3 DE DEZEMBRO DE 2021 / #REACT

Como criar um app em React com o back-end em Node: O guia completo

Tradutor: Daniel Rosa

Autor: Reed Barger (em inglês)

Um front-end em React conectado com um back-end em Node é uma combinação sólida para qualquer aplicação que você queira criar.

Este guia foi projetado para ajudá-lo a criar projetos full-stack com React da maneira mais fácil possível.

Vamos ver como configurar um projeto inteiro usando React e Node do zero e lançar o projeto na web.

> Quer criar e lançar aplicativos em React e Node por conta própria? Confira a série de cursos do autor, que mostra como criar seus próprios projetos full-stack em React, como este aqui.

Ferramentas necessárias

1. Verifique se o Node e o NPM estão instalados no seu computador. Você pode fazer o download dos dois em nodejs.org (o NPM vem incluído na instalação do Node)
2. Use um editor de código de sua preferência. Eu utilizo e recomendo pessoalmente o uso do VSCode. Você pode fazer o download do VSCode em code.visualstudio.com.
3. Verifique se tem o Git instalado no seu computador. Isso é necessário para fazer o deploy de sua aplicação no Heroku. Você consegue fazer o download do Git em git-scm.com
4. Faça uma conta em heroku.com. Usaremos o Heroku para publicar nosso app na web totalmente de graça.

Primeiro, crie uma pasta para o seu projeto chamada `react-node-app` (por exemplo).

Em seguida, arraste essa pasta para o seu editor de código.

Para criar nosso projeto em Node, execute o seguinte comando no terminal:

```
npm init -y
```

Isso criará um arquivo package.json, que permitirá acompanhar todos os scripts do nosso app e gerenciar as dependências das quais nosso app do Node necessita.

Nosso código do servidor ficara em uma pasta chamada `server`. Vamos criar essa pasta.

Nela, colocaremos um único arquivo, a partir do qual rodaremos o servidor: `index.js`.

Usaremos o Express para criar um servidor simples na web, o qual rodará na porta 3001 se nenhum valor tiver sido dado para a variável de ambiente `PORT` (o Heroku definirá o valor dessa variável ao fazer o deploy do seu app).

```
// server/index.js

const express = require("express");

const PORT = process.env.PORT || 3001;
```

```
});
```

Em seguida, no terminal, instalaremos o Express como dependência para utilizá-lo:

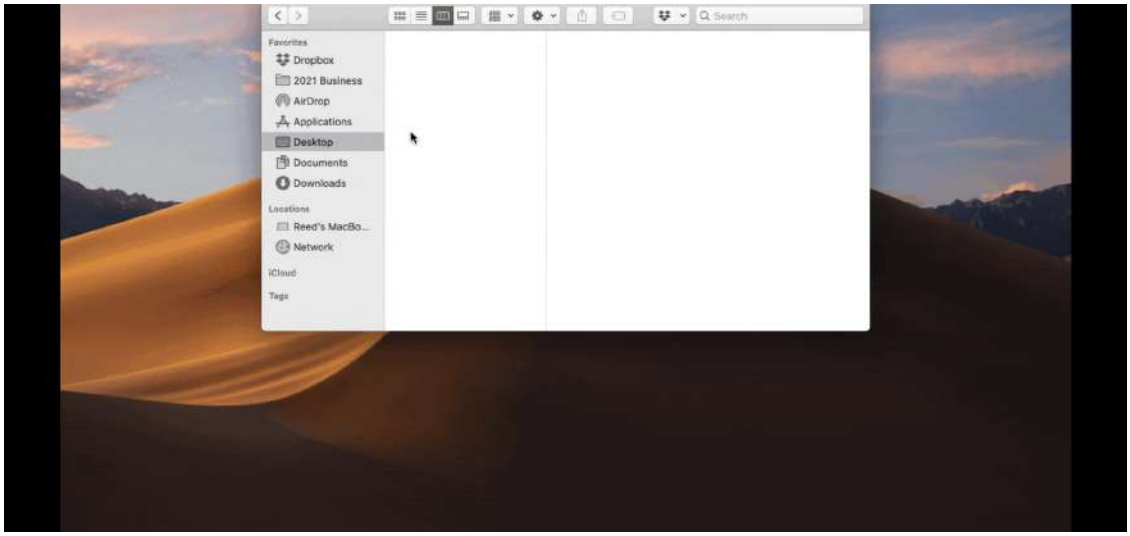
```
npm i express
```

Depois disso, criaremos um script no package.json que iniciará nosso servidor da web ao executá-lo com `npm start`:

```
// server/package.json

...
"scripts": {
  "start": "node server/index.js"
},
...
```

Por fim, podemos rodar nosso app usando esse script escrevendo npm start em nosso terminal e clicando em Enter. Veremos que ele está em execução na porta 3001:

```
npm start

> node server/index.js

Server listening on 3001
```



Passo 2: Criar um endpoint de API

Queremos usar o servidor em Node e Express como uma API, para fornecer dados ao nosso app em React, alterar esses dados ou fazer algumas operações que somente nosso servidor pode fazer.

Em nosso caso, simplesmente enviaremos ao nosso app em React uma mensagem que diz "Hello from server!" (Olá vindo do servidor!) em um objeto JSON.

O código abaixo cria um endpoint para a rota `/api`.

Se nosso app em React fizer uma solicitação GET para aquela rota, responderemos (usando `res`, que significa resposta) com nossos dados em JSON:

```
// server/index.js
...
```

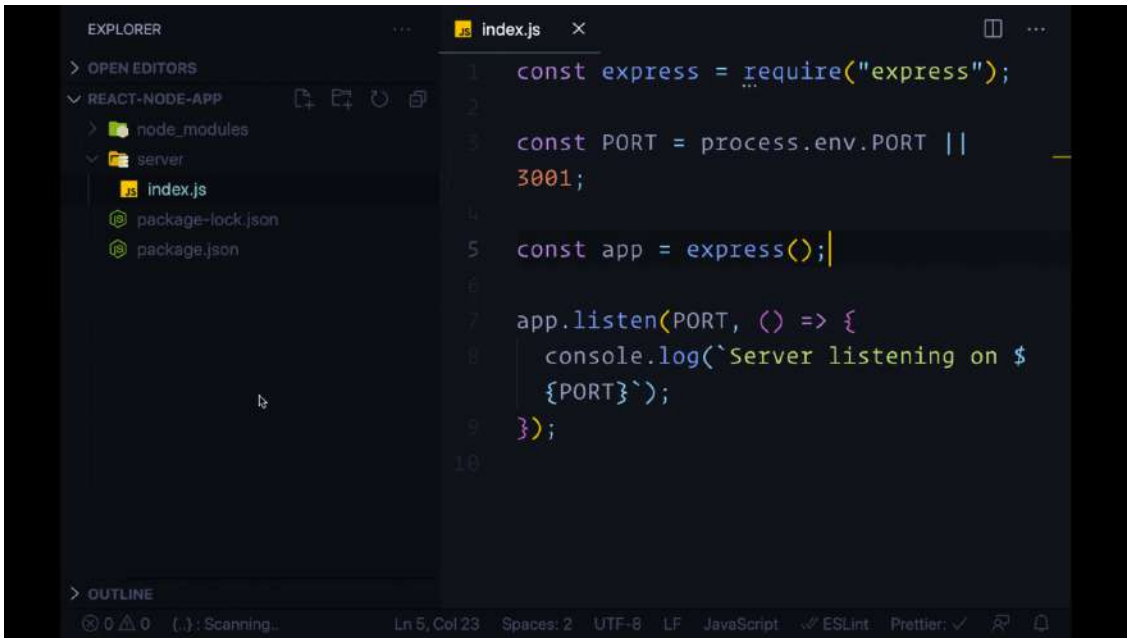
```
app.listen(PORT, () => {
  console.log(`Server listening on ${PORT}`);
});
```

Observação: Não se esqueça de colocar o bloco de `app.get` acima da função `app.listen`.

Como fizemos alterações em nosso código do Node, precisamos reiniciar nosso servidor.

Para fazer isso, encerre o script de inicialização no terminal pressionando Command/Ctrl + C. Em seguida, reinicie-o usando `npm start` novamente.

Para fazer um teste, simplesmente visitamos `http://localhost:3001/api` no navegador e vemos nossa mensagem:



Depois de criar nosso back-end, passemos para o front-end.

Abra outra guia do terminal e use create-react-app para criar um novo projeto do React com o nome `client`:

```
npx create-react-app client
```

Depois disso, teremos um app do React com todas as suas dependências instaladas.

A única mudança que teremos de fazer é adicionar uma propriedade chamada `proxy` ao nosso arquivo package.json.

Isso nos permitirá fazer solicitações ao nosso servidor em Node sem ter de fornecer a origem de onde ele está em execução (http://localhost:3001) toda vez que fizermos uma solicitação de rede para ele:

```
// client/package.json

...
"proxy": "http://localhost:3001",
...
```

Em seguida, podemos iniciar nosso app do React rodando seu script de inicialização, que é o mesmo do nosso servidor em Node. Primeiro, certifique-se de usar `cd` para estar dentro da nossa pasta do client recém-criada.

```
cd client
npm start

Compiled successfully!

You can now view client in the browser.

Local:            http://localhost:3000
```

Passo 4: Fazer solicitações HTTP do React para o Node

Agora que temos um app em React funcionando, queremos usá-lo para interagir com nossa API.

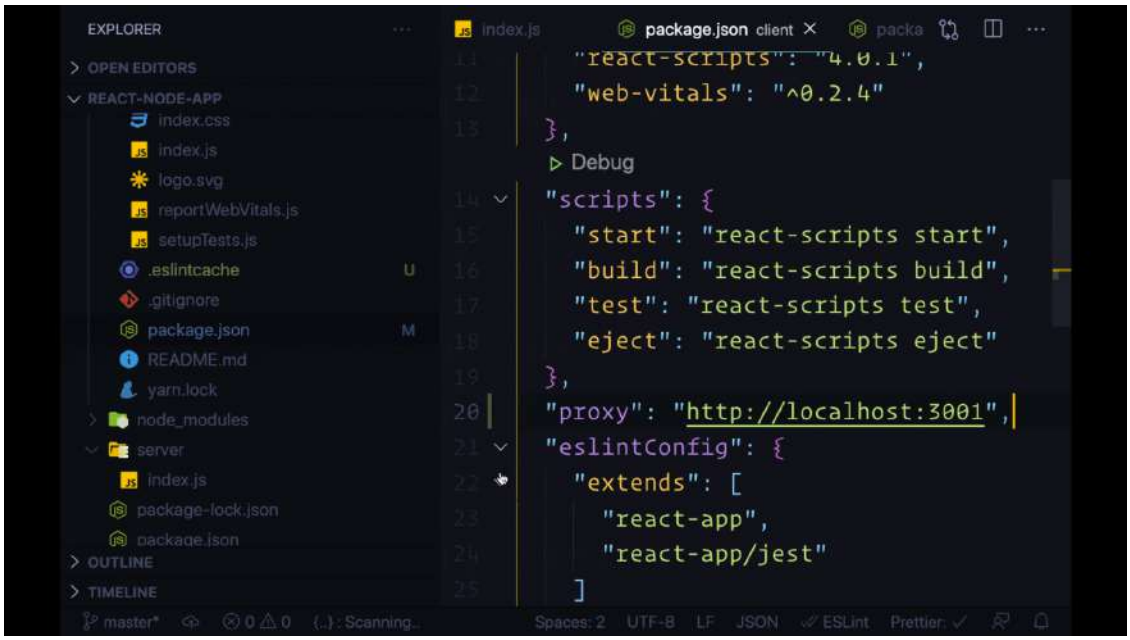
Vamos ver como obter dados do endpoint da `/api` que criamos anteriormente.

Faremos uma solicitação GET simples para o nosso back-end usando a Fetch API e, em seguida, teremos nossos dados retornados como JSON.

Quando os dados tiverem sido retornados para nós, pegaremos a propriedade message (para obter a saudação que enviamos do servidor) e a colocaremos em uma variável de estado (state) chamada `data`.

Isso nos permitirá exibir aquela mensagem em nossa página se a tivermos. Usaremos um condicional em nosso JSX para dizer que, se nossos dados ainda não tiverem sido obtidos, mostraremos o texto "Loading..." (Carregando...).

```
// client/src/App.js

import React from "react";
import logo from "./logo.svg";
import "./App.css";

function App() {
  const [data, setData] = React.useState(null);

  React.useEffect(() => {
    fetch("/api")
      .then((res) => res.json())
      .then((data) => setData(data.message));
  }, []);

  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>{!data ? "Loading..." : data}</p>
      </header>
    </div>
  );
```



Passo 5: Colocar o app na web com o Heroku

Por fim, vamos fazer o deploy da nossa aplicação na web.

Primeiro, dentro da pasta client, verifique se removemos o repositório Git que é inicializado automaticamente pelo create-react-app.

Isso é essencial para fazer o deploy do app, pois vamos configurar um repositório do Git na pasta raiz do nosso projeto (`react-node-app`), em vez de no `client` :

```
cd client
rm -rf .git
```

meusuperapp.herokuapp.com).

Veremos como nossas solicitações estão sendo tratadas por nossa API em Node. Para isso, precisamos escrever um código que exibirá nosso app em React quando ele for solicitado por nosso usuário (por exemplo, quando vamos para a página inicial do nosso app).

Podemos fazer isso em `server/index.js` adicionando o código a seguir:

```
// server/index.js
const path = require('path');
const express = require('express');

...

// Fazer com que o Node sirva os arquivos do app em React criado
app.use(express.static(path.resolve(__dirname, '../client/build')));

// Lidar com as solicitações GET feitas à rota /api
app.get("/api", (req, res) => {
  res.json({ message: "Hello from server!" });
});

// Todas as outras solicitações GET não tratadas retornarão nosso app e
app.get('*', (req, res) => {
  res.sendFile(path.resolve(__dirname, '../client/build', 'index.html'
});
```

Este código permitirá que o Node acesse nosso projeto criado em React usando a função `express.static` para arquivos estáticos.

Se uma solicitação GET vier e não for tratada pela rota `/api`, nosso servidor responderá com nosso app em React.

Em seguida, podemos informar nosso app em Node como fazer isso adicionando um script de `build` ao nosso arquivo package.json do servidor, que cria nosso app em React para a produção:

```
// server/package.json

...
"scripts": {
    "start": "node server/index.js",
    "build": "cd client && npm install && npm run build"
  },
...
```

Recomendo fornecer um campo chamado "engines" (mecanismos), onde você vai querer especificar a versão do Node que você está usando para criar seu projeto. Esse campo será usado para o deploy.

Você pode saber qual é a sua versão do Node executando o comando `node -v`. Coloque o resultado em "engines" (por exemplo, 14.15.4):

```
// server/package.json

"engines": {
  "node": "sua-versão-do-node"
}
```

Depois disso, estamos prontos para o deploy usando o Heroku. Então, não se esqueça de fazer sua conta no Heroku.com.

Logo após, talvez você queira instalar a CLI do Heroku em seu computador para fazer o deploy do seu app sempre que fizer mudanças usando o Git. Podemos instalar a CLI com o comando (no Linux):

```
sudo npm i -g heroku
```

Assim que ela estiver instalada, faça login no Heroku pela CLI usando o comando `heroku login`:

```
heroku login

Press any key to login to Heroku
```

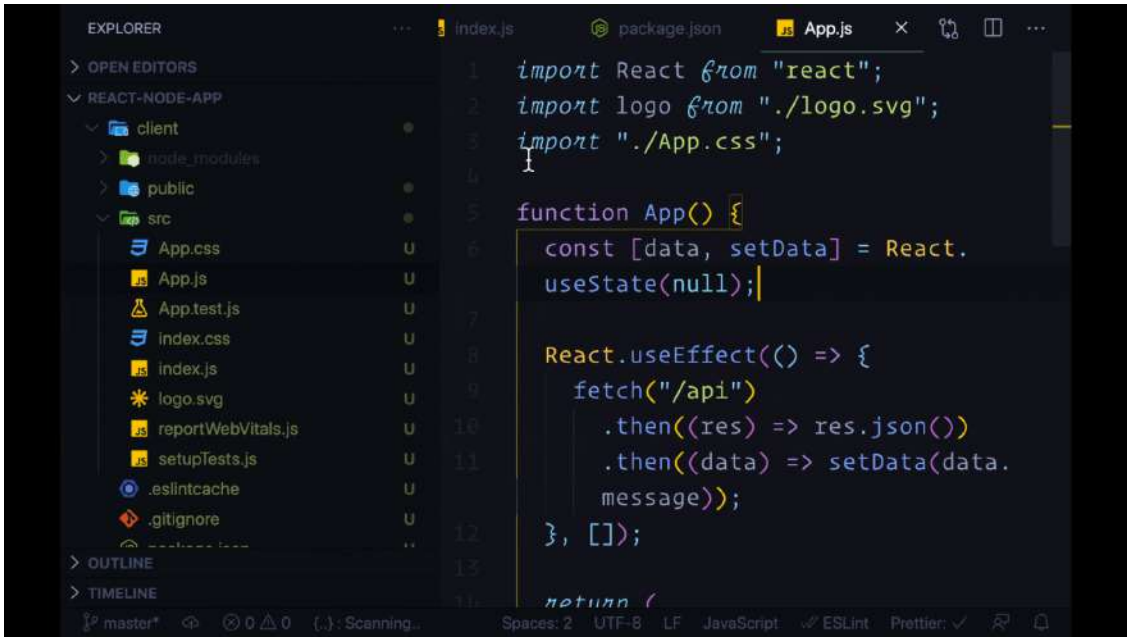
Depois de fazer o login, você só precisa seguir as instruções de deploy para o nosso app criado na guia "Deploy".

Os quatro comandos a seguir iniciarão um novo repositório do Git para nosso projeto, adicionarão nossos arquivos a ele, farão o commit dos arquivos e adicionarão um repositório remoto do Git para o Heroku.

```
git init
heroku git:remote -a insert-your-app-name-here
git add .
git commit -am "Deploy app to Heroku"
```

```
git push heroku master
```

Parabéns! Seu app em React e Node full-stack já está disponível na web! 🎉



Quando quiser fazer mudanças no seu app daqui para diante (e fazer o deploy dessas mudanças), só precisa usar o Git para adicionar seus arquivos, fazer o commit deles e fazer o push dos arquivos para o repositório remoto no Heroku:

```
git add .
git commit -m "my commit message"
git push heroku master
```

com React? Saiba como.

Ao fim de cada mês, o autor lançará um curso exclusivo, mostrando com exatidão como criar um clone de app completo com o React, do início ao fim.

Quer ser notificado sobre a chegada do próximo curso? **Entre na lista de espera aqui**.

Tradutor: Daniel Rosa

Um profissional dos idiomas humanos apaixonado por linguagens de computador. | A world languages professional in love with computer languages.

Autor: Reed Barger (em inglês)

Full stack developer sharing everything I know.

Se você leu até aqui, agradeça ao autor para mostrar que você se importa com o trabalho. Agradeça

Aprenda a programar gratuitamente. O plano de estudos em código aberto do freeCodeCamp já ajudou mais de 40.000 pessoas a obter empregos como desenvolvedores. Comece agora

O freeCodeCamp é uma organização beneficente 501(c)(3), isenta de impostos e apoiada por doações (Número de identificação fiscal federal dos Estados Unidos: 82-0779546).

As doações feitas ao freeCodeCamp vão para nossas iniciativas educacionais e ajudam a pagar servidores, serviços e a equipe.

Você pode fazer uma doação dedutível de impostos aqui.

Guias de tendências

Nova aba em HTML
Máscaras de sub-rede
40 projetos em JavaScript
Tutorial de button onClick
Bot do Discord
Centralizar em CSS
Excluir pastas com o cmd
Imagens em CSS
25 projetos em Python
Excluir branches
Date now em JavaScript
Var, let e const em JavaScript
Axios em React
ForEach em JavaScript
Fotos do Instagram
Jogo do dinossauro
Menu iniciar
Arrays vazios em JS
Caracteres especiais
Python para iniciantes
Provedores de e-mail
15 portfólios
Node.js no Ubuntu
10 sites de desafios
Clonar branches
Media queries do CSS
Fix do Live Server no VS Code
SQL em Python
Interpretadas x compiladas
Imagens SVG em HTML e CSS

Nossa instituição

Sobre Rede de ex-alunos Código aberto Loja Apoio Patrocinadores
Honestidade acadêmica Código de conduta Política de privacidade Termos de serviço
Política de direitos de autor