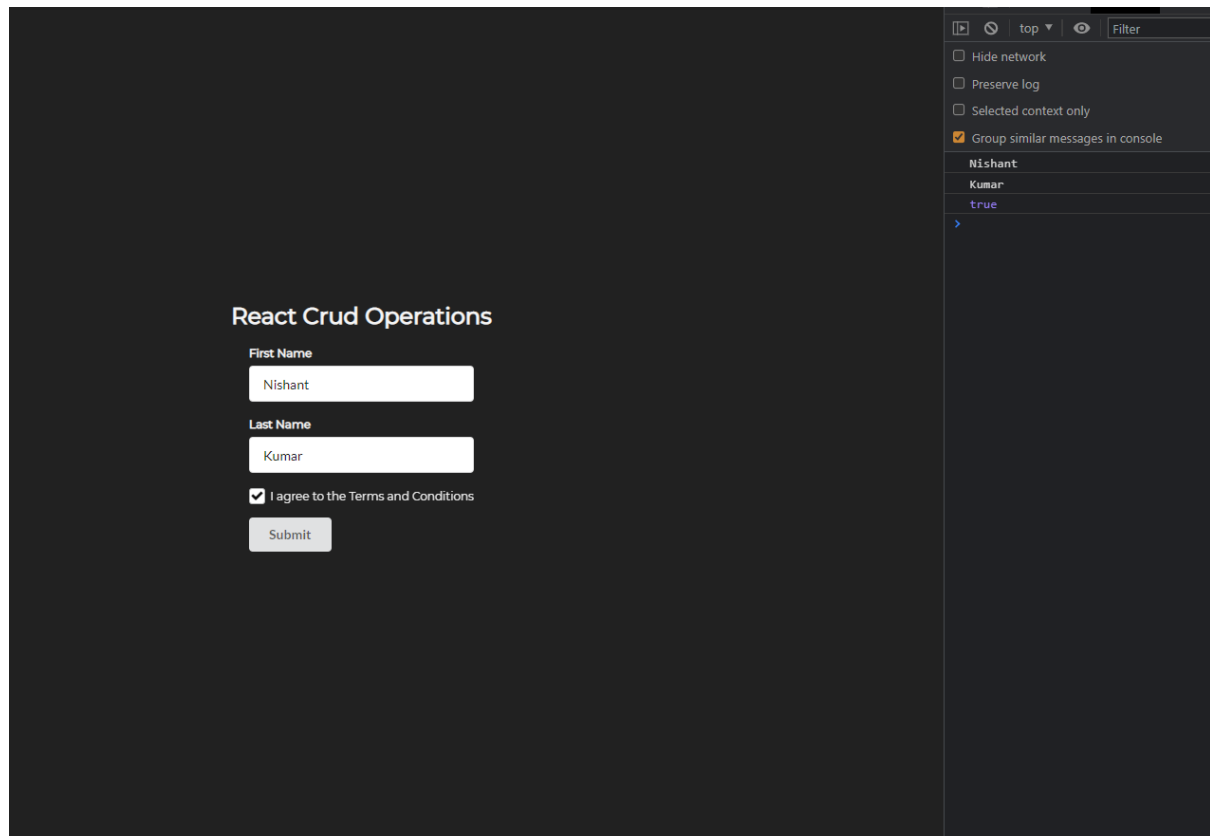
```
    </Form>
```

```
</div>
```

```
)
```

```
}
```

Digite algum valor em *first name* e *last name* e marque a caixa de seleção. Em seguida, clique no botão Submit. Você verá os dados aparecerem no console, assim:

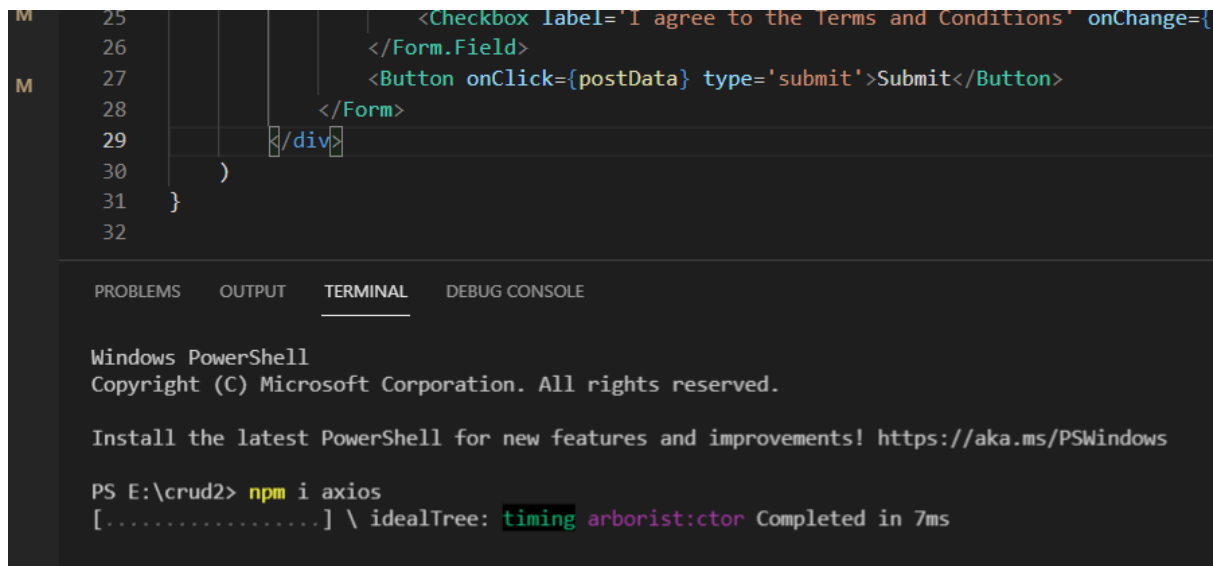


Como usar o Axios para enviar solicitações de API para as Mock APIs

Vamos usar o Axios para enviar os dados do formulário para o servidor mock.

Primeiro, porém, precisamos instalá-lo.

Apenas digite `npm i axios` para instalar o pacote.



The screenshot shows a code editor with a dark theme. The top part displays a JSX element for a form. It includes a checkbox with the label 'I agree to the Terms and Conditions' and an 'onChange' handler, followed by a 'Submit' button with an 'onClick' handler named 'postData'. The code is enclosed in a 'Form.Field' and a 'Form' component, which is then wrapped in a 'div'. Below the code editor, there is a terminal window. The terminal shows the command 'npm i axios' being executed in a Windows PowerShell environment. The output indicates that the package was successfully installed, showing details like 'idealTree', 'timing', 'arborist', and 'ctor'.

```
25 |         <Checkbox label='I agree to the Terms and Conditions' onChange={
26 |         </Form.Field>
27 |         <Button onClick={postData} type='submit'>Submit</Button>
28 |       </Form>
29 |     </div>
30 |   )
31 | }
32 |
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! <https://aka.ms/PSWindows>

PS E:\crud2> npm i axios
[.....] \ idealTree: timing arborist:ctor Completed in 7ms

Após termos instalado o pacote, vamos fazer a operação de criação.

Importe o Axios na parte superior do arquivo.

```
import axios from 'axios';
```

Importação do Axios

Na função `postData`, usaremos o Axios para enviar a solicitação de POST.

```
const postData = () => {
```

```
axios.post(`https://60fbca4591156a0017b4c8a7.
mockapi.io/fakeData`, {

    firstName,

    lastName,

    checkbox

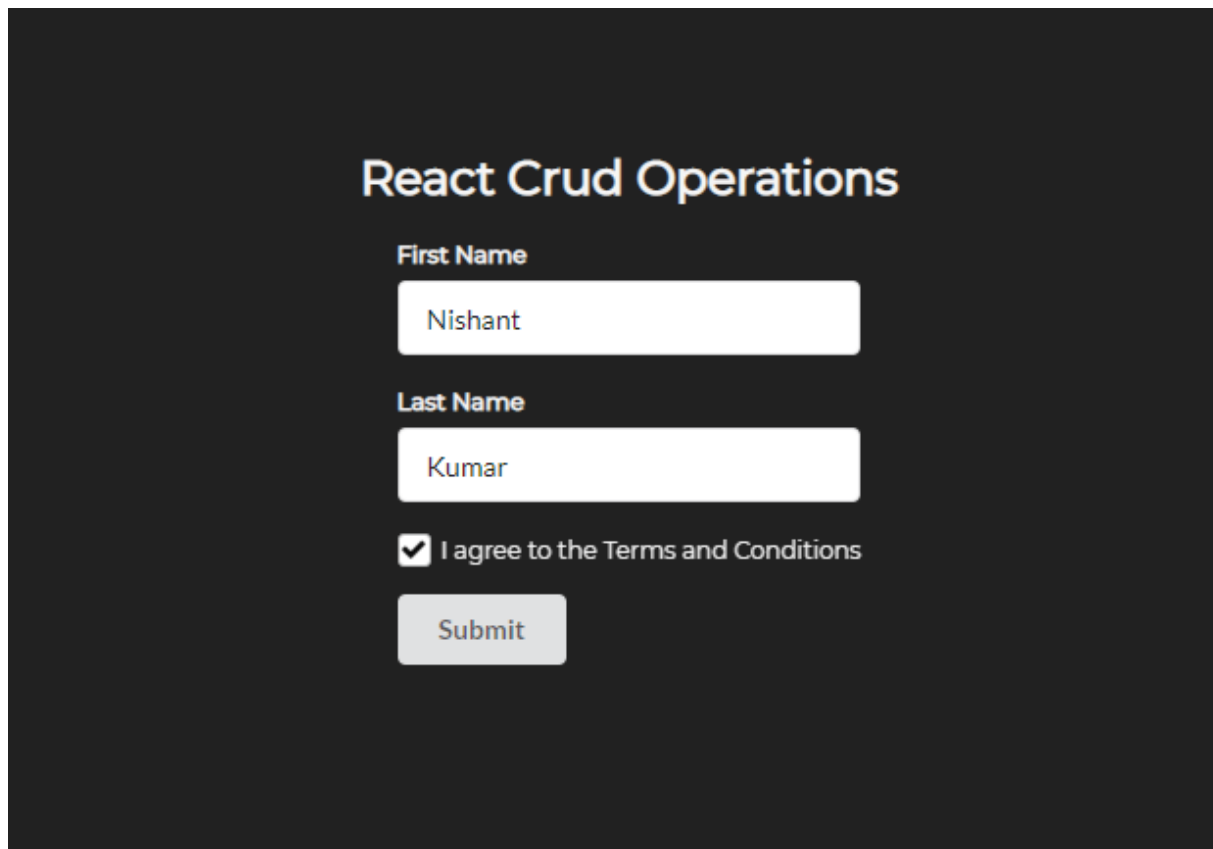
})

}
```

Enviando a solicitação de Post

Como você pode ver, estamos usando `axios.post`. Dentro de `axios.post`, temos o endpoint da API, a qual criamos anteriormente. Em seguida, temos os campos do formulário envolvidos em chaves.

Ao clicarmos em Submit, essa função será chamada e enviará os dados ao servidor da API.



React Crud Operations

First Name

Nishant

Last Name

Kumar

☒ I agree to the Terms and Conditions

Submit

Insira seu nome, sobrenome e marque a caixa de seleção.
Clique em Submit.

```
[{"id":"1","firstName":"Nishant","lastName":"Kumar","checkbox":true}]
```

Se você verificar a API, verá seu nome, sobrenome e a caixa de seleção (ou *checkbox*, em inglês) assinalada como *true*, dentro do objeto.

Como implementar as operações de leitura e atualização

Para iniciar a operação de leitura (Read), precisamos criar uma página de leitura. Também precisaremos do pacote React Router para navegar entre páginas diferentes.

Acesse <https://reactrouter.com/web/guides/quick-start> e instale o pacote usando `npm i react-router-dom`.

Depois de ele ter sido instalado, importamos algumas coisas do React Router:

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

Importando Browser Router como Router e Route do pacote React Router

Em nosso App.js, envolvemos todo o *return* em um Router (roteador). O que isso significa, basicamente, é que tudo o que estiver dentro desse Router poderá usar roteamento em React.

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

```
import Create from './components/create';
```

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

```
function App() {
```

```
  return (
```

```
    <Router>
```

```
      <div className="main">
```

```
        <h2 className="main-header">React  
Crud Operations</h2>
```

```
        <div>
```

```
          <Create />
```

```
        </div>
```

```
</div>
```

```
</Router>
```

```
);
```

```
}
```

```
export default App;
```

Nosso App.js terá, agora, a aparência que vemos acima.

Substitua o Create dentro do return e adicione o código a seguir:

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

```
import Create from './components/create';
```

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

```
function App() {
```

```
  return (
```

```
    <Router>
```

```
      <div className="main">
```

```
        <h2 className="main-header">React  
Crud Operations</h2>
```

```
      <div>
```

```
        <Route exact path='/create'  
component={Create} />
```

```
    </div>
```

```
</div>
```

```
</Router>
```

```
);
```

```
}
```

```
export default App;
```

Aqui, estamos usando o componente Route como Create. Definimos o caminho de Create como '/create'. Assim, se

acessarmos <http://localhost:3000/create>, veremos a página de criação.

Do mesmo modo, precisamos das rotas para leitura e atualização.

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

```
import Create from './components/create';
```

```
import Read from './components/read';
```

```
import Update from './components/update';
```

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

```
function App() {
```

```
  return (
```

```
<Router>
```

```
  <div className="main">
```

```
    <h2 className="main-header">React  
    Crud Operations</h2>
```

```
    <div>
```

```
      <Route exact path='/create'  
      component={Create} />
```

```
    </div>
```

```
    <div style={{ marginTop: 20 }}>
```

```
      <Route exact path='/read'  
      component={Read} />
```

```
    </div>
```

```
      <Route path='/update'  
component={Update} />
```

```
    </div>
```

```
  </Router>
```

```
);
```

```
}
```

```
export default App;
```

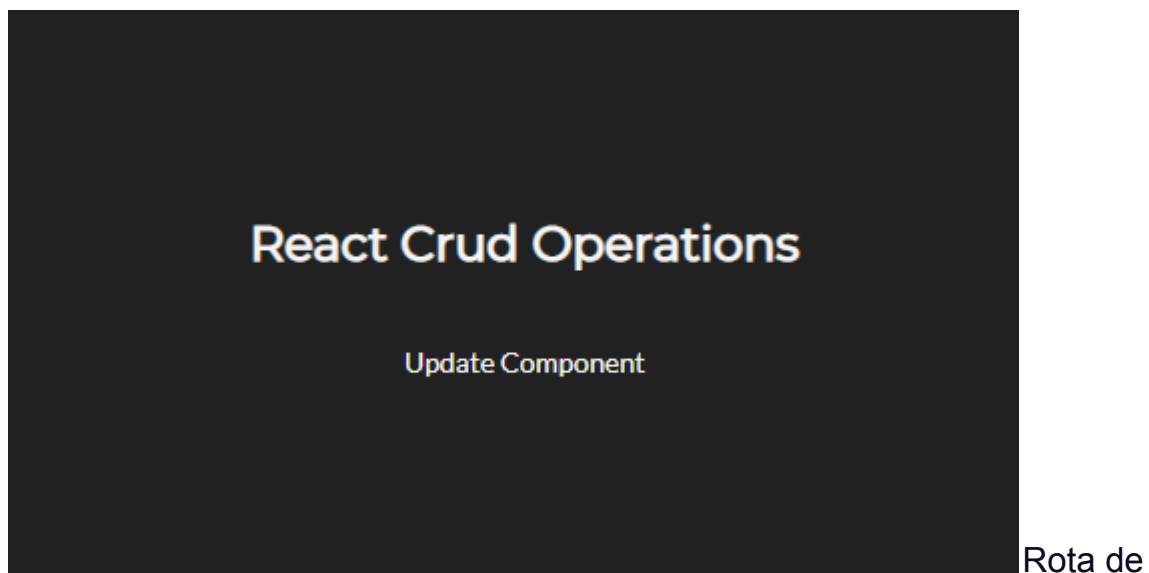
Assim, crie as rotas de leitura e atualização (*read* e *update*) da mesma forma que você vê acima.

Se você acessar <http://localhost:3000/read>, verá o seguinte:



leitura (Read)

E no URL <http://localhost:3000/update>, teremos o componente de atualização (Update) assim:



atualização (Update)

A operação de leitura

Para a operação de leitura (Read), precisaremos de um componente Table (tabela). Então, vá para a Semantic UI do React e use uma tabela da biblioteca.

```
import React from 'react';

import { Table } from 'semantic-ui-react'

export default function Read() {

  return (

    <div>

      <Table singleLine>

        <Table.Header>

          <Table.Row>

            <Table.HeaderCell>Name</Table.HeaderCell>
```

```
<Table.HeaderCell>Registration  
Date</Table.HeaderCell>
```

```
<Table.HeaderCell>E-mail  
address</Table.HeaderCell>
```

```
<Table.HeaderCell>Premium  
Plan</Table.HeaderCell>
```

```
</Table.Row>
```

```
</Table.Header>
```

```
<Table.Body>
```

```
<Table.Row>
```

<div><Table.Cell>John Lilki</Table.Cell></div>	<div><Table.Cell>September 14, 2013</Table.Cell></div>
<div><Table.Cell>jhlilk22@yahoo.com</Table.Cell></div>	
<div><Table.Cell>No</Table.Cell></div>	
<div></Table.Row></div>	
<div><Table.Row></div>	
<div><Table.Cell>Jamie Harrington</Table.Cell></div>	<div><Table.Cell>January 11, 2014</Table.Cell></div>

<Table.Cell>jamieharingonton@yahoo.com</Table
.Cell>

<Table.Cell>Yes</Table.Cell>

</Table.Row>

<Table.Row>

<Table.Cell>Jill
Lewis</Table.Cell>

<Table.Cell>May 11,
2014</Table.Cell>

<Table.Cell>jilsewris22@yahoo.com</Table.Cell
>

```
<Table.Cell>Yes</Table.Cell>
```

```
</Table.Row>
```

```
</Table.Body>
```

```
</Table>
```

```
</div>
```

```
)
```

```
}
```

Read.js

Aqui, você pode ver uma tabela com alguns dados de teste. Temos apenas uma linha de tabela. Assim, removeremos o resto.

```
import React from 'react';

import { Table } from 'semantic-ui-react'

export default function Read() {

  return (

    <div>

      <Table singleLine>

        <Table.Header>

          <Table.Row>

            <Table.HeaderCell>Name</Table.HeaderCell>
```

```
<Table.HeaderCell>Registration  
Date</Table.HeaderCell>
```

```
<Table.HeaderCell>E-mail  
address</Table.HeaderCell>
```

```
<Table.HeaderCell>Premium  
Plan</Table.HeaderCell>
```

```
</Table.Row>
```

```
</Table.Header>
```

```
<Table.Body>
```

```
<Table.Row>
```



```
<Table.Cell>John
Lilki</Table.Cell>

<Table.Cell>September
14, 2013</Table.Cell>

<Table.Cell>jhlilk22@yahoo.com</Table.Cell>

<Table.Cell>No</Table.Cell>

</Table.Row>

</Table.Body>


</Table>

</div>

)
```

}

Read.js



The screenshot shows a web application with a dark background. At the top, the title 'React Crud Operations' is displayed in a light-colored font. Below the title is a white table with four columns: 'Name', 'Registration Date', 'E-mail address', and 'Premium Plan'. The table contains one row of data: 'John Lilki', 'September 14, 2013', 'jhlilk22@yahoo.com', and 'No'.

Name	Registration Date	E-mail address	Premium Plan
John Lilki	September 14, 2013	jhlilk22@yahoo.com	No

Esse é o resultado da página de leitura (Read). Temos uma tabela com quatro colunas, mas somente precisamos de três.

Remova as colunas adicionais de campo e renomeie os campos assim:

React Crud Operations

First Name	Last Name	Checked
Nishant	Kumar	Yes

Esta é a aparência de nossa página de leitura agora:

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

```
import { Table } from 'semantic-ui-react'
```

```
export default function Read() {
```

```
  return (
```

```
    <div>
```

```
      <Table singleLine>
```

<Table.Header>

<Table.Row>

<Table.HeaderCell>First
Name</Table.HeaderCell>

<Table.HeaderCell>Last
Name</Table.HeaderCell>

<Table.HeaderCell>Checked</Table.HeaderCell>

</Table.Row>

</Table.Header>

<Table.Body>

<Table.Row>

<Table.Cell>Nishant</Table.Cell>

<Table.Cell>Kumar</Table.Cell>

<Table.Cell>Yes</Table.Cell>

</Table.Row>

</Table.Body>

</Table>

</div>

```
)  
  
}
```

Read.js

Em seguida, vamos enviar uma solicitação de GET para obter os dados da API.

Precisamos dos dados quando nossa aplicação carregar. Assim, vamos usar o hook `useEffect`.

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

```
useEffect(() => {
```

```
}, [])
```

Hook `useEffect`

Crie um *state* que conterà os dados recebidos. Este state será um array.

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

```
const [APIData, setAPIData] = useState([]);
```

```
useEffect(() => {
```

```
}, [])
```

State de `APIData` para armazenar os dados recebidos da API

No hook `useEffect`, vamos enviar a solicitação de GET.

```
useEffect(() => {
```

```
axios.get(`https://60fbca4591156a0017b4c8a7.m  
ockapi.io/fakeData`)  
  
  .then((response) => {  
  
    setAPIData(response.data);  
  
  })  
  
}, [])
```

Assim, usaremos `axios.get` para enviar a solicitação de GET à API. Então, se a solicitação for preenchida, definiremos os dados da resposta em nosso *state* da *APIData*.

Agora, vamos mapear nossas linhas da tabela de acordo com os dados da API.

Vamos usar a função `Map` para fazer isso. A função vai percorrer o array e exibir os dados no resultados.


```
<Table.Body>
```

```
  {APIData.map((data) => {
```

```
    return (
```

```
      <Table.Row>
```

```
        <Table.Cell>{data.firstName}</Table.Cell>
```

```
        <Table.Cell>{data.lastName}</Table.Cell>
```

```
          <Table.Cell>{data.checkbox ?
```

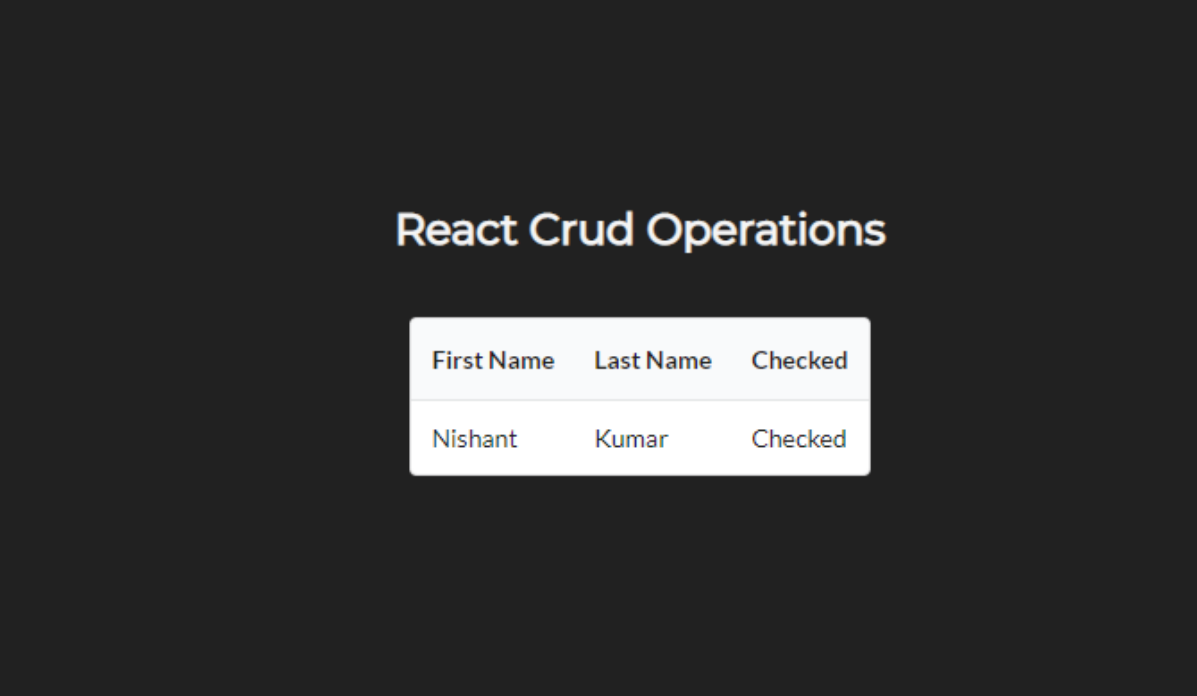
```
            'Checked' : 'Unchecked'}</Table.Cell>
```

```
        </Table.Row>
```

```
      )})}
```

```
</Table.Body>
```

Estamos mapeando *firstName*, *lastName* e a caixa de seleção de acordo com os dados na API. Nossa caixa de seleção, no entanto, é um pouco diferente. Usei aqui um operador ternário ('?'). Se *data.checkbox* for *true*, o resultado será *Checked* (marcado). Do contrário, será *Unchecked* (desmarcado).



The screenshot shows a web application with a dark background. At the top, the title 'React Crud Operations' is displayed in a light green font. Below the title is a white table with three columns: 'First Name', 'Last Name', and 'Checked'. The table contains one row of data with the values 'Nishant', 'Kumar', and 'Checked'.

First Name	Last Name	Checked
Nishant	Kumar	Checked

Resultado de Read.js

A operação de atualização

Crie mais um cabeçalho para a atualização (Update) e uma coluna para cada linha da tabela para um botão de atualização. Use o botão de Semantic UI do React.

```
<Table.HeaderCell>Update</Table.HeaderCell>
```

```
<Table.Cell>
```

```
  <Button>Update</Button>
```

```
</Table.Cell>
```

Criação do botão Update

Agora, ao clicarmos nesse botão, deveremos ser redirecionados para a página de atualização. Para isso, precisamos do Link de React Router.

Importe Link de React Router. Envolve a célula da tabela para o botão Update com as tags para Link.

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

```
<Link to='/update'>
```

```
  <Table.Cell>
```

```
    <Button>Update</Button>
```

```
  </Table.Cell>
```

```
</Link>
```

Link para o botão Update

Desse modo, se clicarmos no botão Update, seremos redirecionados para a página de atualização.

Para atualizar os dados da coluna, precisaremos dos ID respectivos, os quais obteremos na API.

Crie uma função chamada `setData`. Vincule essa função ao botão Update.

```
<Button onClick={() =>
setData()}>Update</Button>
```

Depois disso, precisamos passar os dados como parâmetro para a função superior.

```
<Button onClick={() =>
setData(data)}>Update</Button>
```

Na função acima, registramos esses dados no console:

```
const setData = (data) => {

  console.log(data);

}
```

```
▼ {id: "1", firstName: "Nishant", lastName: "Kumar", checkbox: true} ⓘ read.js:17  
  checkbox: true  
  firstName: "Nishant"  
  id: "1"  
  lastName: "Kumar"  
  ► __proto__: Object
```

Dados no console

Clique no botão Update na tabela e confira o console. Você obterá os dados do campo da tabela respectivo.

Vamos definir esses dados no localStorage.

```
const setData = (data) => {  
  
    let { id, firstName, lastName,  
checkbox } = data;  
  
    localStorage.setItem('ID', id);  
  
    localStorage.setItem('First Name',  
firstName);  
  
    localStorage.setItem('Last Name',  
lastName);
```

```
        localStorage.setItem('Checkbox  
Value', checkbox)  
  
    }
```

Definindo os dados no localStorage (Armazenamento local)

Estamos fazendo a desestruturação dos nossos dados em *id*, *firstName*, *lastName*, e *checkbox*. Em seguida, definimos esses dados no armazenamento local. Você pode usar o armazenamento local para armazenar os dados localmente no navegador.

Em seguida, no componente Update, precisamos de um formulário para a operação de atualização. Vamos reutilizar o formulário do componente Create. Apenas mude o nome da função de Create para Update.

```
import React, { useState } from 'react';  
  
import { Button, Checkbox, Form } from  
'semantic-ui-react'
```

```
import axios from 'axios';

export default function Update() {

    const [firstName, setFirstName] =
    useState('');

    const [lastName, setLastName] =
    useState('');

    const [checkbox, setCheckbox] =
    useState(false);

    return (

        <div>
```



```
<Form className="create-form">
```

```
  <Form.Field>
```

```
    <label>First Name</label>
```

```
    <input placeholder='First  
Name' onChange={(e) =>  
setFirstName(e.target.value)}/>
```

```
  </Form.Field>
```

```
  <Form.Field>
```

```
    <label>Last Name</label>
```

```
    <input placeholder='Last  
Name' onChange={(e) =>  
setLastName(e.target.value)}/>
```

```
  </Form.Field>
```

```
        <Form.Field>

            <Checkbox label='I agree
to the Terms and Conditions' onChange={(e) =>
setChecked(!checkbox)}/>

        </Form.Field>

        <Button
type='submit'>Update</Button>

    </Form>

</div>

)

}
```

[Aprenda a programar — currículo gratuito de 3 mil horas](#)

29 DE MARÇO DE 2022/[#REACT](#)

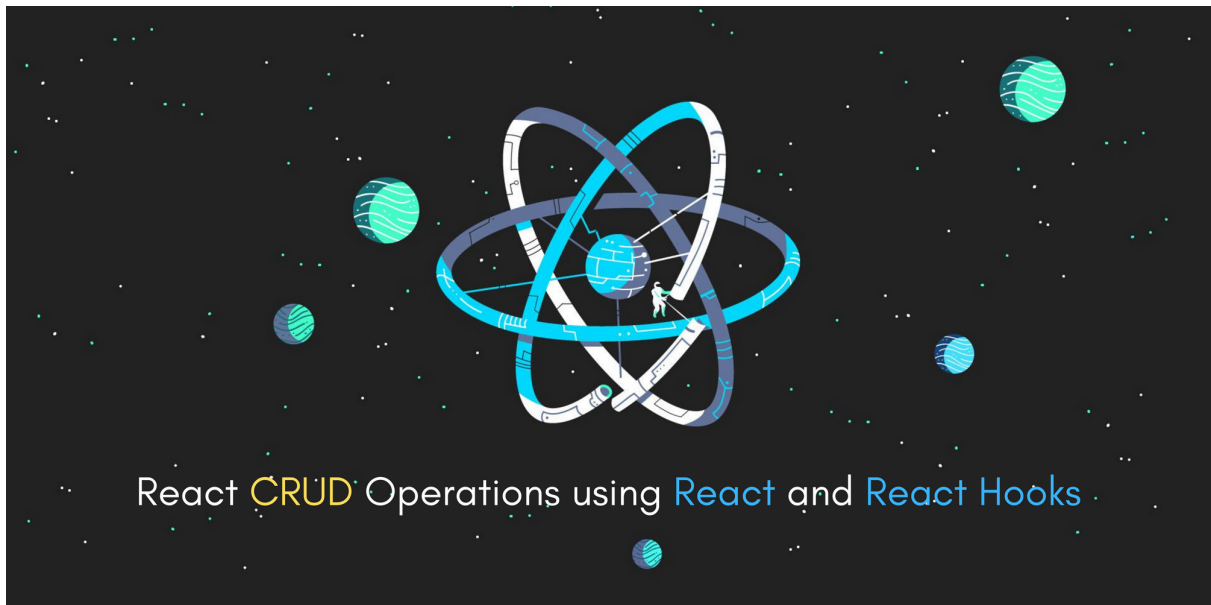
Como realizar operações de CRUD usando React, hooks do React e Axios



Tradutor: [Daniel Rosa](#)



Autor: Nishant Kumar (em inglês)



Artigo original: [How to Perform CRUD Operations using React, React Hooks, and Axios](#)

Se você está trabalhando com React, pode ser difícil entender e implementar solicitações de API.

Neste artigo, aprenderemos como tudo funciona implementando operações de CRUD usando React, hooks do React, React Router e Axios.

Vamos lá.

Como instalar o Node e o npm

Primeiramente, vamos instalar o Node em nosso sistema. Vamos usá-lo, primeiramente, para executar nosso código em JavaScript.

Para baixar o Node, acesse <https://nodejs.org/en/>.

Você também precisará do **node package manager**, ou npm, que já vem integrado ao Node. Você pode usá-lo para instalar os pacotes para seus apps em JavaScript.

Felizmente, como ele vem com o Node, não é necessário baixá-lo separadamente.

Depois de baixá-los e instalá-los, abra seu terminal ou o prompt de comando e digite `node -v`. Assim, você poderá conferir a versão do Node que você tem.

Como criar sua aplicação em React

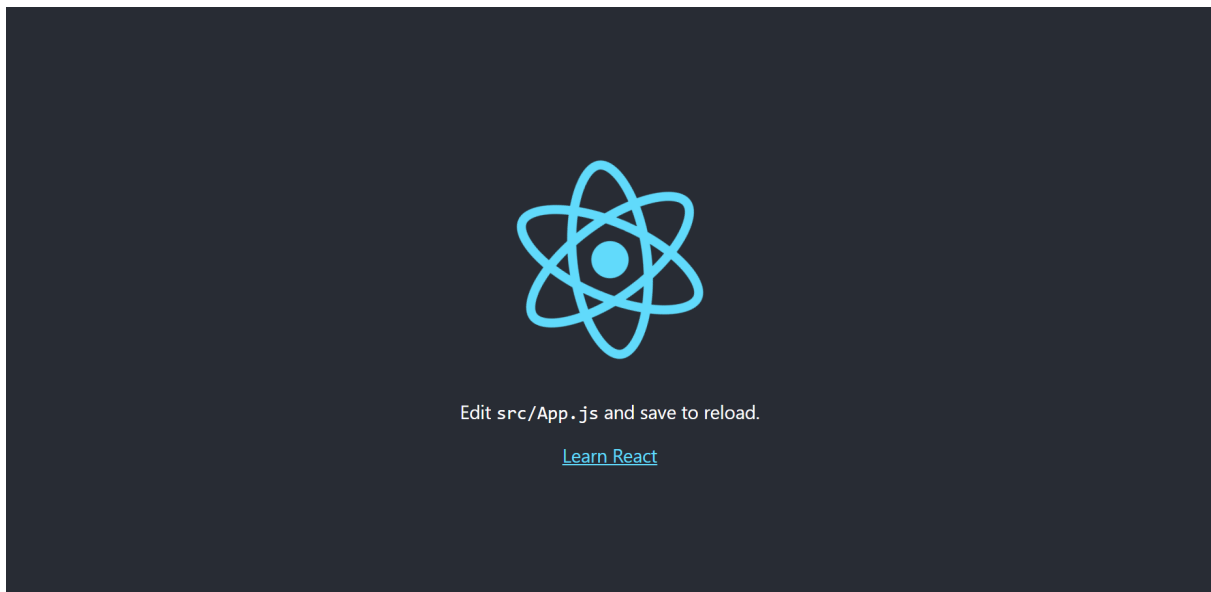
Para criar sua aplicação em React, digite

`npx-create-react-app <nome-do-seu-app>` no seu terminal ou `npx-create-react-app react-crud`, neste caso.

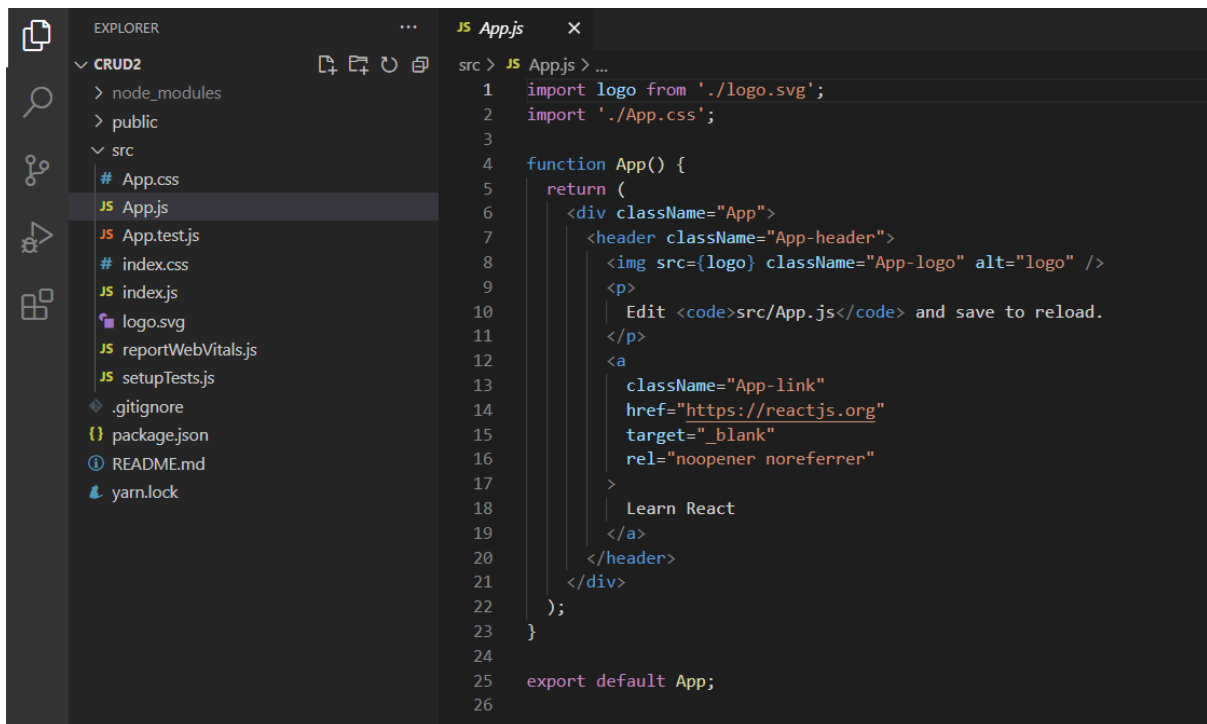
Você verá os pacotes serem instalados.

Ao terminar de baixar os pacotes, vá até a pasta do projeto e digite `npm start`.

Você verá o modelo padrão do React, que tem essa aparência:



O boilerplate padrão do React



```
src > JS App.js > ...
1  import logo from './logo.svg';
2  import './App.css';
3
4  function App() {
5    return (
6      <div className="App">
7        <header className="App-header">
8          <img src={logo} className="App-logo" alt="logo" />
9          <p>
10             Edit <code>src/App.js</code> and save to reload.
11          </p>
12          <a
13            className="App-link"
14            href="https://reactjs.org"
15            target="_blank"
16            rel="noopener noreferrer"
17          >
18            Learn React
19          </a>
20        </header>
21      </div>
22    );
23  }
24
25  export default App;
26
```

Nosso arquivo App.js

Como instalar o pacote Semantic UI para o React

Vamos instalar o pacote Semantic UI para o React em nosso projeto. Semantic UI é uma biblioteca de UI (interface de usuário) feita para o React que tem componentes de UI pré-construídos, como tabelas, botões e muitos outros recursos.

Você pode instalar o pacote usando um dos comandos abaixo, dependendo do seu gerenciador de pacotes.

```
yarn add semantic-ui-react semantic-ui-css
```


Para o gerenciador de pacotes do Yarn

```
npm install semantic-ui-react semantic-ui-css
```

Para o gerenciador de pacotes do Node, o NPM

Além disso, importe a biblioteca em seu arquivo de entrada principal do app, chamado index.js.

```
import 'semantic-ui-css/semantic.min.css'
```

Cole isto no seu arquivo index.js

Como criar sua aplicação com CRUD

Agora, vamos começar a criar nossa aplicação com CRUD usando o React.

Primeiro, vamos adicionar um título à nossa aplicação.

Em nosso arquivo app.js, teremos o seguinte:

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

```
function App() {  
  
  return (  
  
    <div>  
  
      React Crud Operations  
  
    </div>  
  
  );  
  
}  
  
export default App;
```

Adicionando um título à nossa aplicação

Agora, vamos garantir que ela esteja centralizada.

Dê à `div` pai a `classname main`. No arquivo `App.css`, usaremos `Flexbox` para centralizar o título.

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

```
function App() {
```

```
  return (
```

```
    <div className="main">
```

```
      React Crud Operations
```

```
    </div>
```

```
  );
```

```
}
```

```
export default App;
```

app.js com a className main na div pai

```
.main{
```

```
display: flex;
```

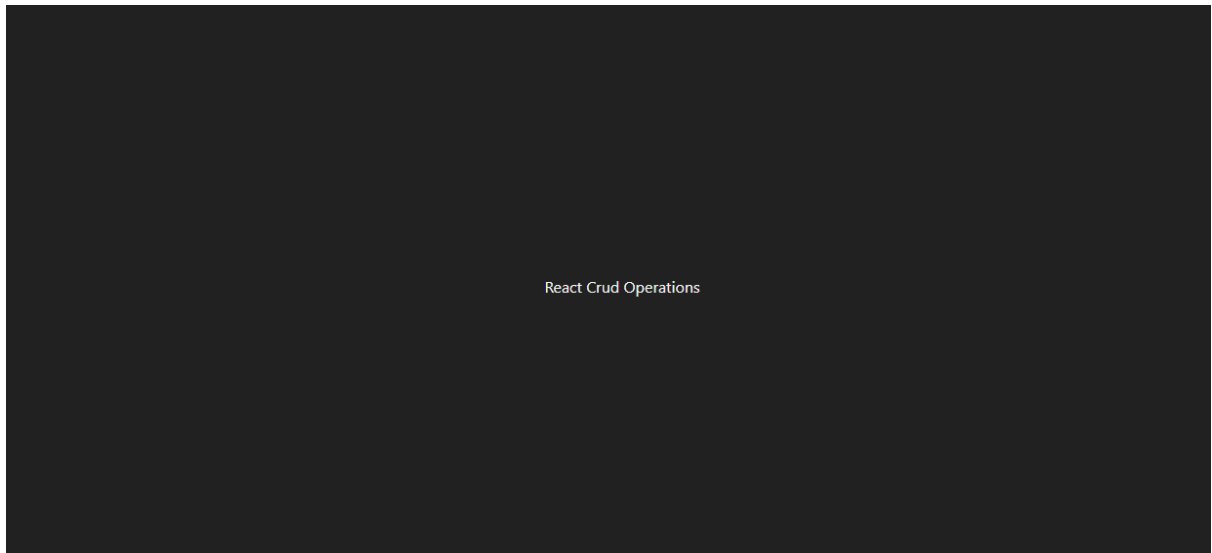
```
justify-content: center;
```

```
align-items: center;
```

```
height: 100vh;
```

```
}
```

Nosso arquivo app.css



Nosso título, agora, está centralizado com perfeição.

Agora que a aparência está melhor, precisamos colocá-lo em realce e adicionar umas fontes legais. Para fazer isso, usaremos as tags ao redor do título, assim:

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

```
function App() {
```

```
return (  
  
  <div className="main">  
  
    <h2 className="main-header">React Crud  
Operations</h2>  
  
  </div>  
  
);  
  
}  
  
export default App;
```

Vamos importar uma Google Font. Acesse
<https://fonts.google.com/> para escolhermos uma.

Selecione a fonte de sua preferência. Para o exemplo, usaremos a família de fontes Montserrat.

Importe a sua fonte preferida no arquivo App.css, assim:

```
@import  
url('https://fonts.googleapis.com/css2?family  
=Montserrat&display=swap');
```

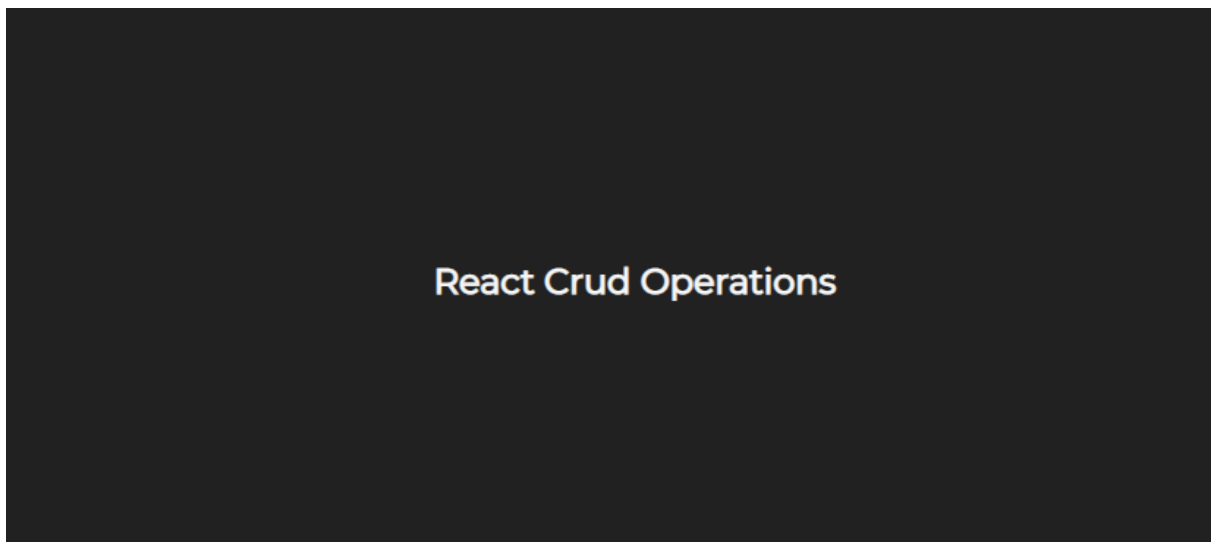
Agora, vamos mudar a fonte do título.

```
<div className="main">  
  
    <h2 className="main-header">React Crud  
Operations</h2>  
  
</div>
```

Dê à tag do título a `className main-header`, assim.

Em seguida, no seu App.css, adicione a família da fonte:

```
.main-header{  
  
  font-family: 'Montserrat', sans-serif;  
  
}
```

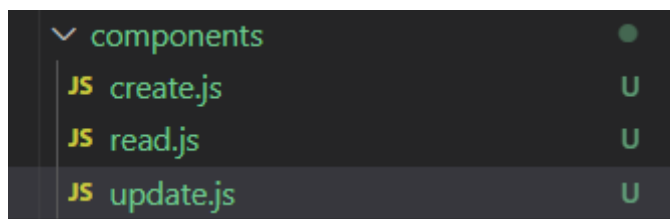


Agora, você verá o título modificado.

Como criar seus componentes de CRUD

Vamos criar quatro componentes de CRUD – Create, Read, Update e Delete (criar, ler, atualizar e excluir, respectivamente).

Na nossa pasta *src*, criamos uma pasta chamada *components*. Dentro dela, criamos três arquivos – *create.js*, *read.js* e *update.js*. Para a exclusão (*delete*), não precisamos de um componente adicional.



Agora, vamos implementar a operação de criação.

Para isso, precisamos usar APIs *mock* ou de simulação. Essas APIs enviarão dados ao servidor falso que criaremos, apenas para fins didáticos.

Assim, acesse <https://mockapi.io/> e crie sua conta.

mockapi.io

The easiest way to mock REST APIs! (Check out [docs](#))

GET STARTED

DEMO PROJECT

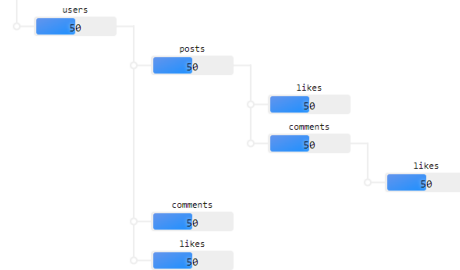
API endpoint

https://SECRET.mockapi.io/:endpoint

NEW RESOURCE

GENERATE ALL

RESET ALL



MockAPI




Criar um projeto clicando no botão +.



Clique no botão + para criar um novo projeto

The image shows a modal dialog for creating a new project. It has a close button (X) in the top right corner. The form contains three sections: 1. 'Project name' with a text input field containing the placeholder 'Example: Todo App, Project X...'. 2. 'API Prefix (optional)' with a text input field containing the placeholder 'Example: /api/v1'. Below this input is a description: 'Add API prefix to all endpoints in this project.' 3. 'Collaborators (optional)' with a text input field containing the placeholder 'Search by name...'. Below this input is a description: 'Collaborators can create, update, and delete resources in this project.' At the bottom of the modal, there are two buttons: a blue 'CREATE' button and a grey 'CANCEL' button.

Adicione o nome do projeto e clique no botão Create (Criar).

Projects →  CRUD  

API endpoint
https://60fbca4591156a0017b4c8a7.mockapi.io/:endpoint

[NEW RESOURCE](#) [GENERATE ALL](#) [RESET ALL](#)

🔍 No resources yet...

Em seguida, crie um novo recurso clicando no botão NEW RESOURCE (Novo recurso).

Resource name

Enter meaningful resource name, it will be used to generate API endpoints.

Example: users, comments, articles...

Schema (optional)

Define Resource schema, it will be used to generate mock data.

id	Object ID	
createdAt	Faker.js	Recent
name	Faker.js	Find name
avatar	Faker.js	Avatar
+		

Object template (optional)

To define more complex structure for your data use JSON template. You can reference Faker.js methods using `\$.`.

```
{
  "username": "$internet.userName",
  "knownIps": ["$internet.ip", "$internet.ipv6"],
  "profile": {
    "firstName": "$name.firstName",
    "lastName": "$name.lastName",
    "staticData": [100, 200, 300]
  }
}
```

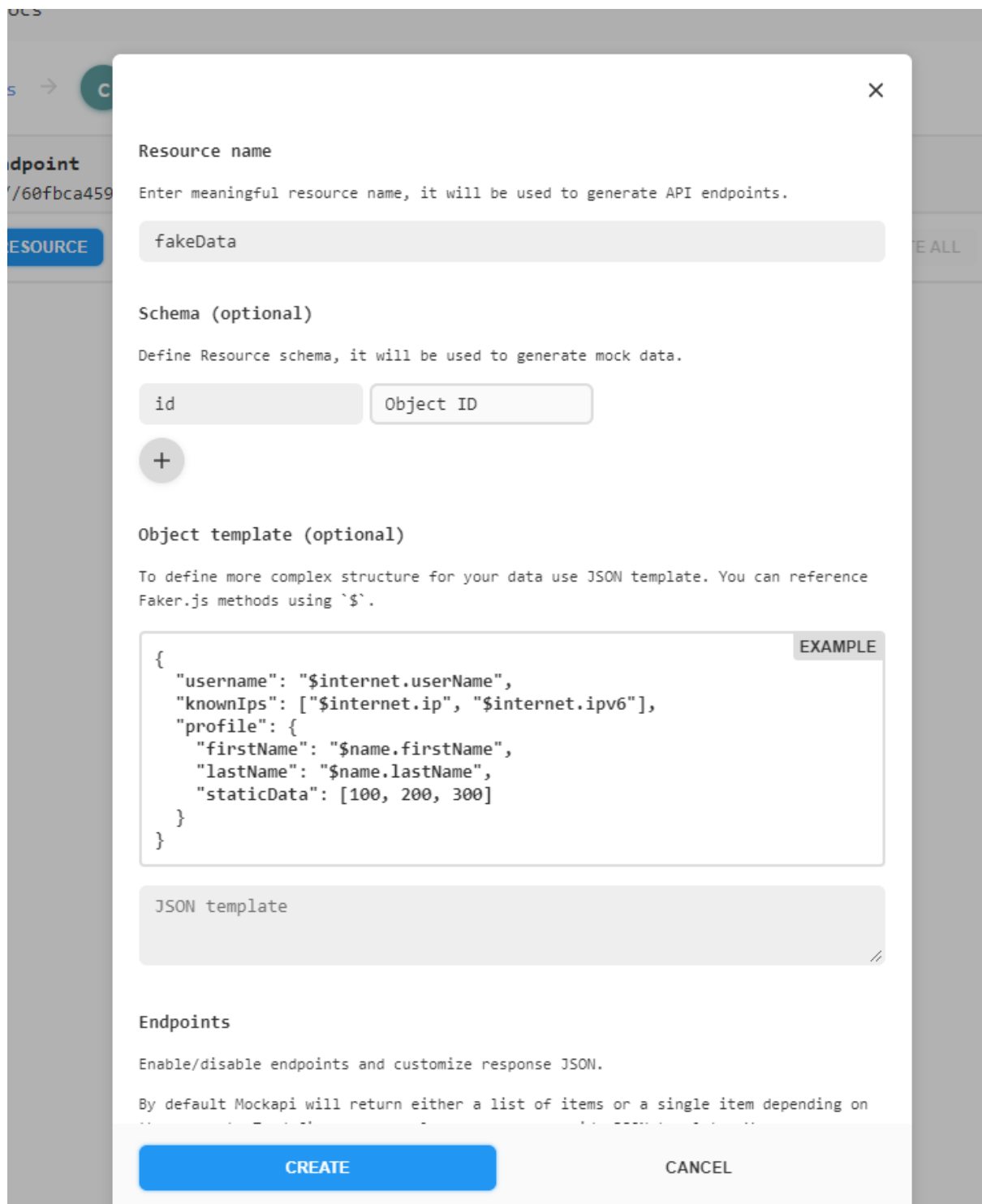
EXAMPLE

JSON template

CREATE

CANCEL

Será solicitado o nome do recurso (em *Resource Name*).
Coloque ali o nome que achar apropriado.

A screenshot of a web application showing a modal dialog for creating a new resource in Mockapi. The dialog has a close button (X) in the top right corner. It contains several sections: 'Resource name' with a text input containing 'fakeData'; 'Schema (optional)' with a text input containing 'id' and a dropdown menu set to 'Object ID'; 'Object template (optional)' with a text area containing a JSON template and an 'EXAMPLE' button; and 'Endpoints' with a text area containing 'JSON template'. At the bottom, there are two buttons: 'CREATE' (blue) and 'CANCEL' (gray).

Resource name

Enter meaningful resource name, it will be used to generate API endpoints.

fakeData

Schema (optional)

Define Resource schema, it will be used to generate mock data.

id Object ID

+

Object template (optional)

To define more complex structure for your data use JSON template. You can reference Faker.js methods using `\$.`.

`{
 "username": "$internet.userName",
 "knownIps": ["$internet.ip", "$internet.ipv6"],
 "profile": {
 "firstName": "$name.firstName",
 "lastName": "$name.lastName",
 "staticData": [100, 200, 300]
 }
}`

EXAMPLE

JSON template

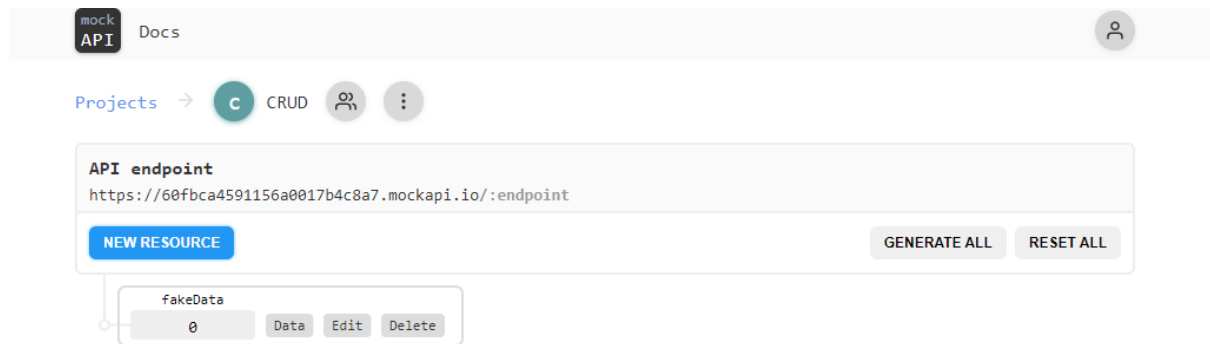
Endpoints

Enable/disable endpoints and customize response JSON.

By default Mockapi will return either a list of items or a single item depending on

CREATE CANCEL

Remova os campos adicionais, como name, avatar e createdAt, pois não precisaremos deles. Então, clique em Create (Criar).



Criamos nossa API falsa, que chamei de fakeData (dados falsos).

Clique em fakeData e você verá a API sendo aberta em uma nova guia. O banco de dados agora está vazio.

Como criar um formulário para o componente Create

Vamos usar um formulário da biblioteca Semantic UI.

Vá até Semantic UI, e procure por *Form* na barra de pesquisa à esquerda.

Form

A form.

Try it

CodeSandbox

Maximize

Permalink

Forms also have a robust shorthand props API for generating controls wrapped in `FormFields`. See shorthand examples below.

First Name

Last Name

☐ I agree to the Terms and Conditions

Submit

Você verá um formulário assim. Clique em Try it (Experimente) na parte superior esquerda para obter o código.

Form

A form.

Try it

CodeSandbox

Maximize

Permalink

Forms also have a robust shorthand props API for generating controls wrapped in `FormFields`. See shorthand examples below.

First Name

Last Name

☐ I agree to the Terms and Conditions

Submit

```
1 import React from 'react'
2 import { Button, Checkbox, Form } from 'semantic-ui-react'
3
4 const FormExampleForm = () => (
5   <Form>
6     <Form.Field>
7       <label>First Name</label>
8       <input placeholder='First Name' />
9     </Form.Field>
10    <Form.Field>
11      <label>Last Name</label>
12      <input placeholder='Last Name' />
13    </Form.Field>
14    <Form.Field>
15      <Checkbox label='I agree to the Terms and Conditions' />
16    </Form.Field>
17    <Button type='submit'>Submit</Button>
18  </Form>
19 )
20
21 export default FormExampleForm
22
```

Copie este código e cole-o no seu arquivo `Create.js`, assim:

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



```
import { Button, Checkbox, Form } from  
'semantic-ui-react'
```

```
const Create = () => (
```

```
  <Form>
```

```
    <Form.Field>
```

```
      <label>First Name</label>
```

```
      <input placeholder='First Name'
```

```
    />
```

```
    </Form.Field>
```

```
    <Form.Field>
```

```
      <label>Last Name</label>
```

```
<input placeholder='Last Name' />
```

```
</Form.Field>
```

```
<Form.Field>
```

```
<Checkbox label='I agree to the  
Terms and Conditions' />
```

```
</Form.Field>
```

```
<Button type='submit'>Submit</Button>
```

```
</Form>
```

```
)
```

```
export default Create;
```

Importe o componente Create no seu arquivo app.js.

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

```
import Create from './components/create';
```

```
function App() {
```

```
  return (
```

```
    <div className="main">
```

```
      <h2 className="main-header">React Crud  
Operations</h2>
```

```
      <div>
```

```
        <Create/>
```

```
</div>
```

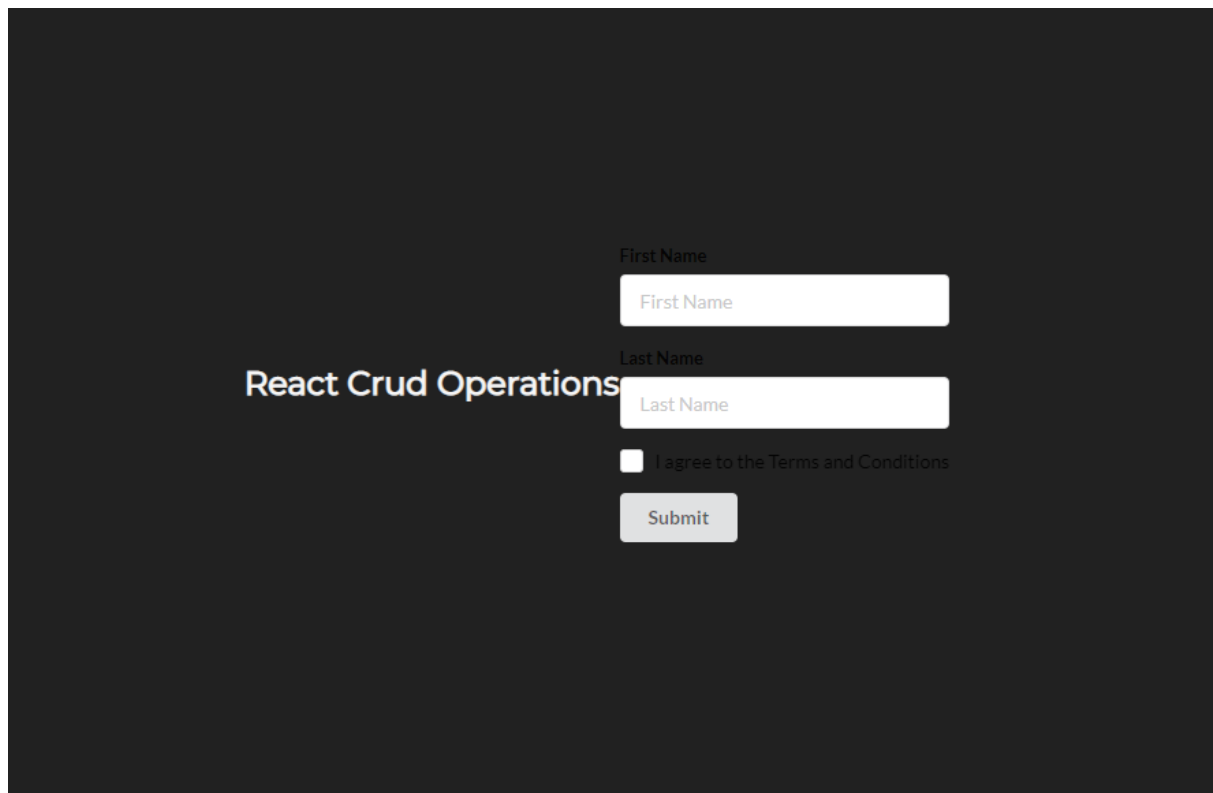
```
</div>
```

```
);
```

```
}
```

```
export default App;
```

Este é o resultado:



Temos um problema agora – os itens não estão alinhados adequadamente e as cores dos labels para os inputs de texto estão em preto. Vamos mudar isso.

No arquivo Create.js, dê à **Form** a className de `create-form`.

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

```
import { Button, Checkbox, Form } from  
'semantic-ui-react'
```

```
const Create = () => (  
  
  <Form className="create-form">  
  
    <Form.Field>  
  
      <label>First Name</label>  
  
      <input placeholder='First Name'  
/>  
  
    </Form.Field>  
  
    <Form.Field>  
  
      <label>Last Name</label>  
  
      <input placeholder='Last Name' />  
    </Form.Field>  
  </Form>  
)
```

```
</Form.Field>
```

```
<Form.Field>
```

```
    <Checkbox label='I agree to the  
Terms and Conditions' />
```

```
</Form.Field>
```

```
<Button type='submit'>Submit</Button>
```

```
</Form>
```

```
)
```

```
export default Create;
```

app.js

E adicione a classe abaixo ao seu arquivo App.css:

```
.create-form label{  
  
    color: whitesmoke !important;  
  
    font-family: 'Montserrat', sans-serif;  
  
    font-size: 12px !important;  
  
}
```

App.css

Esta classe terá como alvo todos os labels dos campos do formulários e aplicará a eles a cor whitesmoke. Ela também mudará a fonte e aumentará o tamanho da fonte.

Em nosso `className main`, adicione a propriedade `flex-direction`. Essa propriedade definirá a orientação como `column`, de modo que cada elemento na `className main` seja alinhado na horizontal.

```
.main{
```



```
display: flex;
```

```
justify-content: center;
```

```
align-items: center;
```

```
height: 100vh;
```

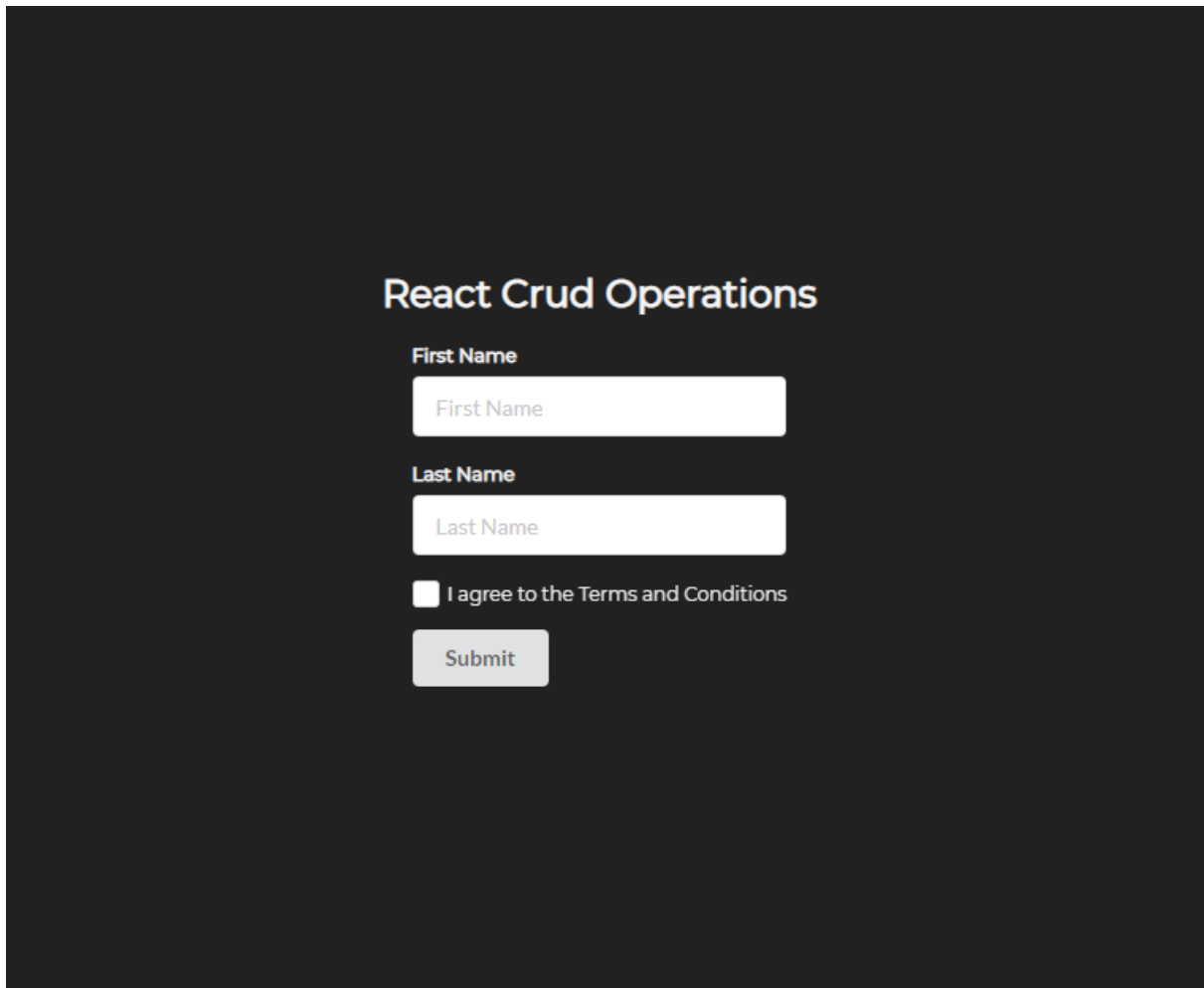
```
background-color: #212121;
```

```
color: whitesmoke;
```

```
flex-direction: column;
```

```
}
```

App.css



React Crud Operations

First Name

Last Name

☐ I agree to the Terms and Conditions

Submit

O formulário, agora, tem uma aparência bem melhor.

Em seguida, vamos obter os dados dos campos do formulário e colocá-los em nosso console. Para isso, usaremos o hook `useState` em React.

No arquivo `Create.js`, importe `useState` de React.

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

Depois, crie states para *first name*, *last name* (nome e sobrenome, respectivamente) e para a caixa de seleção. Inicializaremos os states como vazios ou *false*.

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

```
import { Button, Checkbox, Form } from  
'semantic-ui-react'
```

```
export default function Create() {
```

```
    const [firstName, setFirstName] =  
    useState('');
```

```
    const [lastName, setLastName] =  
    useState('');
```

```
    const [checkbox, setCheckbox] =  
    useState(false);
```

```
return (  
  
  <div>  
  
    <Form className="create-form">  
  
      <Form.Field>  
  
        <label>First Name</label>  
  
        <input placeholder='First  
Name' />  
  
      </Form.Field>  
  
      <Form.Field>  
  
        <label>Last Name</label>  
  
        <input placeholder='Last  
Name' />  

```

```
</Form.Field>
```

```
<Form.Field>
```

```
    <Checkbox label='I agree  
to the Terms and Conditions' />
```

```
</Form.Field>
```

```
    <Button  
type='submit'>Submit</Button>
```

```
</Form>
```

```
</div>
```

```
)
```

```
}
```

Você poderá ver agora que o código está agindo como um componente funcional. Por isso, precisamos transformar o componente em um componente funcional . Somente poderemos usar hooks com esse tipo de componente.

Vamos configurar *first name*, *last name* e a caixa de seleção usando as propriedades `setFirstName`, `setLastName` e `setCheckbox`, respectivamente.

```
<input placeholder='First Name' onChange={{(e)
=> setFirstName(e.target.value)}}/>
```

```
<input placeholder='Last Name' onChange={{(e)
=> setLastName(e.target.value)}}/>
```

```
<Checkbox label='I agree to the Terms and
Conditions' onChange={{(e) =>
setCheckbox(!checkbox)}}/>
```

Agora, estamos capturando os *states* de *first name*, *last name* e da caixa de seleção.

Crie uma função chamada `postData`, que usaremos para enviar dados para a API. Dentro da função, escreveremos este código:

```
const postData = () => {  
  
    console.log(firstName);  
  
    console.log.lastName);  
  
    console.log.checkbox);  
  
}
```

Estamos imprimindo no console o conteúdo de *first name*, *last name* e da caixa de seleção.

No botão Submit, atribua essa função usando o evento `onClick` para que, quando o botão Submit for pressionado, essa função seja chamada.

```
<Button onClick={postData}  
type='submit'>Submit</Button>
```

Aqui temos o código completo para o arquivo *create.js*:

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

```
import { Button, Checkbox, Form } from  
'semantic-ui-react'
```

```
export default function Create() {
```

```
    const [firstName, setFirstName] =  
    useState('');
```



```
    const [lastName, setLastName] =  
    useState('');
```

```
    const [checkbox, setCheckbox] =  
    useState(false);
```

```
    const postData = () => {
```

```
        console.log(firstName);
```

```
        console.log(lastName);
```

```
        console.log(checkbox);
```

```
    }
```

```
    return (
```

```
        <div>
```

```
            <Form className="create-form">
```

```
<Form.Field>
```

```
  <label>First Name</label>
```

```
  <input placeholder='First  
Name' onChange={(e) =>  
setFirstName(e.target.value)}/>
```

```
</Form.Field>
```

```
<Form.Field>
```

```
  <label>Last Name</label>
```

```
  <input placeholder='Last  
Name' onChange={(e) =>  
setLastName(e.target.value)}/>
```

```
</Form.Field>
```

```
<Form.Field>
```

```
        <Checkbox label='I agree  
to the Terms and Conditions' onChange={(e) =>  
setChecked(!checkbox)}/>
```

```
    </Form.Field>
```

```
        <Button onClick={postData}  
type='submit'>Submit</Button>
```

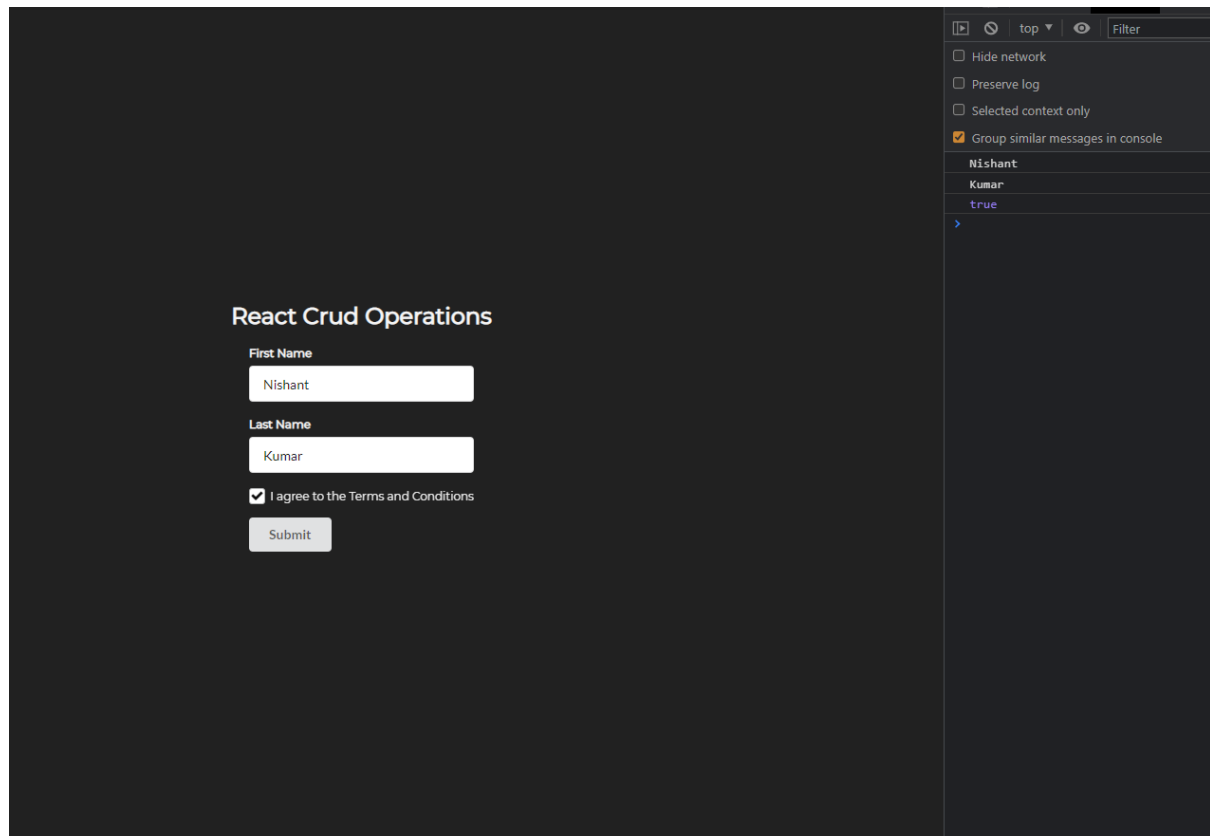
```
    </Form>
```

```
</div>
```

```
)
```

```
}
```

Digite algum valor em *first name* e *last name* e marque a caixa de seleção. Em seguida, clique no botão Submit. Você verá os dados aparecerem no console, assim:

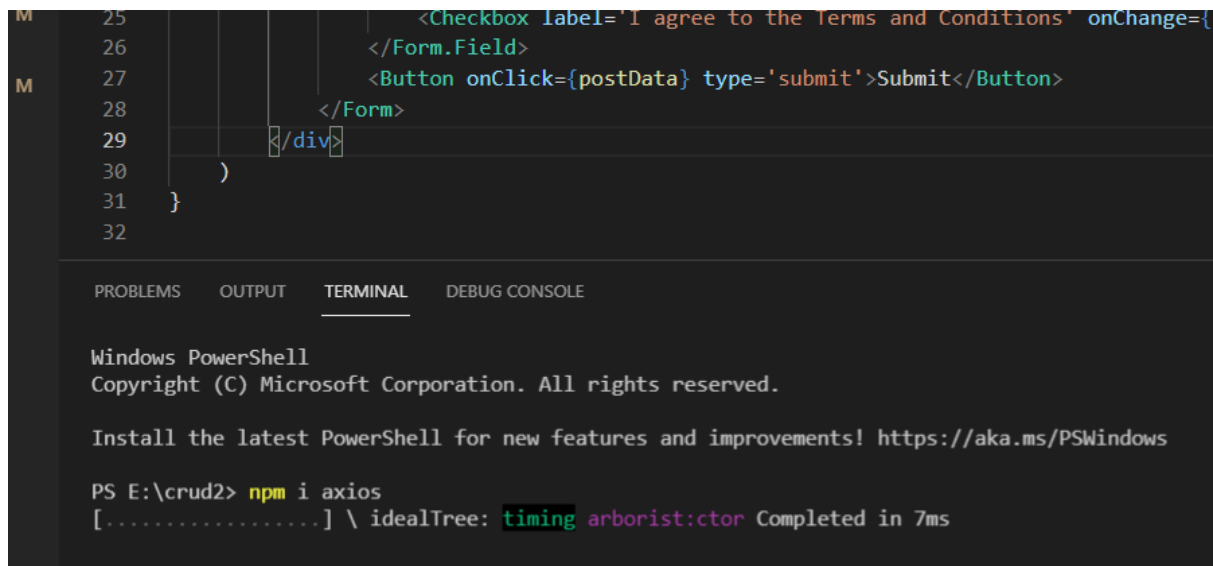


Como usar o Axios para enviar solicitações de API para as Mock APIs

Vamos usar o Axios para enviar os dados do formulário para o servidor mock.

Primeiro, porém, precisamos instalá-lo.

Apenas digite `npm i axios` para instalar o pacote.



The screenshot shows a code editor with a dark theme. The top part displays a JSX element for a form. It includes a checkbox with the label 'I agree to the Terms and Conditions' and an 'onChange' event handler. Below the checkbox is a 'Submit' button with an 'onClick' event handler that calls 'postData'. The form is wrapped in a 'Form.Field' component and a 'Form' component. The bottom part of the screenshot shows a terminal window with the following text:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS E:\crud2> npm i axios
[.....] \ idealTree: timing arborist:ctor Completed in 7ms
```

Após termos instalado o pacote, vamos fazer a operação de criação.

Importe o Axios na parte superior do arquivo.

```
import axios from 'axios';
```

Importação do Axios

Na função `postData`, usaremos o Axios para enviar a solicitação de POST.

```
const postData = () => {
```

```
axios.post(`https://60fbca4591156a0017b4c8a7.
mockapi.io/fakeData`, {

    firstName,

    lastName,

    checkbox

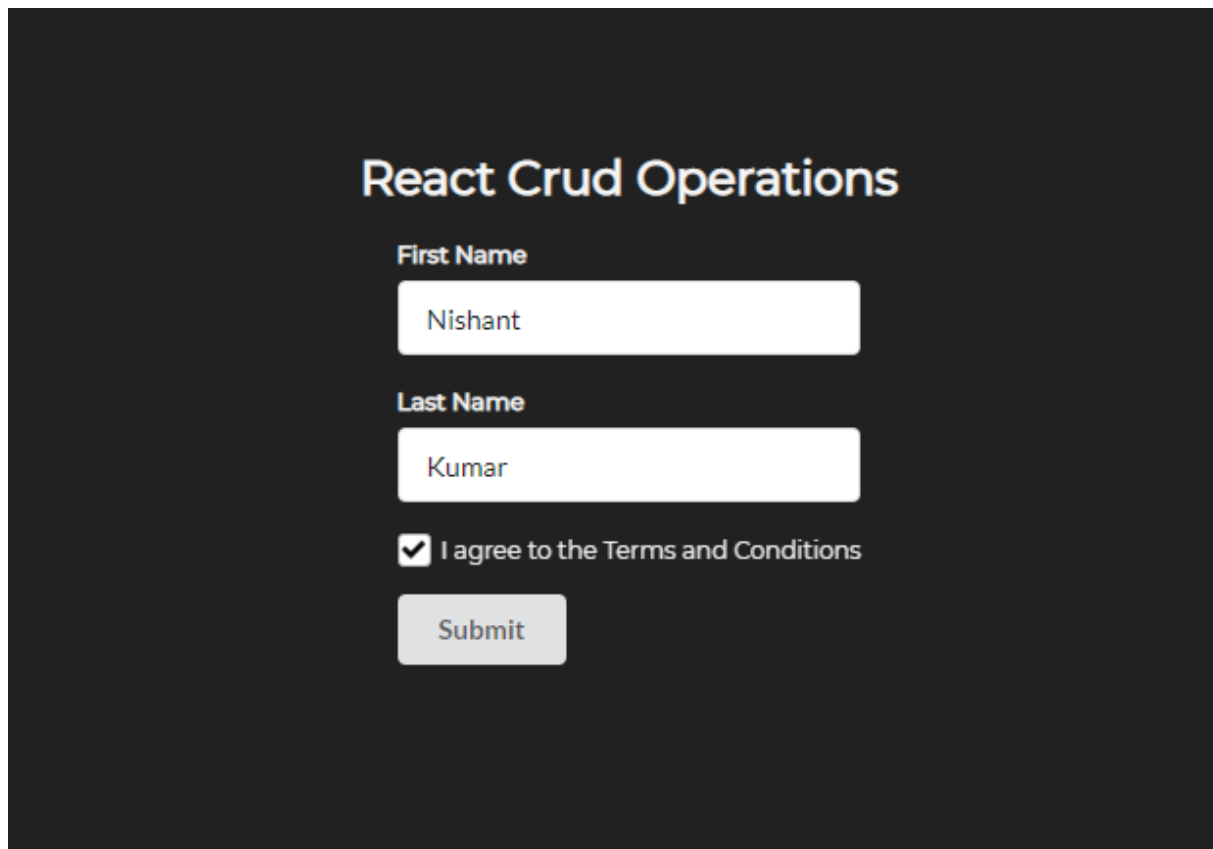
})

}
```

Enviando a solicitação de Post

Como você pode ver, estamos usando `axios.post`. Dentro de `axios.post`, temos o endpoint da API, a qual criamos anteriormente. Em seguida, temos os campos do formulário envolvidos em chaves.

Ao clicarmos em Submit, essa função será chamada e enviará os dados ao servidor da API.



React Crud Operations

First Name

Nishant

Last Name

Kumar

☒ I agree to the Terms and Conditions

Submit

Insira seu nome, sobrenome e marque a caixa de seleção.
Clique em Submit.

```
[{"id":"1","firstName":"Nishant","lastName":"Kumar","checkbox":true}]
```

Se você verificar a API, verá seu nome, sobrenome e a caixa de seleção (ou *checkbox*, em inglês) assinalada como *true*, dentro do objeto.

Como implementar as operações de leitura e atualização

Para iniciar a operação de leitura (Read), precisamos criar uma página de leitura. Também precisaremos do pacote React Router para navegar entre páginas diferentes.

Acesse <https://reactrouter.com/web/guides/quick-start> e instale o pacote usando `npm i react-router-dom`.

Depois de ele ter sido instalado, importamos algumas coisas do React Router:

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

Importando Browser Router como Router e Route do pacote React Router

Em nosso App.js, envolvemos todo o *return* em um Router (roteador). O que isso significa, basicamente, é que tudo o que estiver dentro desse Router poderá usar roteamento em React.

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

```
import Create from './components/create';
```



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

```
function App() {
```

```
  return (
```

```
    <Router>
```

```
      <div className="main">
```

```
        <h2 className="main-header">React  
Crud Operations</h2>
```

```
        <div>
```

```
          <Create />
```

```
        </div>
```

```
</div>
```

```
</Router>
```

```
);
```

```
}
```

```
export default App;
```

Nosso App.js terá, agora, a aparência que vemos acima.

Substitua o Create dentro do return e adicione o código a seguir:

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

```
import Create from './components/create';
```

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

```
function App() {
```

```
  return (
```

```
    <Router>
```

```
      <div className="main">
```

```
        <h2 className="main-header">React  
Crud Operations</h2>
```

```
      <div>
```

```
        <Route exact path='/create'  
component={Create} />
```

```
    </div>
```

```
</div>
```

```
</Router>
```

```
);
```

```
}
```

```
export default App;
```

Aqui, estamos usando o componente Route como Create. Definimos o caminho de Create como '/create'. Assim, se

acessarmos <http://localhost:3000/create>, veremos a página de criação.

Do mesmo modo, precisamos das rotas para leitura e atualização.

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

```
import Create from './components/create';
```

```
import Read from './components/read';
```

```
import Update from './components/update';
```

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

```
function App() {
```

```
  return (
```

```
<Router>
```

```
  <div className="main">
```

```
    <h2 className="main-header">React  
    Crud Operations</h2>
```

```
    <div>
```

```
      <Route exact path='/create'  
      component={Create} />
```

```
    </div>
```

```
    <div style={{ marginTop: 20 }}>
```

```
      <Route exact path='/read'  
      component={Read} />
```

```
    </div>
```

```
      <Route path='/update'  
component={Update} />
```

```
    </div>
```

```
  </Router>
```

```
);
```

```
}
```

```
export default App;
```

Assim, crie as rotas de leitura e atualização (*read* e *update*) da mesma forma que você vê acima.

Se você acessar <http://localhost:3000/read>, verá o seguinte:



Rota de

leitura (Read)

E no URL <http://localhost:3000/update>, teremos o componente de atualização (Update) assim:



Rota de

atualização (Update)

A operação de leitura

Para a operação de leitura (Read), precisaremos de um componente Table (tabela). Então, vá para a Semantic UI do React e use uma tabela da biblioteca.

```
import React from 'react';

import { Table } from 'semantic-ui-react'

export default function Read() {

  return (

    <div>

      <Table singleLine>

        <Table.Header>

          <Table.Row>

            <Table.HeaderCell>Name</Table.HeaderCell>
```

```
<Table.HeaderCell>Registration  
Date</Table.HeaderCell>
```

```
<Table.HeaderCell>E-mail  
address</Table.HeaderCell>
```

```
<Table.HeaderCell>Premium  
Plan</Table.HeaderCell>
```

```
</Table.Row>
```

```
</Table.Header>
```

```
<Table.Body>
```

```
<Table.Row>
```

<div><Table.Cell>John Lilki</Table.Cell></div>	<div><Table.Cell>September 14, 2013</Table.Cell></div>
<div><Table.Cell>jhlilk22@yahoo.com</Table.Cell></div>	
<div><Table.Cell>No</Table.Cell></div>	
<div></Table.Row></div>	
<div><Table.Row></div>	
<div><Table.Cell>Jamie Harrington</Table.Cell></div>	<div><Table.Cell>January 11, 2014</Table.Cell></div>

<Table.Cell>jamieharingonton@yahoo.com</Table
.Cell>

<Table.Cell>Yes</Table.Cell>

</Table.Row>

<Table.Row>

<Table.Cell>Jill
Lewis</Table.Cell>

<Table.Cell>May 11,
2014</Table.Cell>

<Table.Cell>jilsewris22@yahoo.com</Table.Cell
>

```
<Table.Cell>Yes</Table.Cell>
```

```
</Table.Row>
```

```
</Table.Body>
```

```
</Table>
```

```
</div>
```

```
)
```

```
}
```

Read.js

Aqui, você pode ver uma tabela com alguns dados de teste. Temos apenas uma linha de tabela. Assim, removeremos o resto.

```
import React from 'react';

import { Table } from 'semantic-ui-react'

export default function Read() {

  return (

    <div>

      <Table singleLine>

        <Table.Header>

          <Table.Row>

            <Table.HeaderCell>Name</Table.HeaderCell>
```

```
<Table.HeaderCell>Registration  
Date</Table.HeaderCell>
```

```
<Table.HeaderCell>E-mail  
address</Table.HeaderCell>
```

```
<Table.HeaderCell>Premium  
Plan</Table.HeaderCell>
```

```
</Table.Row>
```

```
</Table.Header>
```

```
<Table.Body>
```

```
<Table.Row>
```

```
<Table.Cell>John
Lilki</Table.Cell>

<Table.Cell>September
14, 2013</Table.Cell>

<Table.Cell>jhlilk22@yahoo.com</Table.Cell>

<Table.Cell>No</Table.Cell>

</Table.Row>

</Table.Body>


</Table>

</div>

)
```


}

Read.js



The screenshot shows a web application with a dark background. At the top, the title 'React Crud Operations' is displayed in a light-colored font. Below the title is a white table with four columns: 'Name', 'Registration Date', 'E-mail address', and 'Premium Plan'. The table contains one row of data: 'John Lilki', 'September 14, 2013', 'jhlilk22@yahoo.com', and 'No'.

Name	Registration Date	E-mail address	Premium Plan
John Lilki	September 14, 2013	jhlilk22@yahoo.com	No

Esse é o resultado da página de leitura (Read). Temos uma tabela com quatro colunas, mas somente precisamos de três.

Remova as colunas adicionais de campo e renomeie os campos assim:

React Crud Operations

First Name	Last Name	Checked
Nishant	Kumar	Yes

Esta é a aparência de nossa página de leitura agora:

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

```
import { Table } from 'semantic-ui-react'
```

```
export default function Read() {
```

```
  return (
```

```
    <div>
```

```
      <Table singleLine>
```

<Table.Header>

<Table.Row>

<Table.HeaderCell>First
Name</Table.HeaderCell>

<Table.HeaderCell>Last
Name</Table.HeaderCell>

<Table.HeaderCell>Checked</Table.HeaderCell>

</Table.Row>

</Table.Header>

<Table.Body>

<Table.Row>

<Table.Cell>Nishant</Table.Cell>

<Table.Cell>Kumar</Table.Cell>

<Table.Cell>Yes</Table.Cell>

</Table.Row>

</Table.Body>

</Table>

</div>

```
)  
  
}
```

Read.js

Em seguida, vamos enviar uma solicitação de GET para obter os dados da API.

Precisamos dos dados quando nossa aplicação carregar. Assim, vamos usar o hook `useEffect`.

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

```
useEffect(() => {
```

```
}, [])
```

Hook `useEffect`

Crie um *state* que conterà os dados recebidos. Este state será um array.

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

```
const [APIData, setAPIData] = useState([]);
```

```
useEffect(() => {
```

```
}, [])
```

State de `APIData` para armazenar os dados recebidos da API

No hook `useEffect`, vamos enviar a solicitação de GET.

```
useEffect(() => {
```

```
axios.get(`https://60fbca4591156a0017b4c8a7.m  
ockapi.io/fakeData`)  
  
  .then((response) => {  
  
    setAPIData(response.data);  
  
  })  
  
}, [])
```

Assim, usaremos `axios.get` para enviar a solicitação de GET à API. Então, se a solicitação for preenchida, definiremos os dados da resposta em nosso *state* da *APIData*.

Agora, vamos mapear nossas linhas da tabela de acordo com os dados da API.

Vamos usar a função `Map` para fazer isso. A função vai percorrer o array e exibir os dados no resultados.

```
<Table.Body>
```

```
  {APIData.map((data) => {
```

```
    return (
```

```
      <Table.Row>
```

```
        <Table.Cell>{data.firstName}</Table.Cell>
```

```
        <Table.Cell>{data.lastName}</Table.Cell>
```

```
          <Table.Cell>{data.checkbox ?
```

```
            'Checked' : 'Unchecked'}</Table.Cell>
```

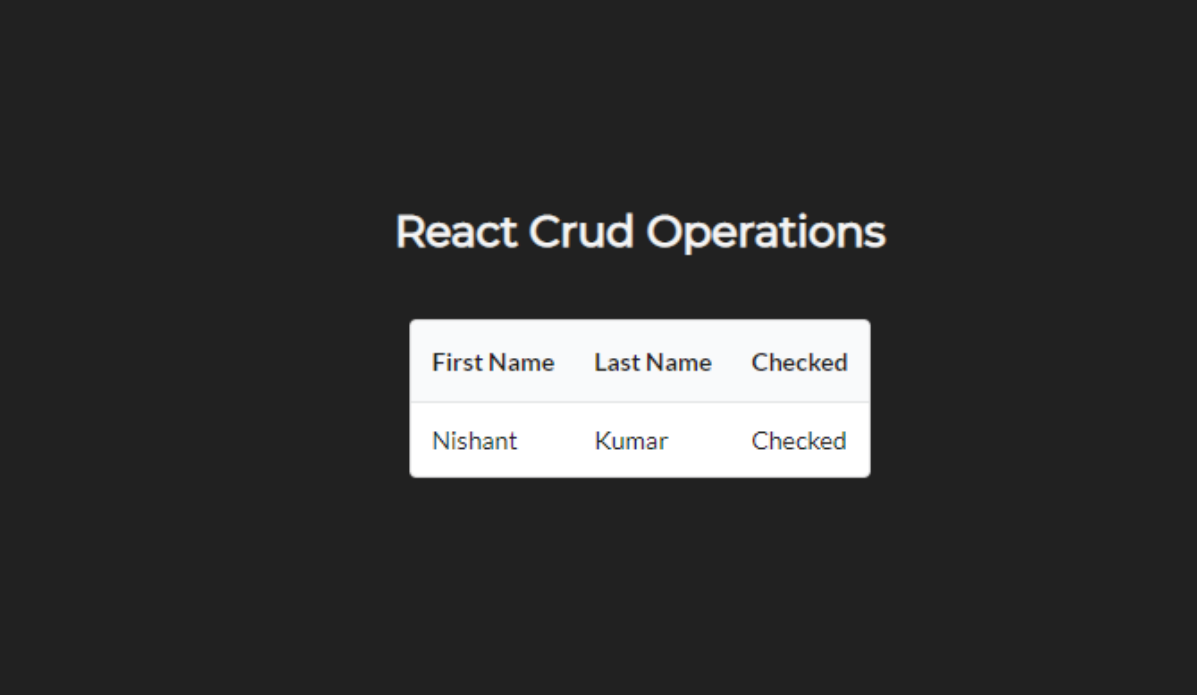
```
        </Table.Row>
```

```
      )})}
```



```
</Table.Body>
```

Estamos mapeando *firstName*, *lastName* e a caixa de seleção de acordo com os dados na API. Nossa caixa de seleção, no entanto, é um pouco diferente. Usei aqui um operador ternário ('?'). Se *data.checkbox* for *true*, o resultado será *Checked* (marcado). Do contrário, será *Unchecked* (desmarcado).



The screenshot shows a web application with a dark background. At the top, the title 'React Crud Operations' is displayed in a light green font. Below the title is a white table with three columns: 'First Name', 'Last Name', and 'Checked'. The table contains one row of data with the values 'Nishant', 'Kumar', and 'Checked'.

First Name	Last Name	Checked
Nishant	Kumar	Checked

Resultado de Read.js

A operação de atualização

Crie mais um cabeçalho para a atualização (Update) e uma coluna para cada linha da tabela para um botão de atualização. Use o botão de Semantic UI do React.

```
<Table.HeaderCell>Update</Table.HeaderCell>
```

```
<Table.Cell>
```

```
  <Button>Update</Button>
```

```
</Table.Cell>
```

Criação do botão Update

Agora, ao clicarmos nesse botão, deveremos ser redirecionados para a página de atualização. Para isso, precisamos do Link de React Router.

Importe Link de React Router. Envolve a célula da tabela para o botão Update com as tags para Link.

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

```
<Link to='/update'>
```

```
  <Table.Cell>
```

```
    <Button>Update</Button>
```

```
  </Table.Cell>
```

```
</Link>
```

Link para o botão Update

Desse modo, se clicarmos no botão Update, seremos redirecionados para a página de atualização.

Para atualizar os dados da coluna, precisaremos dos ID respectivos, os quais obteremos na API.

Crie uma função chamada `setData`. Vincule essa função ao botão Update.

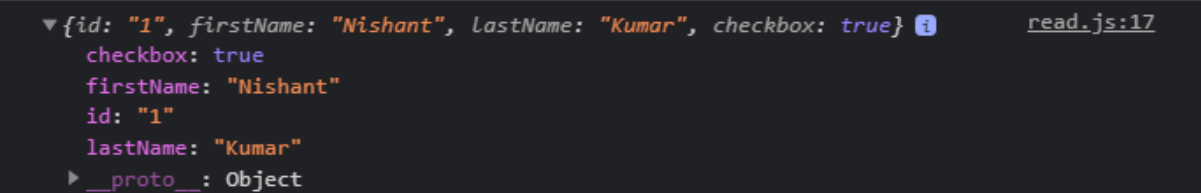
```
<Button onClick={() => setData()}>Update</Button>
```

Depois disso, precisamos passar os dados como parâmetro para a função superior.

```
<Button onClick={() => setData(data)}>Update</Button>
```

Na função acima, registramos esses dados no console:

```
const setData = (data) => {  
  
  console.log(data);  
  
}
```

A screenshot of a browser's developer console. It shows a log entry for a JavaScript object. The object has the following properties: 'id' with value '1', 'firstName' with value 'Nishant', 'lastName' with value 'Kumar', and 'checkbox' with value 'true'. The log entry is expanded, showing the object's structure. The file name 'read.js' and line number '17' are visible in the top right corner of the console area.

```
▼ {id: "1", firstName: "Nishant", lastName: "Kumar", checkbox: true} read.js:17  
  checkbox: true  
  firstName: "Nishant"  
  id: "1"  
  lastName: "Kumar"  
  ► __proto__: Object
```

Dados no console

Clique no botão Update na tabela e confira o console. Você obterá os dados do campo da tabela respectivo.

Vamos definir esses dados no localStorage.

```
const setData = (data) => {  
  
    let { id, firstName, lastName, checkbox } = data;  
  
    localStorage.setItem('ID', id);  
  
    localStorage.setItem('First Name', firstName);  
  
    localStorage.setItem('Last Name', lastName);  
  
    localStorage.setItem('Checkbox  
Value', checkbox)  
  
}
```

Definindo os dados no localStorage (Armazenamento local)

Estamos fazendo a desestruturação dos nossos dados em *id*, *firstName*, *lastName*, e *checkbox*. Em seguida, definimos esses dados no armazenamento local. Você pode usar o armazenamento local para armazenar os dados localmente no navegador.

Em seguida, no componente Update, precisamos de um formulário para a operação de atualização. Vamos reutilizar o formulário do componente Create. Apenas mude o nome da função de Create para Update.

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

```
import { Button, Checkbox, Form } from 'semantic-ui-react'
```

```
import axios from 'axios';
```

```
export default function Update() {
```

```
    const [firstName, setFirstName] = useState('');
```

```
    const [lastName, setLastName] = useState('');
```

```
    const [checkbox, setCheckbox] = useState(false);
```

```
    return (
```

```
<div>

  <Form className="create-form">

    <Form.Field>

      <label>First Name</label>

      <input placeholder='First Name' onChange={{(e)
=> setFirstName(e.target.value)}}/>

    </Form.Field>

    <Form.Field>

      <label>Last Name</label>

      <input placeholder='Last Name' onChange={{(e)
=> setLastName(e.target.value)}}/>

    </Form.Field>

    <Form.Field>

      <Checkbox label='I agree to the Terms and
Conditions' onChange={{(e) => setCheckbox(!checkbox)}}/>
```

```
        </Form.Field>

        <Button type='submit'>Update</Button>

    </Form>

</div>

)

}
```

Crie um hook `useEffect` no componente Update. Nós o utilizaremos para obter os dados que armazenamos anteriormente em `localStorage`. Além disso, crie mais um *state* para o campo *ID*.

```
const [id, setID] = useState(null);

useEffect(() => {

    setID(localStorage.getItem('ID'))

    setFirstName(localStorage.getItem('First Name'));
```



```
        setLastName(localStorage.getItem('Last Name'));

        setCheckbox(localStorage.getItem('Checkbox Value'))

    }, []);
```

Defina os dados respectivos de acordo com suas chaves no armazenamento local. Precisamos definir esses valores nos campos do formulário. Isso preencherá os campos automaticamente quando a página de Update for carregada.

```
<Form className="create-form">

    <Form.Field>

        <label>First Name</label>

        <input placeholder='First Name'
value={firstName} onChange={(e) =>
setFirstName(e.target.value)}/>

    </Form.Field>

    <Form.Field>

        <label>Last Name</label>
```

```
        <input placeholder='Last Name'
value={lastName} onChange={(e) => setLastName(e.target.value)}/>

    </Form.Field>

    <Form.Field>

        <Checkbox label='I agree to the Terms and
Conditions' checked={checkbox} onChange={(e) =>
setChecked(!checkbox)}/>

    </Form.Field>

    <Button type='submit'>Update</Button>

</Form>
```

Axios DELETE Requests

O Axios possui uma `axios.delete()` função que facilita o envio de uma solicitação HTTP DELETE para uma determinada URL.

```
const res = await axios.delete('https://httpbin.org/delete');

res.status; // 200
```

Ao contrário de `axios.post()` e `axios.put()`, o segundo parâmetro `axios.delete()` são as **opções do Axios**, **não** o corpo da solicitação. Para enviar um corpo de solicitação com uma solicitação DELETE, você deve usar a **data opção**.

```
const res = await axios.delete('https://httpbin.org/delete', { data: {  
  answer: 42 } });  
  
res.data.json; // { answer: 42 }
```

Usaremos o `axios.delete` para excluir as colunas respectivas.

```
const onDelete = (id) => {
```

```
  axios.delete(`https://60fbca4591156a0017b4c8a7.mockapi.io/fakeData/${id}`)  
  
}
```

Clique no botão Delete e confira a API. Você verá que os dados foram excluídos.

Precisamos carregar os dados da tabela após eles terem sido excluídos.

Desse modo, crie uma função para carregar os dados da API.

```
const getData = () => {
```

```
  axios.get('https://60fbca4591156a0017b4c8a7.mockapi.io/fakeData')
```

```
.then((getData) => {  
  
    setAPIData(getData.data);  
  
}))  
  
}
```

Obtendo os dados da API

Agora, na função `onDelete`, precisamos carregar os dados atualizados após excluirmos um campo.

```
const onDelete = (id) => {  
  
    axios.delete(`https://60fbca4591156a0017b4c8a7.mockapi.io/fakeData/${id}`)  
  
    .then(() => {  
  
        getData();  
  
    })  
  
}
```

