

Criando uma aplicação React.js online via <https://playcode.io/>

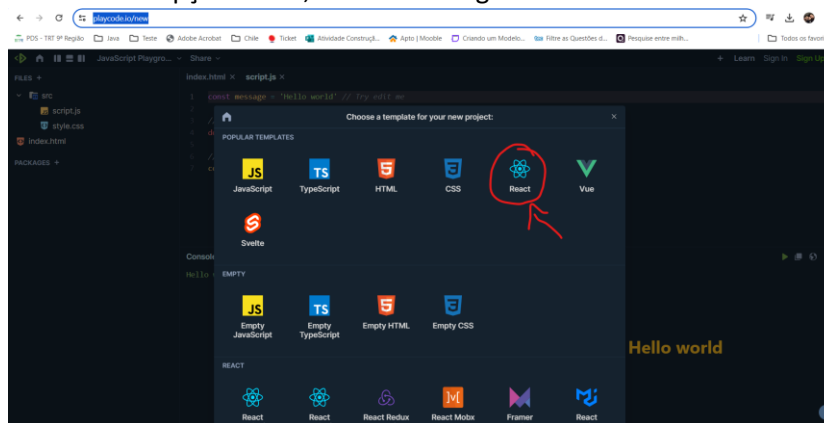
## Contextualização

No exemplo a seguir, iremos utilizar a IDE Online <https://playcode.io> para não termos nenhuma dependência de configuração de ambiente, mas a implementação do front pode ser feito também de forma local, porém, será necessário ter instalado o Node no computador e recomenda-se os plugins para React e Node no IDE a ser utilizado.

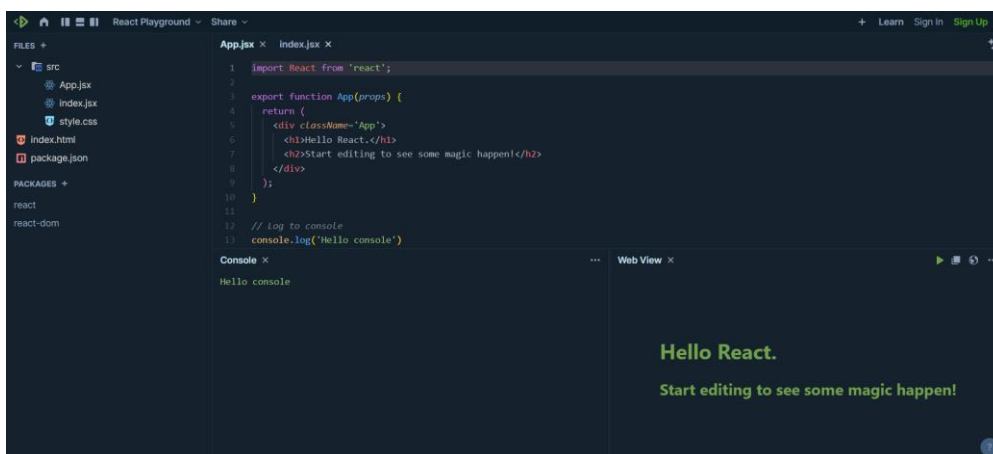
## Criando a aplicação.

### Passo 1

1. Digite em seu terminal `node -v`. Caso tenha o Node instalado pule para o próximo passo, senão, você pode criar o seu projeto padrão no endereço: <https://playcode.io/new>. Basta selecionar a opção React, conforme a figura abaixo.



Com a aplicação criada, é hora de conhecer um pouco mais sobre a estrutura do React e do ambiente <https://playcode.io>



2. Caso você tenha o Node instalado, basta digitar `npx create-react-app@latest`. No seu terminal e responder Yes.  
Quando terminar a instalação, digitar `npm start` e verificar se temos o seguinte:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Compiled successfully!

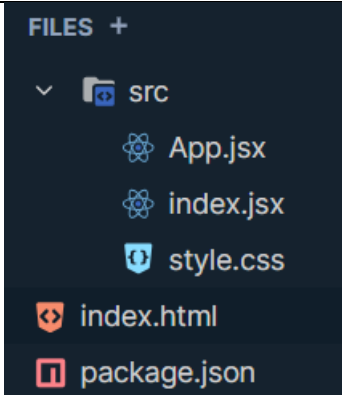
You can now view front_todo_list in the browser.

Local:          http://localhost:3000
On Your Network: http://192.168.5.116:3000

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

webpack compiled successfully
```

## Passo 2: Estrutura de Arquivos

Estrutura no <a href="https://playcode.io/">https://playcode.io/</a>	
	<p>No nível raiz temos:</p> <ul style="list-style-type: none"><li>• <b>src:</b> pasta responsável pelos arquivos de código do projeto React.</li><li>• <b>index.html:</b> arquivo de entrada ao projeto (estático) e que não necessita ser alterado. Basicamente ele aponta para o arquivo <b>src/index.jsx</b>.</li><li>• <b>package.json:</b> arquivo responsável pela configuração do projeto, não deve ter o nome alterado e nele, temos as configurações do projeto, as dependências e os scripts necessários. É um arquivo padrão dos frameworks baseado em Node.JS</li></ul>
Estrutura no VSCode	

EXPLORER

...  
▼ FRONT\_TODO\_LIST  
  > node\_modules  
  ▼ public  
    ★ favicon.ico  
    <> index.html  
    🖼 logo192.png  
    🖼 logo512.png  
    {} manifest.json  
    📄 robots.txt  
  ▼ src  
    # App.css  
    JS App.js  
    JS App.test.js  
    # index.css  
    JS index.js  
    🖼 logo.svg  
    JS reportWebVitals.js  
    JS setupTests.js  
  🔍 .gitignore  
  {} package-lock.json  
  {} package.json  
  📄 README.md

No nível raiz temos:

- **src:** pasta responsável pelos arquivos de código do projeto React. **Vamos excluir todos os arquivos desta pasta.**
- **public:** pasta onde devem ficar os arquivos públicos a serem expostos na app. **Vamos excluir todos os arquivos desta pasta.**
- **package.json:** arquivo responsável pela configuração do projeto, não deve ter o nome alterado e nele, temos as configurações do projeto, as dependências e os scripts necessários. É um arquivo padrão dos frameworks baseado em Node.JS

### Passo 3: Os arquivos da pasta src

- **App.jsx:** é o aplicativo em React propriamente dito e é nele que vamos definir as questões básicas do nosso frontend. Este arquivo deve ter seu conteúdo alterado, mas nunca devemos alterar o seu nome, para que possamos manter o padrão do framework.
- **index.jsx:** é o arquivo que faz a ligação entre o index.html da pasta raiz, com o arquivo App.jsx. Este arquivo pode ter seu conteúdo alterado, mas nunca devemos alterar o seu nome, para que possamos manter o padrão do framework.
- **Style.css ou App.css:** é o arquivo responsável pelo estilo do app. Parte da comunidade prefere alterar seu nome para app.css, mas caso isso ocorra, devemos alterar também sua importação no arquivo **App.jsx**.

**Observação,** alguns programadores utilizam a extensão js para os arquivos relativo a código fonte, mas é uma boa prática utilizar a extensão jsx, para explicitar que se trata de um arquivo React e não apenas javascript.

### Passo 4: O arquivo index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="src/style.css">
  </head>
  <body>
    <div id="root"></div>
```

```
<script src="src/index.jsx"></script>
</body>
</html>
```

#### Passo 5: O arquivo src/index.jsx

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<App />);
```

Segue as explicações de cada trecho.

- Linha 1: importa a biblioteca principal do React.
- Linha 2: Importa a biblioteca ReactDOM
- Linha 3: Importa o arquivo src/App.jsx
- Bloco entre linha 5 e 6: responsável por alterar o index.html e injetar o código React [DOM](#) do arquivo.

#### Passo 6: Componentizando o projeto

Uma boa prática em toda aplicação, é realizar a [componentização do projeto](#), e em React não é diferente. Para isso, vamos criar uma pasta chamada **src/components** e dentro da pasta, iremos criar um arquivo de [wrapper](#) chamado **src/ToDoWrapper.jsx**.

Inicialmente, no arquivo ToDoWrapper.jsx vamos colocar o conteúdo exibido no arquivo src/App.jsx, conforme o exemplo abaixo.

FILES +

- src
  - App.jsx
- components
  - ToDoWrapper.jsx
- index.jsx
- style.css
- index.html
- package.json

PACKAGES +

- react
- react-dom

ToDoWrapper.jsx

```
import React from 'react';
export const ToDoWrapper = () => {
  return (
    <div className='App'>
      <h1>Hello React.</h1>
      <h2>Start editing to see some magic happen!</h2>
    </div>
  );
}
```

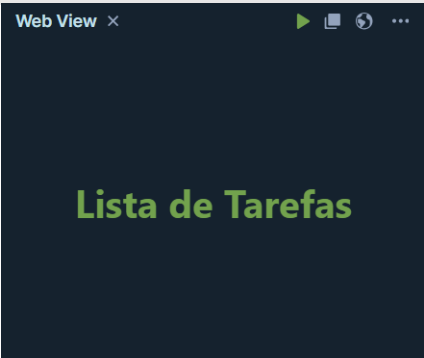
### Passo 7: O arquivo src/App.jsx

Como este arquivo é responsável por representar nossa aplicação propriamente dita, será necessário alterá-lo, e iremos agora ao invés de colocar o conteúdo da página principal, iremos apontar para o arquivo criado no passo anterior (TodoWrapper.jsx), conforme o exemplo abaixo.

```
App.jsx
import React from 'react';
import { TodoWrapper } from './components/TodoWrapper';

export function App(props) {
  return (
    <div className='App'>
      <TodoWrapper />
    </div>
  );
}
```

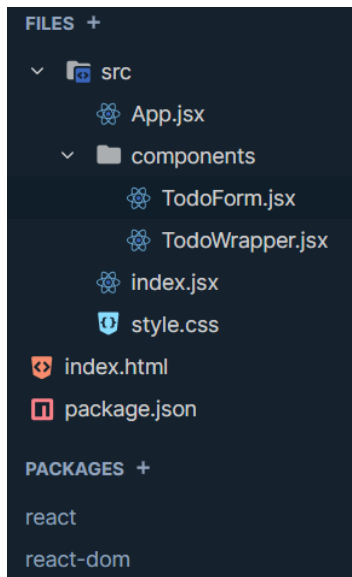
### Passo 8: Editando o TodoWrapper.jsx para começar a parecer nossa lista de Tarefas

TodoWrapper.jsx	Visualização
<pre>import React from 'react'; export const TodoWrapper = () =&gt; {   return (     &lt;div className='TodoWrapper'&gt;       &lt;h1&gt;Lista de Tarefas&lt;/h1&gt;     &lt;/div&gt;   ); }</pre>	 A screenshot of a web browser window titled 'Web View'. The page has a dark blue background and displays the text 'Lista de Tarefas' in a large, bold, green font.

### Passo 9: Criando o Componente de TodoForm.jsx

O componente TodoForm.jsx será responsável pela inclusão de novas tarefas em nosso aplicativo.

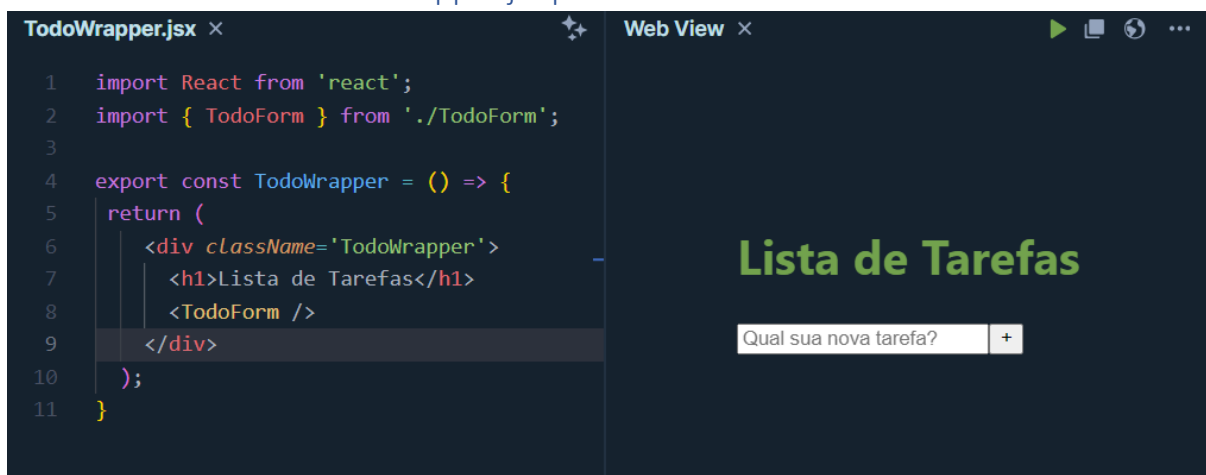
Organização dos arquivos	TodoForm.jsx
--------------------------	--------------



```
import React from 'react';

export const TodoForm = () => {
  return (
    <form className='TodoForm'>
      <input
        type='text'
        className='todo-input'
        placeholder='Qual sua nova tarefa?'
      />
      <button type='submit' className='todo-btn'>
        +
      </button>
    </form>
  );
};
```

Passo 10: Editando o TodoWrapper.jsx para incluir o TodoForm

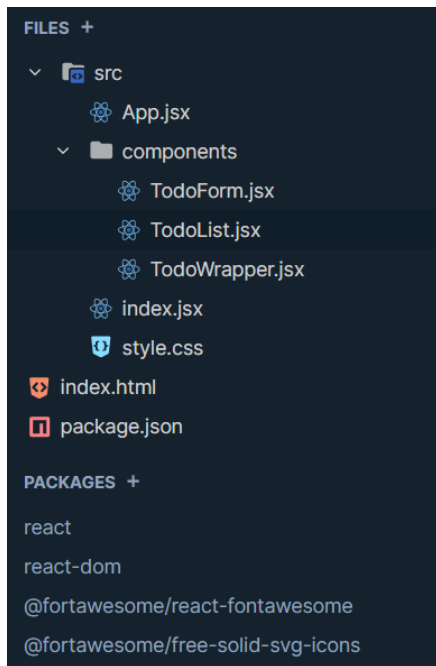


No arquivo TodoWrapper.jsx, basta incluir a linha de importação do arquivo TodoForm.jsx (linha 2) e a linha do Componente (linha 8).

Passo 9: Criando o Componente TodoList.jsx

O componente TodoList.jsx será responsável por listar as tarefas em nosso aplicativo.

### Organização dos arquivos



Será necessário a inclusão de 2 pacotes, para que possamos utilizar ícones em nossa aplicação.

Os pacotes são:

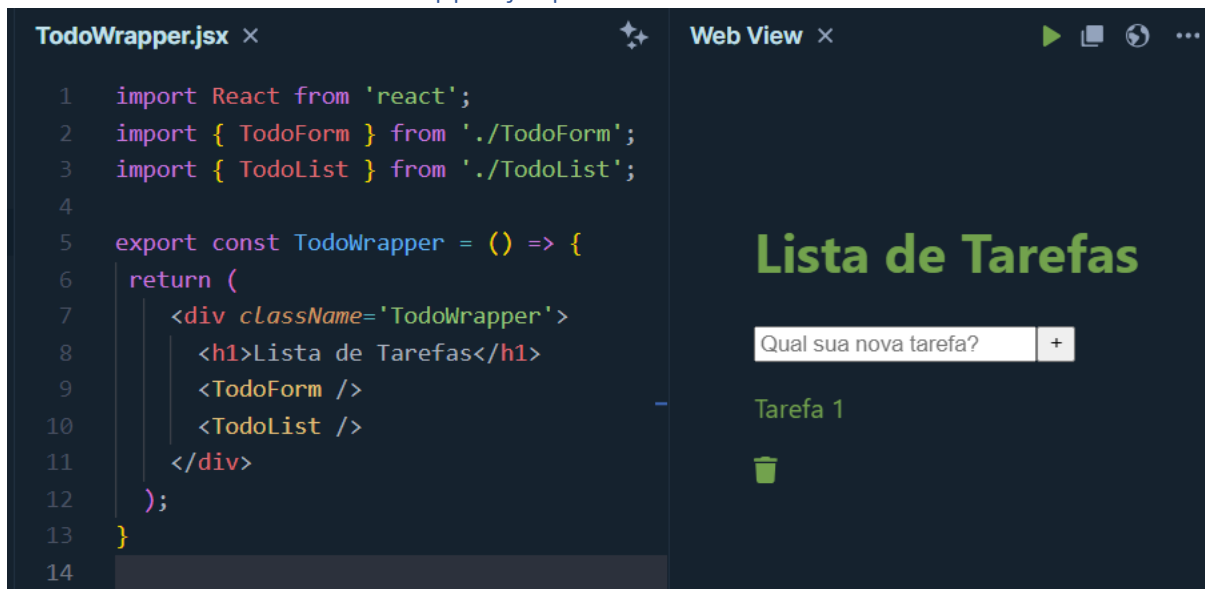
- @fortawesome/react-fontawesome
- @fortawesome/free-solid-svg-icons

### TodoList.jsx

```
import React from 'react';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { faTrash } from '@fortawesome/free-solid-svg-icons';

export const TodoList = () => {
  return (
    <div className='Todo'>
      <p
        className='completed'
      >
        Tarefa 1
      </p>
      <div>
        <FontAwesomeIcon
          className='delete-icon'
          icon={faTrash}
        />
      </div>
    </div>
  );
};
```

### Passo 11: Editando o TodoWrapper.jsx para incluir o TodoList



Incluimos as linhas 3 e 10.

### Passo 12: Alterando estilo da aplicação

```
* {
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, Oxygen,
  Ubuntu, Cantarell, 'Open Sans', 'Helvetica Neue', sans-serif;
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

body {
  background: transparent; /* Make it white if you need */
  display: flex;
  justify-content: center;
  align-items: center;
}

.App {
  color: #72a24d;
  text-align: center;
}

h1 {
  color: #fff;
  margin-bottom: 0.5rem;
  font-size: 1.75rem;
}
```



```
.TodoWrapper {
  margin-top: 5rem;
  padding: 2rem;
  border-radius: 5px;
}

.TODOForm {
  width: 100%;
  margin-bottom: 1rem;
}

.todo-input {
  outline: none;
  background: none;
  border: 1px solid #8758ff;
  padding: 0.5rem 1rem;
  margin-top: 1rem;
  margin-bottom: 2rem;
  width: 300px;
  color: #fff;
}

.todo-input::placeholder {
  color: #ffffff4d;
}

.todo-btn {
  background: #8758ff;
  color: #fff;
  border: none;
  padding: 0.55rem;
  cursor: pointer;
}

.TodoList {
  display: flex;
  justify-content: space-between;
  align-items: center;
  background: #8758ff;
  color: #fff;
  padding: 0.75rem 1rem;
  border-radius: 5px;
  margin-bottom: 1rem;
}
```

```
.fa-trash {  
  margin-left: 0.75rem;  
}  
  
.incompleted {  
  color: #fff;  
  text-decoration: line-through;  
  cursor: pointer;  
}  
  
.completed {  
  color: #c5aeff;  
  text-decoration: line-through;  
  cursor: pointer;  
}  
  
.delete-icon {  
  cursor: pointer;  
}
```

## Sugestões de leitura

- ☐ <https://pt-br.react.dev/blog/2023/03/16/introducing-react-dev>
- ☐ <https://react.dev/learn>
- ☐ <https://nextjs.org/learn>
- ☐ <https://nextjs.org/docs/getting-started/installation>
- ☐ <https://www.dio.me/articles/deciso-es-cruciais-como-escolher-o-framework-de-front-end-ideal>