




Ignite



ReactJS



Desafio 02 - Refactoring de classes e typescript

 [Sobre o desafio](#)

[Template da aplicação](#)

[Se preparando para o desafio](#)

[Fake API com JSON Server](#)

[O que devo editar na aplicação?](#)

[Preparando ambiente Typescript](#)

[Estou com dificuldade na conversão classes→função](#)

[Como deve ficar a aplicação ao final?](#)

 [17 Entrega](#)



Sobre o desafio

Nesse desafio, você deverá criar uma aplicação para treinar o que aprendeu até agora no ReactJS

Essa será uma aplicação já funcional onde o seu principal objetivo é realizar dois processos de migração: de Javascript para Typescript e de Class Components para Function Components.

A seguir veremos com mais detalhes o que e como precisa ser feito 🚀

Template da aplicação

Para realizar esse desafio, criamos para você esse modelo que você deve utilizar como um template do GitHub.

O template está disponível na seguinte URL:

github.com

<https://github.com/rocketseat-education/ignite-template-reactjs-refactoring-classes-ts>

Dica: Caso não saiba utilizar repositórios do GitHub como template, temos um guia em [nosso FAQ](#).

Se preparando para o desafio

Para esse desafio, além dos conceitos vistos em aula utilizaremos o JSON server para criar uma Fake API com os dados das comidas.

Fake API com JSON Server

Assim como utilizamos o MirageJS no módulo 2 para simular uma API com os dados das transações da aplicação dt.money, vamos utilizar o JSON Server para simular uma API que possui as informações das comidas.

Navegue até a pasta criada, abra no Visual Studio Code e execute os seguintes comandos no terminal:

```
yarn yarn server
```

Em seguida, você vai ver a mensagem:

```

) yarn server
yarn run v1.22.10
$ json-server server.json -p 3333

\{^_^}/ hi!

Loading server.json
Done

Resources
http://localhost:3333/foods

Home
http://localhost:3333

Type s + enter at any time to create a snapshot of the database

```

Perceba que ele iniciou uma fake API com o recurso `/foods` em `localhost` na porta `3333` a partir das informações do arquivo `server.json` localizado na raiz do seu projeto. Acessando essa rota no seu navegador, você consegue ver o retorno das informações já em JSON:

```

1 // 20210310100913
2 // http://localhost:3333/foods
3
4 ▾ [
5   ▾ {
6     "id": 1,
7     "name": "Ao molho 2",
8     "description": "Macarrão ao molho branco, fughi e cheiro verde das montanhas",
9     "price": "19.90",
10    "available": false,
11    "image": "https://storage.googleapis.com/golden-wind/bootcamp-gostack/desafio-food/food1.png"
12  },
13  ▾ {
14    "id": 2,
15    "name": "Veggie",
16    "description": "Macarrão com pimentão, ervilha e ervas finas colhidas no himalaia.",
17    "price": "21.90",
18    "available": true,
19    "image": "https://storage.googleapis.com/golden-wind/bootcamp-gostack/desafio-food/food2.png"
20  },
21  ▾ {
22    "id": 3,
23    "name": "A la Camarón",
24    "description": "Macarrão com vegetais de primeira linha e camarão dos 7 mares.",
25    "price": "25.90",
26    "available": false,
27    "image": "https://storage.googleapis.com/golden-wind/bootcamp-gostack/desafio-food/food3.png"
28  }
29 ]

```

Dessa forma, basta consumir essas rotas da API normalmente com o Axios. Caso queira estudar mais sobre o **JSON Server**, dê uma olhada aqui:

typicode/json-server

Get a full fake REST API with zero coding in less than 30 seconds (seriously) Created with public/index.html json-server db.json

 <https://github.com/typicode/json-server>



O que devo editar na aplicação?

Com o template já clonado, as dependências instaladas e a fake API rodando, você deve editar os seguintes arquivos:

- src/components/Food/index.jsx;
- src/components/Food/styles.js;
- src/components/Header/index.jsx;
- src/components/Header/styles.js;
- src/components/Input/index.jsx;
- src/components/Input/styles.js;
- src/components/Modal/index.jsx;
- src/components/ModalAddFood/index.jsx;
- src/components/ModalAddFood/styles.js;
- src/components/ModalEditFood/index.jsx;
- src/components/ModalEditFood/styles.js;
- src/pages/Dashboard/index.jsx;
- src/pages/Dashboard/styles.js;
- src/routes/index.jsx;
- src/services/api.js;
- src/styles/global.js;
- src/App.js;
- src/index.js.

Todos esses arquivos devem ser migrados de Javascript para Typescript. Além disso, os arquivos que possuírem componentes em classe devem ser migrados para componentes funcionais.

Preparando ambiente Typescript

Como esse é um projeto CRA sem TypeScript, você primeiro precisa instalar as dependências/tipagens e configurar o TS. Nossa sugestão é criar um novo projeto CRA com Typescript e comparar a estrutura atual com a que você precisa ter.

Realizando essa comparação, facilmente você consegue ver que:

- É preciso instalar o `typescript`
- É preciso criar um arquivo de configuração `tsconfig.json`. Inclusive, você pode utilizar a configuração gerada automaticamente no CRA template Typescript para criar o seu arquivo.
- É preciso criar um arquivo `react-app-env.d.ts` com o conteúdo:

```
/// <reference types="react-scripts" />
```

- É preciso instalar as tipagens das bibliotecas.

Configurando esse setup, você deve ser capaz de trabalhar normalmente com o Typescript no seu projeto.

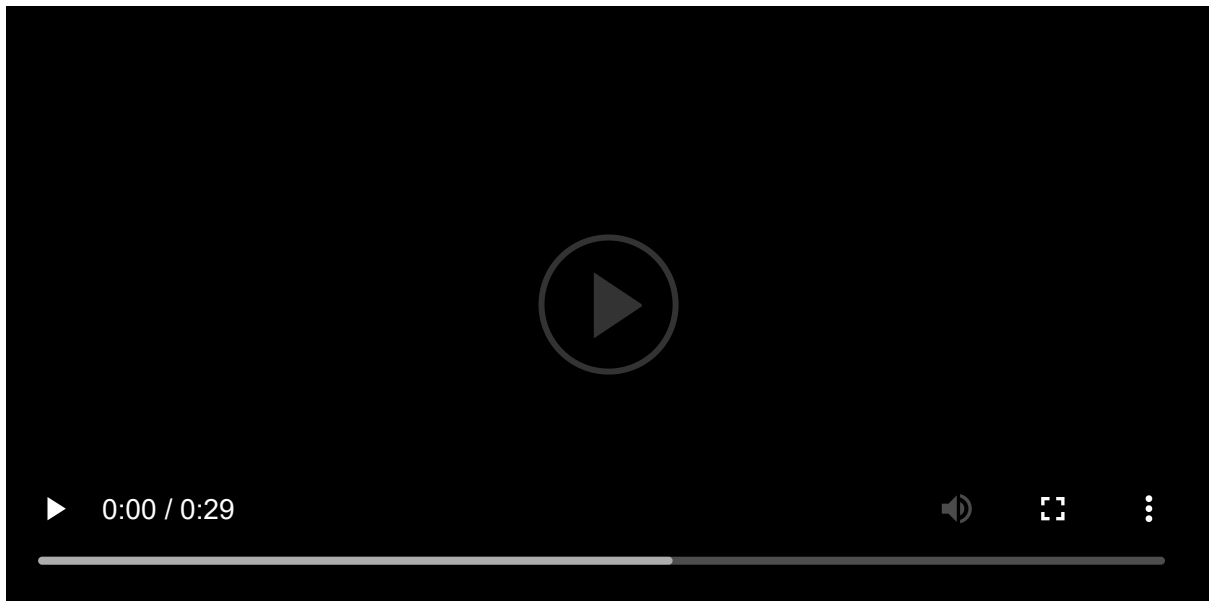
Estou com dificuldade na conversão classes→função

Caso você tenha dificuldade nesse processo de migração, dê uma olhada no nosso guia sobre esse assunto:

 [Componentes no React](#)

Como deve ficar a aplicação ao final?

Nesse desafio você já recebe a aplicação totalmente funcional, então todos os recursos mostrados no vídeo abaixo já estão implementados no template e devem permanecer funcionando após suas alterações.



Entrega

Esse desafio deve ser entregue a partir da plataforma da Rocketseat. Envie o link do repositório que você fez suas alterações. Após concluir o desafio, além de ter mandado o código para o GitHub, fazer um post no LinkedIn é uma boa forma de demonstrar seus conhecimentos e esforços para evoluir na sua carreira para oportunidades futuras.

Feito com  por Rocketseat  Participe da nossa [comunidade aberta!](#)