

1. Introdução

Este documento contém as informações sobre o desenvolvimento do software Stock Manager desenvolvido por Miller César de Oliveira com parte dos requisitos para o teste da vaga de desenvolvedor da empresa DTI.

2. Requisitos

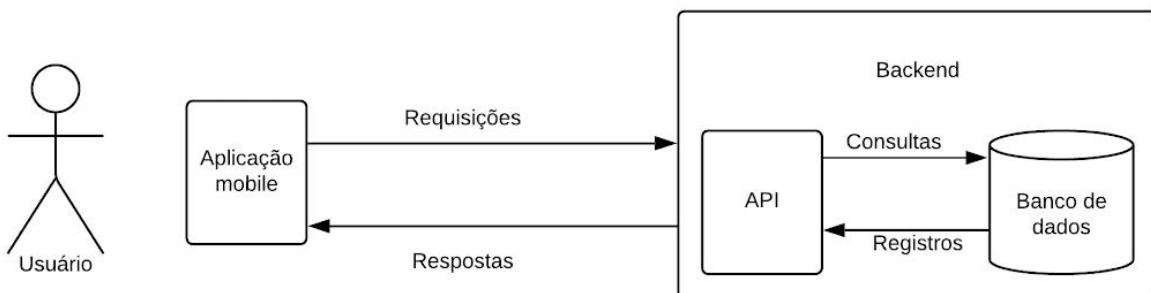
2.1. Funcionais

- O sistema deve ser capaz de cadastrar, alterar e excluir produtos;
- O sistema deve conter uma tela onde serão listadas as informações do produto, permitindo que eu selecione um produto para exclusão ou edição;
- Ao escolher a opção edição uma nova tela deverá abrir onde as informações do produto serão editadas;
- O produto deve conter um nome, quantidade e valor por unidade;
- O sistema deve possuir uma tela que permitirá o cadastramento de um novo produto.

2.2. Não funcionais

- App desenvolvido em React Native para Android com redux
- Backend desenvolvido em Ruby on Rails

3. Visão Geral



4. Mecanismos Arquiteturais

Mecanismo de análise	Mecanismo de design	Mecanismo de implementação
Persistência	Banco de dados relacional	SQLite3
Front-end	Interface de interação com usuário	React native, Javascript, Java
Back-end	Recebe requisições do Front-end, comunicação com banco de dados	Ruby on Rails
Build	Programação da IDE para validação do código fonte	Visual Studio code, Android Studio
Ambientação	Requisitos necessários para o desenvolvimento e testes.	Ubuntu/bionical 16, Vagrant, Virtual box
Testes	Mecanismos utilizados para testar o funcionamento da API	Postman, Dispositivo mobile

5. Fundamentação

Para o desenvolvimento do Front-end foi utilizado react-native-cli: 2.0.1, perfeitamente funcional com Java versão 1.8.0_231, testado junto ao Gradle 5.5.

No front end foram utilizadas as bibliotecas lodash para o tratamento dos objetos json, formik para o controle do formulário, yup para a validação dos formulários, react-native-masked-text para criar inputs que contenham máscara, react-native-router-flux para o gerenciamento de rotas e navegação, react-native-elements com o objeto de utilizar elementos para o desenvolvimento da interface, redux-thunk para o tratamento de requisições assíncronas e a axios para solicitações http na API.

Para o desenvolvimento do back-end foi utilizado rails 5 com ruby 2.5, SQLite3 banco de dados relacional compatível com o ambiente Rails, capaz de gerar um banco que pode ser entregue com a aplicação é muito eficiente para pequenos projeto, porém deve ser substituído quando posto em produção em alguns servidores. O backend foi desenvolvido em um máquina virtual utilizando Vagrant 2.2.6 e Virtual box 6.1, esta estratégia permite o

desenvolvimento com Ruby on Rails sem problemas com compatibilidade que podem ocorrer, além de criar uma máquina facilmente reutilizável em outros projetos.

No backend também foi utilizadas a gem ransack, que permite a realização de buscas com mais facilidade no Rails, e a gem active model serializer que permite a serialização dos models para Json, permitindo um fácil tratamento das respostas a serem enviadas.

O Visual Studio Code é uma IDE que permite o rápido desenvolvimento e possui diversas extensões que apoiam o programador na execução de suas tarefas. Alguns exemplos de suas vantagens são extensões para Ruby e React que permitem o auto complementação e correção de erros simplificada.

O Postman permite o teste de forma simplificada para uma API, gerando a requisição com os dados necessário e permitindo a sua documentação e compartilhamento entre equipes de desenvolvimento.

O backend foi desenvolvido na arquitetura MVC (onde a view é o JSON que retornamos), deste modo permitindo a fácil implementação de novas funcionalidades e sua ampliação.

O frontend foi desenvolvido utilizando Redux para o controle dos estados, permitindo assim que os componentes tenham acesso a uma única fonte de verdade e gerando rapidez na criação de novas funcionalidades.


Foram utilizados os verbos GET (recuperar todos os produtos ou apenas um), POST (criar um novo produto), PUT (atualizar algum produto), DELETE (excluir um produto).

As rotas definidas foram:

- GET /products (recupera todos os produtos)
- GET /products/:id (recupera um produto)
- POST /products (salva um novo produto)
- PUT /products/:id (atualiza um produto existente)
- DELETE /products/:id (apaga um produto do banco de dados)

Os produtos criados ou editados são validados no front end utilizando a biblioteca yup, auxiliando assim na garantia que os dados chegarão de maneira correta ao backend.

6. Esquema banco de dados

Products		
	id	integer
	name	string
	value	float
	quantity	integer