Sistema de Estoque de Produtos

1. Introdução

1.1. FrontEnd

O FrontEnd foi desenvolvimento com React Native e testado no sistema operacional Android. Também foi utilizado o Expo (https://expo.io/) como framework para criar o projeto e buildar a aplicação.

1.2. BackEnd

O BackEnd foi desenvolvido em Node.js, utilizando o framework Express.js (https://expressjs.com/pt-br/).

1.3. BD

O Banco de Dados foi desenvolvido em MySQL (https://www.mysql.com/).

2. Como rodar o projeto

2.1. Clonar a pasta

Utilizando o Git, clonar o projeto
<https://github.com/JuliaAbud/sistemaEstoque.git>

2.2. Baixar o aplicativo do Expo

Como o aplicativo foi desenvolvido com um aparelho Android para testes, o aplicativo Expo deve ser baixado na Play Store.
<https://play.google.com/store/apps/details?id=host.exp.exponent&hl=pt_BR>

2.3. FrontEnd

Após clonar o projeto, abrir a pasta "estoqueDti", que contém o FrontEnd do projeto, rode o comando `npm install`. Mude também a URL das requisições para a que estará rodando o BackEnd(com '/' no final), que está armazenada em uma variável dentro da pasta `environments`, no arquivo `dev.js`. Depois que o npm install terminar de instalar as dependências, digite `npm start` no terminal para rodar o FrontEnd. Para acessar do celular, abra o aplicativo da Expo e escaneie o QR Code gerado no terminal.

2.4. BackEnd

Após clonar o projeto, abrir a pasta “backendDti”, que contém o BackEnd do projeto, rode o comando `npm install`. Depois que o npm install terminar de instalar as dependências digite `node server.js` no terminal para rodar o BackEnd.

2.5. BD

O banco do projeto foi feito em MySQL e o dump do banco está na pasta “bdDti”. Também inclui um model do Workbench, caso seja utilizado para rodar o projeto.

3. Estrutura do projeto

3.1. FrontEnd

3.1.1 Componentes

- ➔ **DeleteButton**: contém a estrutura da imagem do botão, com um método de que retorna para o componente pai o evento de click.
- ➔ **EditButton**: contém a estrutura da imagem do botão, com um método de que retorna para o componente pai o evento de click.
- ➔ **InputContainer:** é o componente pai para o conjunto de TextInput e o botão de cadastro de produtos. Contém três componentes de input, um botão e três componentes de label. Recebe e envia os estados por meio do Redux. Também gerencia as requisições de editar/criar um produto.
- ➔ **InputLabel:** contém a estrutura de label dos inputs. Recebe o texto a ser exibido.
- ➔ **InputText:** contém a estrutura de TextInput. Retorna o texto digitado.
- ➔ **Product:** contém a estrutura de exibição do produto. Recebe os atributos de produto.
- ➔ **ProductBox:** contém a estrutura de exibição de Product e dos botões de ação deletar/editar. O botão de editar redireciona para a tela de edição e o botão de deletar gerencia a requisição de deletar o produto.
- ➔ **ProductContainer:** é o componente pai da listagem de produtos. Gerencia os estados por meio de Redux e a requisição de listagem de produtos. Faz um map de produtos para exibir todos na tela Home.
- ➔ **App.js:** é o componente geral, que gerencia os estados, as rotas, e as telas.

3.1.2 Funções

Descrição breve das funções mais importantes do projeto.

- ➔ **Gerenciamento das chamadas de requisições:** as funções desenvolvidas são: loadProducts(), createProduct(), editProduct(), selectProduct(), que chama a requisição para listar todos os produtos, para criar um produto, para editar um produto e para selecionar um produto específico, respectivamente.
- ➔ **Requisições:** as funções desenvolvidas são: genericGet(),genericSelect(), genericPost(), genericPut() e genericDelete() que fazem a requisição para o backend, que faz um GET na rota para retornar todos os produtos, que faz um GET na rota com a key de parâmetro para retornar um produto, um POST para criar um produto, PUT para editar e DELETE para deletar, respectivamente.

3.2. Backend

Descrição breve das estruturas mais importantes do projeto.

- ➔ **Server.js:** contém a estrutura do servidor, com o roteamento das urls, as respectivas requisições e a chamada de query para o banco de dados.
- ➔ **sqlCommands.js:** contém a estrutura de conexão com o banco de dados, assim como as queries para operações no banco.

3.3. Banco de dados

O banco só possui uma tabela product, com os atributos: key como PK, name, value e qnt. A PK é automaticamente incrementada pelo SGBD.