



ELECTRON TOP

Electron - construindo aplicações desktop com JavaScript

por **Thiago Marinho** há um ano

10 MIN DE LEITURA

O ecossistema do JavaScript tem se demonstrado muito versátil ao longo dos anos, com grande inovações, uma delas é a possibilidade de construir aplicativos Desktop utilizando apenas Node.js, HTML, CSS e claro JavaScript.



O que é?

Electron é um shell multiplataforma — uma interface do usuário para acessar serviços do sistema operacional tanto via linha de comando (CLI) e interface gráfica (GUI).

Com Electron podemos desenvolver aplicações desktop usando HTML, CSS e Javascript – As famosas tecnologias da Web. Possui integração com Node.js, podemos usá-lo para construir não apenas as telas mas toda a lógica de um BackEnd que acessa recursos do sistema operacional — diretórios locais, banco de dados de maneira mais simples.

É multiplataforma — podemos instalar a aplicação no Windows, Linux e MacOS.



História

Electron foi criado por Cheng Zhao, e lançado em 11 de abril de 2013 com o nome **Atom Shell**. Logo em 6 de maio de 2014 o editor de códigos Atom do GitHub se tornou *open source* sob a licença MIT. Em 2015 o Atom Shell foi

renomeado para **Electron**, e com passar do tempo foi se popularizando e em apenas dois anos conseguiram criar e disponibilizar as atualizações automáticas dos aplicativos, instaladores Windows, envio de logs de erros dos aplicativos, notificações no desktop, entre outras funcionalidades. Em 2016 a Windows Store já começou aceitar aplicativos Electron em sua loja.

No blog do Electron tem outras informações e curiosidades.

Aplicativos desktop em 2020?

Sim. São vários cases de sucesso e necessidades que possuímos — como programadores usamos o editor de código VsCode que foi feito com Electron, o próprio Atom o primeiro app feito com Electron.

Foi construído também com o Slack, Figma, Twitch, Whatsapp desktop, entre outros apps que você pode encontrar nesse site: <https://www.electronjs.org/apps>

Arquitetura

Electron vem com navegador Chromium, um projeto *open source* de onde surgiu o Google Chrome. Toda a parte visual, janelas, etc são renderizadas nessa camada e o BackEnd executado em Node.js. Ambos tem acesso um ao outro via RPC (*Remote Procedure Call*).

Dois conceitos importantes em volta da tecnologia: processo principal (**main process**) e o processo de renderização (**renderer process**)

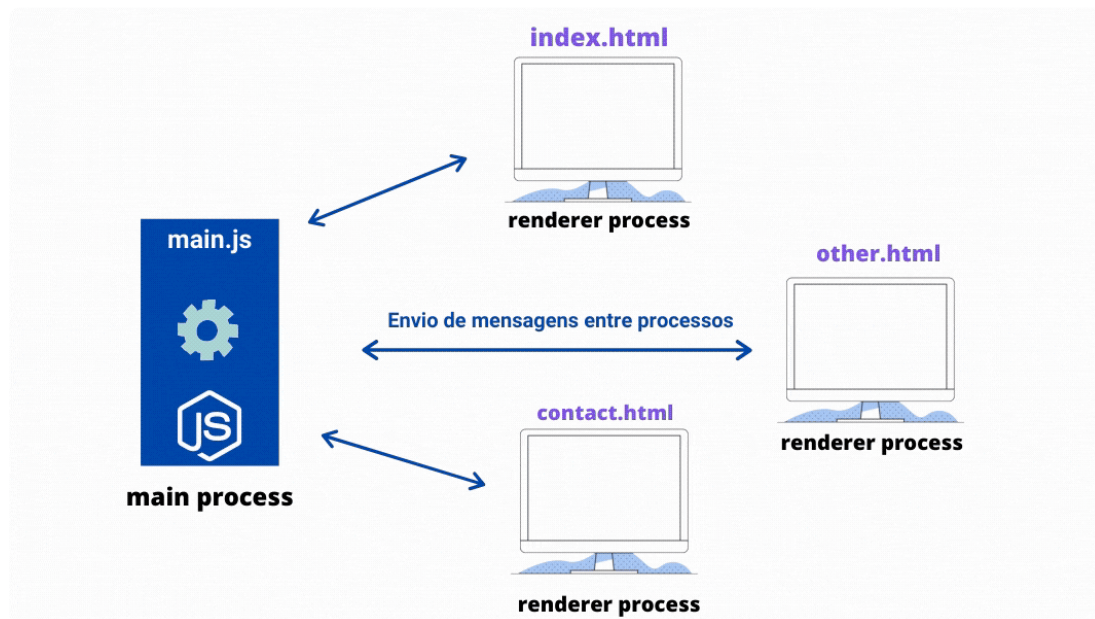
O arquivo que é definido na propriedade `main` do `package.json` é chamado de **processo principal**. Ele é **único** em toda a aplicação, responsável por criar a(s) janela(s) da aplicação através de instâncias de **BrowserWindow**.

Electron usa o Chromium para renderizar as páginas web, a arquitetura de multiprocesso do dessa ferramenta é utilizada também, cada página web (janela desktop do Electron) executa o seu próprio processo, que é chamado de **processo de renderização**.

Resumumindo, temos o Node.js no **processo principal** usando a biblioteca Electron que faz chamadas nativas no sistema operacional. E a API do Electron executando as janelas da aplicação com HTML, CSS, JavaScript e seus assets do por baixo dos panos com Chromium.

As janelas (FrontEnd) não podem invocar os recursos nativos para não quebrar a segurança da aplicação, porém a comunicação entre ambos processos — principal e renderização, é feita através de um conceito de **comunicação entre processos** (IPC) usando o método RPC (Chamada de Procedimentos Remotos). Para isso temos as APIs ipcRender e ipcMain.

Esse é um **assunto avançado**, podemos criar um post só para isso. Esse conhecimento é necessário quando formos acessar um banco de dados e outros recursos do sistema operacional com requisições feitas a partir das janelas da aplicação (*renderer process*).



Representação do envio de mensagens entre processos

Estrutura do Projeto

Aplicação em Electron, é essencialmente uma aplicação em Node.js, precisa de:

- **package.json** - Aponta para o arquivo principal do app e lista as suas dependências.
- **main.js** - Inicia o app e cria uma janela do navegador para renderizar HTML. Este é o processo principal do app (main process.).
- **index.html** - Uma página da web que é renderizada. Este é o processo de renderização do app (renderer process).

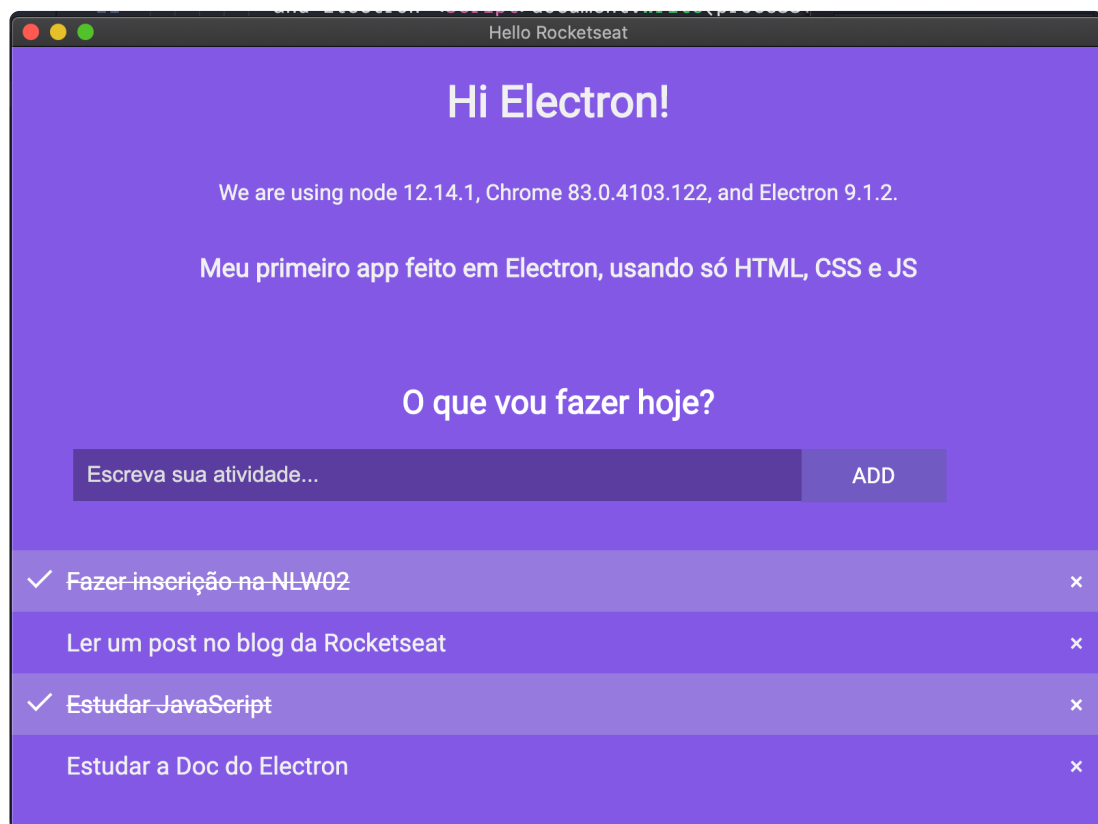


Criando o primeiro projeto

Pré-requisitos: Conhecimento básico em HTML, CSS, JavaScript e Node.js.

Para configurar o ambiente em seu sistema operacional basta seguir a documentação que já está em português.

Vamos criar esse App:



Aqui está o código fonte se precisar consultar durante o passo a passo.

Iniciando o projeto

Crie uma pasta para fazer seu projeto e inicialize-o em Node.js:

```
mkdir hi-electron && npm init -y
```

Abra a pasta `hi-electron` no seu editor de código. Se estiver usando o VSCode: `code .`

Instale a dependência do electron:

```
npm install --save-dev electron
```

Após a instalação, vai ser criada a pasta `node_modules` na raiz do projeto.

No `package.json` vai ter uma propriedade `"main"` você pode alterar de `index.js` para `main.js` esse é o arquivo principal do projeto que vamos criar em seguida. E na propriedade `scripts` altera o `test` para `start` e coloca `electron main.js` para o Electron executar o arquivo `main.js`.

```
{
  "name": "hi-electron",
  "version": "1.0.0",
  "description": "Meu primeiro app em Electron",
  "main": "main.js",
  "scripts": {
    "start": "electron main.js"
  },
  "keywords": [],
  "author": "",
  "license": "MIT",
  "devDependencies": {
    "electron": "^9.1.2"
  }
}
```

```
}  
}
```

O arquivo `main.js` vai conter o código responsável por criar janelas e manipular todos os eventos do sistema operacional.

Vamos criar um arquivo `main.js` e `index.html` na raiz do projeto usando o terminal se preferir:

```
touch main.js index.html
```

Adicionando o Live Reload

Vamos adicionar uma lib para fazer *live reload* da aplicação a cada alteração de código que fizermos, isso ajuda bastante no desenvolvimento.

Sem ela teríamos que parar o processo do node e executar novamente a cada vírgula que altermos no código.

```
npm install electron-reload
```

Logo iremos ver o código `main.js` que irá configurar a aplicação e usar essa lib que acabamos de instalar.

Adicionando um ícone no app

Crie a pasta `build` e adicione a imagem `icon.png` que você pode **baixar aqui**.

O arquivo main.js

Basicamente esse é o primeiro código do app com Electron e deixei comentado cada trecho para você entender. Esse é o arquivo de entrada que será executado quando rodarmos o comando `npm start`.

Copie e cole esse código no arquivo `main.js` que criamos e dê uma lida nos comentários:

```
const { app, BrowserWindow, nativeImage } = require("electron")

// Habilita o live reload no Electron e no FrontEnd da apli
// Assim que alguma alteração no código é feita
require("electron-reload")(__dirname, {
  // Note that the path to electron may vary according to t
  electron: require(`_${__dirname}/node_modules/electron`),
});

// Função que cria uma janela desktop
function createWindow() {
  // Adicionando um ícone na barra de tarefas/dock
  const icon = nativeImage.createFromPath(`${app.getAppPath()}/resources/icon.ico`)

  if (app.dock) {
    app.dock.setIcon(icon);
  }

  // Cria uma janela de desktop
  const win = new BrowserWindow({
    icon,
    width: 800,
    height: 600,
    webPreferences: {
```

```
// habilita a integração do Node.js no FrontEnd
nodeIntegration: true,
},
});

// carrega a janela com o conteúdo dentro de index.html
win.loadFile("index.html");

// Abre o console do navegador (DevTools),
// manter apenas quando estiver desenvolvendo a aplicação
// pode utilizar variáveis de ambiente do node para execu
// win.webContents.openDevTools();
}

// Método vai ser chamado assim que o Electron finalizar su
// e estiver pronto para abrir e manipular o nosso código.
// Algumas APIs podem ser usadas somente depois que este ev
app.whenReady().then(createWindow);

// Quando clicarmos no botão de fechar a janela no app desk
// O evento vai ser ouvido aqui no arquivo main.js e algum
// tipo fechar alguma conexão de banco de dados por exemplo
app.on("window-all-closed", () => {
  // No MacOS quando fecha uma janela, na verdade ela é "mi
  // e o processo executa em segundo-plano tipo um app do c
  // Para fechar e encerrar o app tem que teclar Cmd+Q ou n
  // clicar com botão direito e encerrar o app
  if (process.platform !== "darwin") {
    app.quit();
  }
});

app.on("activate", () => {
  // Esse evento é disparado pelo MacOS quando clica no íco
  // Basicamente cria a janela se não foi criada.
  if (BrowserWindow.getAllWindows().length === 0) {
    createWindow();
  }
});
```

```
}  
});
```

```
// Abaixo você pode colocar seus códigos específicos do Bac  
// pode criar pastas e arquivos separados e importar aqui (
```

O arquivo index.html

No arquivo index.html cole esse código abaixo, nele contém a estrutura HTML, a estilização em CSS e a lógica implementada em JavaScript para manipular o DOM.

Poderia deixar o CSS e o JavaScript separado o que é melhor ainda. [Veja a branch master do projeto.](#)

Fiz alguns comentários no código JavaScript para entender a lógica.

```
<html>  
  <head>  
    <meta charset="UTF-8">  
    <title>Hello Rocketseat</title>  
    <!-- <https://electronjs.org/docs/tutorial/security#content-security-policy>  
    <meta http-equiv="Content-Security-Policy" content="script-src 'self'>  
    <!--Importo o a fonte Roboto do Google Fonts -->  
    <link href="<https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&family=Roboto+Mono:wght@400;700>  
  
  <style>  
    * {  
      box-sizing: border-box;  
    }  
    body {
```

```
margin: 0;
min-width: 250px;
background-color: #8257e5;
color: #f0f0f0;
font-family: "Roboto", sans-serif;
}
.intro {
  display: flex;
  flex-direction: column;
  align-items: center;
}
ul {
  margin: 0;
  padding: 0;
}
/* Style the list items */
ul li {
  cursor: pointer;
  position: relative;
  padding: 12px 8px 12px 40px;
  list-style-type: none;
  background: transparent;
  font-size: 18px;
  transition: 0.2s;
  /* make the list items unselectable */
  -webkit-user-select: none;
  -moz-user-select: none;
  -ms-user-select: none;
  user-select: none;
}
/* Set all odd list items to a different color (z
ul li:nth-child(odd) {
  /* background: #f9f9f9;
  */
}
/* Darker background-color on hover */
ul li:hover {
```

```
background: #7159c1;
}
/* When clicked on, add a background color and st
ul li.checked {
background: rgba(200, 200, 200, 0.3);
color: #fff;
text-decoration: line-through;
}
/* Add a "checked" mark when clicked on */
ul li.checked::before {
content: "";
position: absolute;
border-color: #fff;
border-style: solid;
border-width: 0 2px 2px 0;
top: 10px;
left: 16px;
transform: rotate(45deg);
height: 15px;
width: 7px;
}
/* Style the close button */
.close {
position: absolute;
right: 0;
top: 0;
padding: 12px 16px 12px 16px;
}
.close:hover {
background-color: rgba(130, 702, 205, 0.3);
color: white;
}
/* Style the header */
.card__todo {
margin: 5px;
background-color: transparent;
padding: 30px 40px;
```

```
        color: white;
        text-align: center;
    }
    /* Clear floats after the header */
    .card__todo:after {
        content: "";
        display: table;
        clear: both;
    }
    /* Style the input */
    input {
        margin: 0;
        border: none;
        border-radius: 0;
        width: 75%;
        padding: 10px;
        float: left;
        font-size: 16px;
        background-color: rgba(0, 0, 0, 0.3);
        color: #fff;
    }
    ::placeholder {
        color: #ddd;
    }
    /* Style the "Add" button */
    .btn__add {
        padding: 10px;
        width: 15%;
        background: #7159c1;
        color: #fff;
        float: left;
        text-align: center;
        font-size: 16px;
        cursor: pointer;
        transition: 0.3s;
        border-radius: 0;
    }
```

```
.btn__add:hover {
  background-color: rgba(200, 200, 200, 0.3);
}
</style>
</head>
<body>
  <div class="intro">
    <h1>Hi Electron!</h1>
    <p>
      We are using node <script>document.write(process.
      Chrome <script>document.write(process.versions.
      and Electron <script>document.write(process.ver
    </p>
    <h3>Meu primeiro app feito em Electron, usando só
  </div>
  <div class="card__todo">
    <h2>O que vou fazer hoje?</h2>
    <input type="text" onkeyup="addNewTodo(event)" id="
    <span onclick="addNewTodo()" class="btn__add">ADD<
  </div>
  <ul id="ulTodo">
    <li class="checked">Fazer inscrição na NLW02</li>
    <li>Ler um post no blog da Rocketseat</li>
    <li class="checked">Estudar JavaScript</li>
    <li>Estudar a Doc do Electron</li>
  </ul>
  <script>
    console.log(" ---> Lendo o arquivo todos.js");

    // Cria os botões para excluir um item da lista de
    const myNodeList = document.querySelectorAll("li")
    myNodeList.forEach((_, index) => {
      let span = document.createElement("span");
      let iconDelete = document.createTextNode("\u00D7");
      span.className = "close";
      span.appendChild(iconDelete);
      myNodeList[index].appendChild(span);
    });
  </script>
</body>
</html>
```

```
});

// Clicando no botão para excluir a tarefa da list
const closesButton = document.querySelectorAll(".c

closesButton.forEach( (_, index) => {
  closesButton[index].onclick = function () {
    const div = this.parentElement;
    div.style.display = "none";
  };
});

// Marca/desmarca a tarefa como feita quando clica
const todoList = document.querySelector("ul");
todoList.addEventListener(
  "click",
  function (e) {
    if (e.target.tagName === "LI") {
      e.target.classList.toggle("checked");
    }
  },
  false
);

// Cria uma nova tarefa na lista de tarefas quando
function addNewTodo(event) {
  if (event && event.keyCode !== 13) return;

  const li = document.createElement("li");
  const inputTodo = document.getElementById("input
  let t = document.createTextNode(inputTodo);
  li.appendChild(t);
  if (inputTodo === "") {
    alert("Digite alguma tarefa!");
  } else {
    document.getElementById("ulTodo").appendChild(
  }
```



```
document.getElementById("inputTodo").value = "";

const span = document.createElement("SPAN");
const iconDelete = document.createTextNode("\u2716");
span.className = "close";
span.appendChild(iconDelete);
li.appendChild(span);

span.onclick = function () {
  const div = this.parentElement;
  div.style.display = "none";
};
}

</script>
</body>
</html>
```

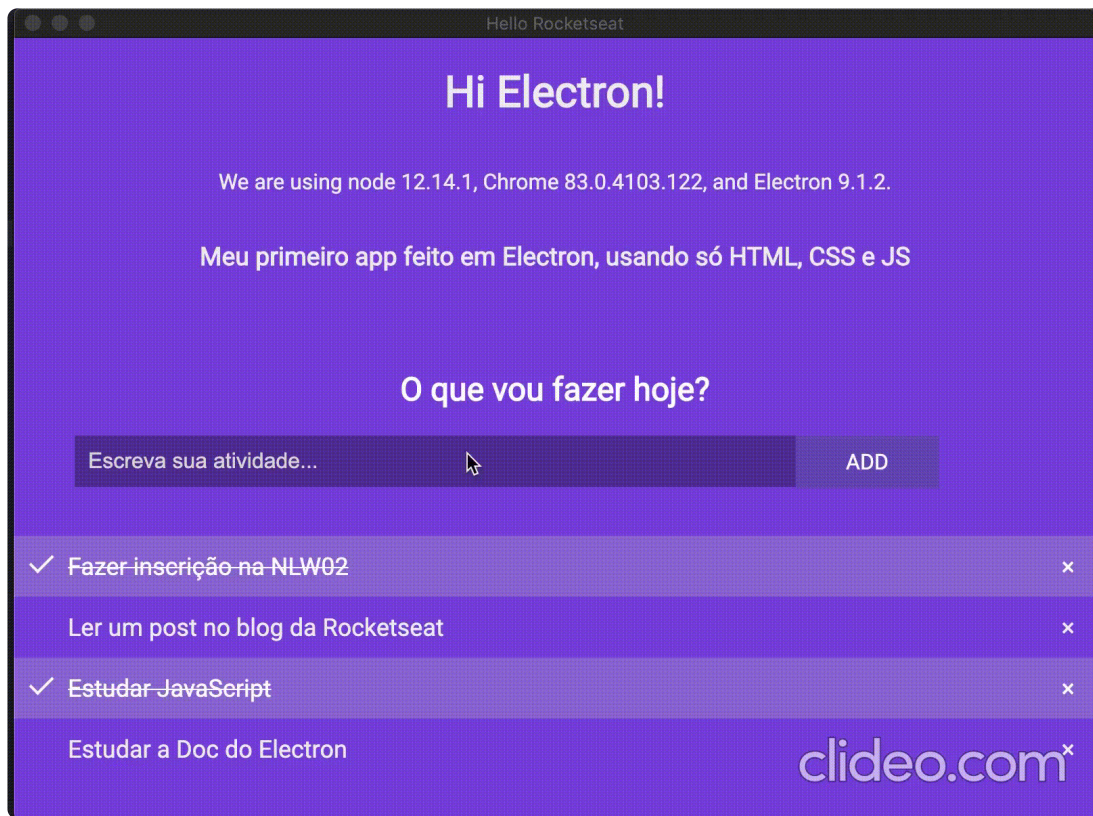
Pronto! Finalizamos o nosso app desktop.

Executando o projeto

Acesse o terminal e na raiz do projeto execute:

```
npm start ou electron main.js
```

O projeto será iniciado e você vai ter esse resultado:



Resultado Final - App Hi Electron

Conclusão

Gostei bastante da possibilidade de criar um aplicativo Desktop bonito e de maneira fácil, usando as tecnologias da Web, podemos utilizar Bootstrap, React, Vue.js, qualquer tecnologia que execute no navegador e facilite a experiência de construção de telas (UI).

Acredito que com HTML, CSS e JS é mais fácil e rápido implementar melhores práticas de UI/UX do que com aplicações Desktop feitas com Java, C#, Visual Basic ou Delphi. *Deixe sua opinião sobre isso nos comentários.*

Acho incrível a possibilidade de reaproveitar os conhecimentos da web para desenvolver apps desktop.

Tem mais assuntos avançados sobre Electron que podemos trazer em um próximo post como: acesso a banco de dados, implementação na prática da comunicação RPC com ipcRender e ipcMain.



O que o time da Rocketseat fez?

- Com Electron, **Mayk Brito** implementou um screenboard que ajuda a dar suas aulas, sublinhando e circulando os textos para dar ênfase no que ele está ministrando.
- Implementou também um navegador flutuante que permite abrir uma janela inteira porém manter uma tela flutuante sem que ela saia da tela quando clica em outra janela. Pode ver o passo a passo da implementação aqui.
- **Diego Fernandes** está construindo o RocketRedis – Uma linda interface gráfica para acessar a base de dados do Redis. Pode ver a introdução sobre Electron e da ferramenta RocketRedis aqui.




Links

- Site oficial
- Documentação oficial
- FAQ
- Twitter Oficial
- Comunidade do Electron
- Community e Awesome Electron
- Electron build


- [Ajude na tradução da documentação](#)
- [Apps construídos com Electron](#)

Espero que tenha curtido! 

O aprendizado é contínuo e sempre haverá um próximo nível! 

READ MORE POSTS BY THIS AUTHOR

Thiago Marinho

Dev Apaixonado por Tecnologia & Educação! Evolua rápido como a tecnologia, aprendizado é contínuo e sempre haverá um próximo nível. Boost Yourself! 



PRÓXIMO POST

Entendendo Falsy e Truthy no JavaScript

POST ANTERIOR

Recoil - biblioteca de gerenciamento de estados no React

Deixe sua reação...

7 Responses



Gostei!



Amei!



Uau!



Não gostei!

7 Comentários

Blog Rocketseat



Política de Privacidade



Entrar



Recomendar 3



Tweet



Compartilhar

Ordenar por Mais votados



Participe da discussão...

FAZER LOGIN COM

OU REGISTRE-SE NO DISQUS ?

Nome

Adriel Borges • um ano atrás

show de bola

1 ^ | v • Responder • Compartilhar ›

While(alive) learn(); • 2 meses atrás

Não está funcionando com a versão atual do Electron. Não acessa o



rocketseat

BLOG

HOME

BACK END

FRONT END

MOBILE



© 2021 **Blog da Rocketseat**. Feito com <3. Published with **Ghost**.