DTI Inventory Project

Este projeto foi desenvolvido utilizando a library **ReactJS** e **Javascript** para a construção frontend e no backend utilizou-se **Typescript** e **NodeJS**. As tecnologias utilizadas foram empregadas pela a maior familiaridade da equipe do projeto, assim como as especificações repassadas. O banco de dados utilizado foi o **MySQL** devido a ampla comunidade que o utiliza,e principalmente, pelo o fato de ter grande desempenho em em sistemas com elevados números de consultas, o que é compatível com o projeto desenvolvido.

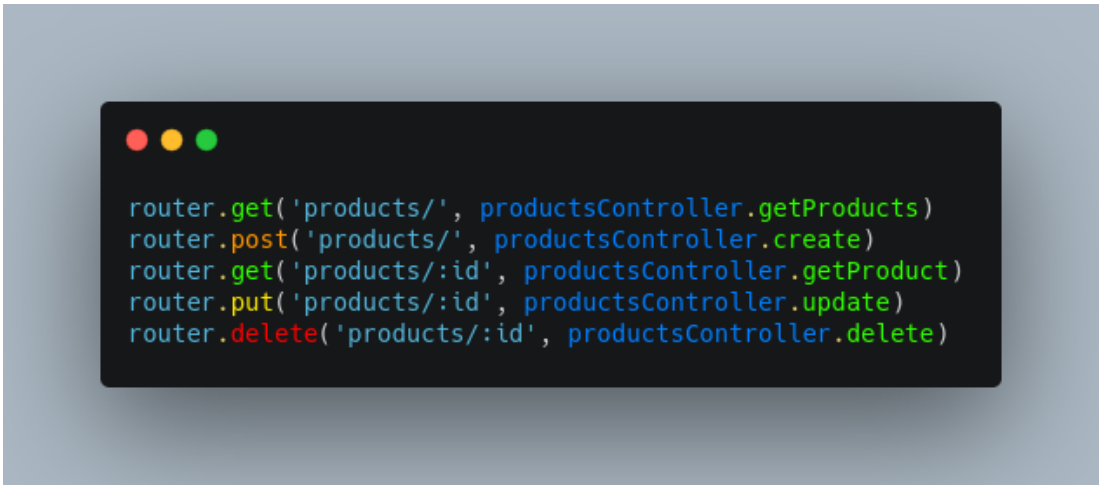
A construção do sistema deu-se inicialmente pela a criação do schema no MySQL denominado-o **dti_backend**. A conexão do banco com o backend deu-se por meio do query builder **Knex**, ao qual a equipe já possui maior familiaridade em utilizar. A tabela **products** foi criada por meio de um script desenvolvido na aplicação que pode ser adaptado para criação de novas tabelas, de acordo com o crescimento do sistema. A seguir temos um esquema da tabela:

id	name	quantity	price	created_at
VARCHAR(255) PRIMARY KEY	VARCHAR(255)	INT	INT	DATETIME DEFAULT CURRENT_TIMESTAMP

Backend - API Rest

A arquitetura da API Rest consistiu no modelo MVC em que os **models** são responsáveis por descrever as interfaces das rotas, os **controllers** são responsáveis por fazer a mediação da entrada e saída de requisições, e as **views** são responsáveis pelas as regras de negócio. Além dessa estrutura, temos os **middlewares** responsáveis por fornecerem serviços à aplicação, neste caso, geração de id com **uuid4**. O “folder” **database** é responsável pela a conexão do banco com a API e o “folder” **data** é responsável pelas as consultas e pela as criações das tabelas.

As rotas criadas na api para consulta da lista de produtos, a consulta de uma produto, a criação, edição e remoção de um produto seguem abaixo:



A API Rest foi desenvolvida utilizando classes o que facilita a criação de métodos de acordo com a necessidade da aplicação.

Frontend - React

A estrutura da aplicação no front end consiste dos seguintes folders:

1. **assets:** armazenamento de imagens/ícones a serem utilizadas na aplicação
2. components: componentes que são utilizados nas screens ou em outros componentes (reutilizáveis)
3. **routes:** criação de rotas para as páginas da aplicação, neste caso **Home, Cadastro e Edição**
4. **screens:** local em que ficam as páginas da nossa aplicação, neste caso as descritas acima
5. **services:** serviços que aplicação faz uso, neste caso as requisições feitas no backend por meio do **axios**
6. **store:** estrutura do Redux Saga, composto por reducers, actions e sagas, também foi adicionado o persistor para uma eventual necessidade de persistir os dados da aplicação.
7. styles: estrutura comum do estilo da aplicação.

No frontend para o gerenciamento de estado a aplicação foi utilizado o Redux Saga, visto que é amplamente utilizado, possui um grande respaldo de manutenção de seus criadores e a equipe de implementação tem facilidade em utilizá-lo. Ele está estruturado no store em que ocorre o setup do **Redux Saga** e em **modules** que possibilitam o crescimento da aplicação com organização.

As actions são responsáveis por, justamente, realizar ações predefinidas, que são interceptadas pelo o middleware Saga, em que através das funções generators é possível fazer requisições assíncronas no banco de dados e aguardar o recebimento das Promises para serem disponibilizadas na store à toda a aplicação.

O design da aplicação foi feito utilizando **styled-components** que permite

escrever “css” usando javascript. Isso torna o design de cada componente mais local e facilita a manutenção e diminui problemas relacionados a conflitos de variáveis. A fim de melhorar a usabilidade dos usuários, foi utilizado o **react-toastify** para envio de mensagens de resposta aos usuários.

As rotas foram desenvolvidas por meio do react-router-dom.

Projeto desenvolvido por: João Paulo Rodrigues Pereira