Sistema de Controle de Estoque - Documentação Técnica

Projeto: Sistema de Controle de Estoque

Linguagem: Haskell

Docente: José Antônio

Discentes: Tallysson Luiz e Kéwen Silva

1. Introdução

O sistema de controle de estoque é uma aplicação desenvolvida em Haskell, com o objetivo de gerenciar produtos e suas quantidades em estoque. A aplicação utiliza arquivos CSV para armazenamento de dados persistentes e implementa uma série de funcionalidades para manipulação de produtos, como adicionar, remover, listar e buscar itens.

Este documento descreve a estrutura do código, as funcionalidades implementadas e o fluxo de execução da aplicação.

2. Estrutura do Projeto

O projeto é composto pelos seguintes arquivos:

- **main.hs**: Arquivo principal do projeto, responsável por gerenciar o fluxo do programa e interações com o usuário.
- **produto.hs**: Define a estrutura dos dados de um produto e funções relacionadas à manipulação dos produtos.
- **estoque.hs**: Implementa as funcionalidades de gerenciamento do estoque, como adicionar e remover produtos.
- **utils.hs**: Funções utilitárias que auxiliam na leitura e escrita de arquivos, manipulação de strings, entre outras.
- **estoque.csv**: Arquivo que armazena os dados dos produtos no formato CSV.
- **.vscode**: Contém configurações de ambiente para facilitar o desenvolvimento no Visual Studio Code.

2.1 - Fluxo Geral do Sistema

O fluxo do sistema segue os seguintes passos:

1. **Leitura Inicial do Estoque**: No início da execução, o sistema carrega os dados do arquivo estoque.csv.
2. **Interação com o Usuário**: O usuário pode escolher diferentes operações, como listar produtos, adicionar, remover, etc.
3. **Atualização do Estoque**: Após cada operação, o arquivo CSV é atualizado para refletir as mudanças.

4. **Fechamento**: Ao finalizar a execução, o sistema salva os dados atualizados no arquivo CSV.

3. Descrição dos Módulos

3.1 – main.hs

O arquivo main.hs contém o ponto de entrada da aplicação. É responsável por iniciar o sistema, carregar o estoque e gerenciar as interações do usuário. A função principal do arquivo é a main, que coordena a execução do programa.

-menu: Exibe o menu principal com as opções para o usuário e redireciona para as funções correspondentes.

3.2 – produto.hs

Este módulo define a estrutura dos produtos, além de funções para manipulação de dados de produtos.

```
module Produto where

-- Declaração de atributos
type Id = Int
type Nome = String
type Preco = Float
type Marca = String
type Quant = Int
data Produto = Produto Id Nome Preco Marca Quant
```

3.3 – estoque.hs

Este módulo contém as funções responsáveis por gerenciar o estoque em si, como adicionar, remover, listar e buscar produtos.

Principais Funções:

- **registrarProduto**: Adiciona um novo produto ao estoque.

```
-- adiciona produto ao estoque
registrarProduto:: Produto -> IO()
registrarProduto (Produto id nome preco marca qtd) = do
    arq <- openFile "estoque.csv" AppendMode
    hPutStrLn arq (show id ++ "," ++ nome ++ "," ++ show preco ++ "," ++ marca ++
"," ++ show qtd)
    hClose arq
```

- **removerProduto**: Remove um produto existente do estoque com base em seu código.

```
-- Função para remover um produto do arquivo de estoque
removerProduto :: Int -> String -> IO ()
removerProduto idProduto caminho = do
    withFile caminho ReadMode (\arquivo -> do
        conteudo <- hGetContents arquivo
        let linhas = lines conteudo
        let novasLinhas = filter2 (\linha -> not (show idProduto `isPrefixOf`
linha))linhas
        -- let novasLinhas = filter2 (\linha -> not (isPrefixOf2 (show idProduto)
(splitBy ',' linha))) linhas

        -- essa parte é responsável por forçar a avaliação completa do arquivo
        -- Se usar somente o hclose arquivo ele dá um erro informando que a
operação é ilegal
        length conteudo `seq` return ()

        -- Sobrescrever o arquivo com as linhas restantes
        writeFile caminho (unlines novasLinhas)
        putStrLn $ "Produto com id " ++ show idProduto ++ " removido do estoque."
        )
```

- **atualizarProduto**: Procura o produto no arquivo utilizando a ‘modificarElemento’ e atualiza as informações, sobrescrevendo o arquivo.

```haskell
-- Atualiza Produto
type Colunas = [String]
type Valores = [String]

modificarElemento :: [String] -> String -> String -> [String]
modificarElemento  [] _ _ = []
modificarElemento  (l:ls) comparador novo
    | comparador `isPrefixOf` l = novo:modificarElemento ls comparador novo
    | otherwise = l:modificarElemento ls comparador novo

atualizarProduto :: Int -> String -> Produto -> IO ()
atualizarProduto idProduto caminho produto = do
    withFile caminho ReadMode (\arquivo -> do
        conteudo <- hGetContents arquivo
        let linhas = lines conteudo

        let novasLinhas = modificarElemento linhas (show idProduto)
(converterProdutoEmString produto)
        -- let novasLinhas = filter2 (\linha -> not (isPrefixOf2 (show idProduto)
(splitBy ',' linha))) linhas

        -- essa parte é responsável por forçar a avaliação completa do arquivo
        -- Se usar somente o hclose arquivo ele dá um erro informando que a
operação é ilegal
        length conteudo `seq` return ()

        -- Sobrescrever o arquivo com as linhas restantes
        writeFile caminho (unlines novasLinhas)
        putStrLn $ "Produto com id " ++ show idProduto ++ " atualizado no
estoque."
        )
```

3.4 – utils.hs

O módulo utils.hs contém funções auxiliares que são utilizadas em vários pontos da aplicação, como manipulação de strings e leitura/escrita de arquivos.

3.4.1 - splitBy :: Char -> String -> [String]

- **Propósito**: Esta função divide uma string em uma lista de strings usando um delimitador específico.
- **Funcionamento**: Utiliza a função break para separar a string até encontrar o delimitador. A primeira parte é adicionada à lista, e o restante é processado recursivamente.

3.4.2 - parseId :: String -> Maybe Int

- **Propósito**: Extrai e converte o ID de uma linha CSV, retornando-o como um Maybe Int.
- **Funcionamento**: Usa splitBy para dividir a linha pelo delimitador ,. Se o primeiro elemento for um número válido, ele será convertido para Int usando readMaybe.

4. Descrição do Arquivo

O arquivo estoque.csv armazena os produtos e suas quantidades.

Cada linha do arquivo representa um produto no estoque. Os campos são separados por vírgulas, onde:

- **codigo**: Código único do produto.
- **nome**: Nome do produto.
- **quantidade**: Quantidade disponível no estoque.

5. Funcionalidades Detalhadas

5.1 – Adicionar Produto

Permite ao usuário adicionar um novo produto ao estoque. O sistema solicita ao usuário o código, nome e quantidade do novo produto. O produto é então salvo no arquivo “estoque.csv”.

5.2 – Remover Produto

Permite ao usuário remover um produto do estoque, utilizando o código do produto como identificador. Após a remoção, o arquivo CSV é atualizado para refletir essa mudança.

5.3 – Listar Produtos

Exibe uma lista de todos os produtos atualmente no estoque, mostrando código, nome e quantidade.

5.4 – Buscar Produto

Permite ao usuário buscar um produto específico pelo código ou nome. Se o produto for encontrado, suas informações são exibidas.

6. Conclusão

Este sistema de controle de estoque oferece funcionalidades básicas de gerenciamento de produtos, utilizando Haskell e arquivos CSV. Ele pode ser facilmente expandido para incluir mais funcionalidades e melhorar a experiência do usuário.