Visão Geral do Projeto de Gestão de Lojas

Este documento visa fornecer uma compreensão abrangente do projeto de desenvolvimento de um sistema de gestão de lojas, incluindo seu propósito, estado atual, tecnologias utilizadas, estrutura de dados no Firestore e as regras de segurança aplicadas.

1. Objetivo Principal do Projeto

Estamos desenvolvendo um sistema web para gestão de lojas de varejo, focado em operações diárias como:

- Gestão de Produtos: Cadastro, estoque (incluindo variações de cor e tamanho), preços.
- Vendas (PDV): Interface para registro de vendas, seleção de produtos com variações, cálculo de total e finalização da venda.
- Gestão de Caixa: Abertura, fechamento, registro de suprimentos (entradas de dinheiro) e sangrias (saídas de dinheiro), e histórico de sessões de caixa.
- Gestão de Usuários e Lojas: Controle de acesso baseado em papéis (Admin, Gerente, Funcionário, Financeiro) e associação de usuários a lojas específicas.

2. Estado Atual do Desenvolvimento

Atualmente, o foco está em refinar as funcionalidades de Vendas e Caixa, especialmente no que diz respeito à persistência de dados no Firebase Firestore e às regras de segurança que governam o acesso a esses dados.

Módulos Implementados e seu Status:

- Autenticação: Baseada em Firebase Authentication. Usuários têm papéis (role) e um ID de loja (storeId) associado, definidos na coleção users.
- Página de Produtos: Permite visualizar e gerenciar produtos, incluindo suas variações.
- Página de Vendas (SalesPage.js):
    - Permite pesquisar e adicionar produtos ao carrinho (com suporte a variações de cor/tamanho).
    - Validações de estoque.
    - Finalização de vendas com diferentes métodos de pagamento.
    - Estado Atual: As vendas só são permitidas quando o caixa da loja está open. O botão de "Finalizar Venda" e a interação com produtos

são desabilitados se o caixa estiver closed. Os produtos são carregados corretamente, mas é crucial que haja produtos cadastrados no caminho correto do Firestore para que apareçam.

- Página de Caixa (CashRegisterPage.js):
    - Permite abrir e fechar a sessão de caixa da loja.
    - Registro de suprimentos e sangrias.
    - Exibição do saldo atual estimado.
    - Estado Atual: O status do caixa (open/closed) é lido em tempo real. O histórico de sessões de caixa anteriores é exibido.

Problemas Recentes Resolvidos (e o que aprendemos):

- Inconsistência de userStoreId: Resolvido garantindo que o userStoreId do AuthProvider seja usado consistentemente em todas as operações de Firestore relacionadas à loja.
- Produtos Não Aparecendo: A causa principal era a ausência de dados de produtos nos novos caminhos aninhados do Firestore. A solução envolveu o ajuste do código para os novos caminhos e a necessidade de adicionar dados manualmente para testes iniciais.
- Permissões de Caixa/Histórico: Erros de "Missing or insufficient permissions" foram corrigidos através de refinamentos nas regras de segurança do Firestore, garantindo que os papéis de usuário (admin, employee, finance, manager) tenham o acesso correto às coleções por storeId.
- Erros de Sintaxe em Regras do Firestore: Corrigido pontualmente a falta de if em expressões condicionais.

3. Tecnologias Utilizadas

- Frontend: React.js
    - Gerenciamento de Estado: React Hooks (useState, useEffect, useCallback, useMemo).
    - Contexto: AuthProvider e FirebaseContext (custom hooks useAuth e useFirebase) para gerenciar autenticação e instâncias do Firebase globalmente.
    - Componentes UI: Estilos inline definidos em objetos styles, ícones da biblioteca lucide-react.

- Backend / Banco de Dados: Google Firebase
    - Firestore: Banco de dados NoSQL para todas as informações da aplicação (usuários, produtos, vendas, caixa).
    - Authentication: Para gerenciamento de usuários.

4. Estrutura de Dados no Firestore (CRUCIAL)

A organização dos dados no Firestore é fundamental para a performance e segurança. A estrutura segue um padrão aninhado por appId e storeId para dados específicos da loja, e coleções de nível raiz para dados globais.

- users/{userId}:
    - Contém informações do perfil do usuário: email, role (e.g., 'admin', 'manager', 'employee', 'finance'), storeId (o ID da loja à qual o usuário está associado).
- product_attributes/{docId}:
    - Coleção de nível raiz para atributos globais, como:
        - categories (documento único contendo um array de categorias)
        - sizeGrades (documento único contendo grades de tamanho)
        - colors (documento único contendo um array de cores)
        - suppliers (documento único contendo um array de fornecedores)
- stores/{storeId}:
    - Coleção de nível raiz para metadados das lojas. Cada documento storeId contém detalhes como name da loja.
- artifacts/{appId}/stores/{storeId}/products/{productId}:
    - Produtos específicos de cada loja. appId é uma variável de ambiente fornecida pelo Canvas/ambiente de execução.
    - Cada documento de produto inclui: name, code, barcode, price, costPrice, totalQuantity, e um array variations (onde cada variação tem color, size e quantity).
- artifacts/{appId}/stores/{storeId}/sales/{saleId}:
    - Transações de vendas específicas de cada loja.

    - Cada documento de venda inclui: date (Timestamp), totalAmount, paymentMethod, clientName, clientCpf, storeId, storeName, sellerId, sellerEmail, e um array items (detalhes dos produtos vendidos).
- cash_register_sessions/{storeId}:
    - O status da sessão de caixa ATUAL para uma loja. É um documento único para cada storeId.
    - Contém: status ('open' ou 'closed'), openedAt, openedBy, openingBalance, currentCashCount, supplies (array), outflows (array), storeId, storeName.
    - Quando o caixa é fechado, este documento é "resetado" (campos como openedAt se tornam null, status para closed), e os dados da sessão são movidos para o histórico.
- artifacts/{appId}/stores/{storeId}/cash_register_history/{historyId}:
    - Histórico de todas as sessões de caixa fechadas para uma loja.
    - Cada documento nesta coleção representa uma sessão de caixa *finalizada*.
    - Contém todos os campos da sessão ativa (status: 'closed', openedAt, closedAt, closingBalance, expectedCash, difference, dailySalesSummary, etc.).

5. Regras de Segurança do Firebase Firestore

As regras do Firestore são cruciais para controlar o acesso aos dados com base na autenticação do usuário (request.auth), seu papel (role) e o ID da loja (storeId) ao qual ele pertence.

A função getUserData(userId) é usada para buscar o perfil do usuário e suas permissões.

Aqui estão as regras de segurança mais recentes e revisadas que estão em uso no projeto. Elas são vitais para a operação correta do aplicativo, especialmente após as mudanças na estrutura de dados:

```
rules_version = '2';

service cloud.firestore {

  match /databases/{database}/documents {
```

```
  // Função auxiliar para obter os dados do usuário (role e storeId)
  function getUserData(userId) {
    return get(/databases/$(database)/documents/users/$(userId)).data;
  }

  // 1. Regras para a coleção 'users' (contém o perfil do usuário, incluindo role e storeId)
  // Cada usuário só pode ler seu próprio perfil. Apenas administradores podem atualizar/deletar outros perfis.
  match /users/{userId} {
    allow read: if request.auth != null && request.auth.uid == userId;
    allow create: if request.auth != null && request.auth.uid == userId; // Usuário pode criar seu próprio perfil ao se registrar
    allow update: if request.auth != null && (request.auth.uid == userId || getUserData(request.auth.uid).role == 'admin');
    allow delete: if request.auth != null && getUserData(request.auth.uid).role == 'admin';
  }

  // 2. Regras para a coleção 'product_attributes' (categorias, grades de tamanho, cores, fornecedores)
  // AGORA NO NÍVEL RAIZ
  match /product_attributes/{docId} {
    allow read, write: if request.auth != null && getUserData(request.auth.uid).role == 'admin';
  }

  // 3. Regras para a coleção 'stores' (lojas - coleção de nível raiz)
  // Admin, Employee, Manager, Finance podem ler lojas
  match /stores/{storeId} {
```

```
      allow read: if request.auth != null && (getUserData(request.auth.uid).role ==
'admin' || getUserData(request.auth.uid).role == 'employee' ||
getUserData(request.auth.uid).role == 'manager' ||
getUserData(request.auth.uid).role == 'finance');

      // Apenas Admin pode criar, atualizar ou deletar lojas

      allow write: if request.auth != null && getUserData(request.auth.uid).role ==
'admin';

    }


    // 4. Regras para a coleção 'products' (produtos)

    // Admin, Employee, Manager podem ler produtos de SUA loja. Admin também
pode escrever.

    match /products/{productId} {

      allow read: if request.auth != null && request.auth.uid != null &&

                  getUserData(request.auth.uid).storeId in resource.data.storeIds && // <--
CORREÇÃO AQUI

                  (getUserData(request.auth.uid).role == 'admin' ||

                   getUserData(request.auth.uid).role == 'employee' ||

                   getUserData(request.auth.uid).role == 'manager');

      allow create, update, delete: if request.auth != null &&
getUserData(request.auth.uid).role == 'admin';

    }


    // 5. Regras para a coleção 'sales' (vendas - ANINHADA POR LOJA)

    // Esta coleção é aninhada em artifacts/{appId}/stores/{storeId}/sales

    match /artifacts/{appId}/stores/{storeId}/sales/{saleId} {

      // Admin, Employee, Manager e Finance podem ler vendas DA SUA loja

      allow read: if request.auth != null && getUserData(request.auth.uid).storeId ==
storeId && (getUserData(request.auth.uid).role == 'admin' ||
getUserData(request.auth.uid).role == 'employee' ||
```

```
getUserData(request.auth.uid).role == 'manager' ||
getUserData(request.auth.uid).role == 'finance');

    // Admin e Employee podem criar e atualizar vendas DA SUA loja

    allow write: if request.auth != null && getUserData(request.auth.uid).storeId ==
storeId && (getUserData(request.auth.uid).role == 'admin' ||
getUserData(request.auth.uid).role == 'employee');

  }


  // 6. REGRAS ESPECIFICAS PARA A COLECAO ANINHADA
'cash_register_sessions'

  // Esta colecao agora armazena TANTO a sessao atual (status 'open') quanto o
historico (status 'closed')

  match /stores/{storeId}/cash_register_sessions/{sessionId}{

    // Leitura: Admin, Employee, Manager e Finance podem ler qualquer sessao de
caixa DA SUA loja.

    allow read: if request.auth != null &&

                getUserData(request.auth.uid).storeId == storeId &&

                (getUserData(request.auth.uid).role == 'admin' ||

                 getUserData(request.auth.uid).role == 'employee' ||

                 getUserData(request.auth.uid).role == 'manager' ||

                 getUserData(request.auth.uid).role == 'finance');


    // Escrita (criar, atualizar): Admin e Employee podem criar e atualizar sessoes
de caixa DA SUA loja.

    allow write: if request.auth != null &&

                  getUserData(request.auth.uid).storeId == storeId &&

                  (getUserData(request.auth.uid).role == 'admin' ||

                   getUserData(request.auth.uid).role == 'employee');

  }
```

```
    // 7. Regras para 'bankAccounts' e 'transactions' (se existirem e forem de nível raiz)
    // Apenas Admin e Finance podem ler e escrever nessas coleções.
    match /bankAccounts/{bankAccountId} {
      allow read, write: if request.auth != null && (getUserData(request.auth.uid).role == 'admin' || getUserData(request.auth.uid).role == 'finance');
    }

    match /transactions/{transactionId} {
      allow read, write: if request.auth != null && (getUserData(request.auth.uid).role == 'admin' || getUserData(request.auth.uid).role == 'finance');
    }
  }
}
```

6. Como Interagir Comigo (Esta IA) para Continuidade

Para garantir uma colaboração eficaz, por favor, siga estas diretrizes ao interagir:

- Forneça Contexto Claro: Sempre que possível, descreva o problema ou a funcionalidade desejada em detalhes.
- Capture Erros Completos: Se houver erros, copie e cole a mensagem de erro completa do console do navegador. As linhas de código (ex: SalesPage.js:123) são extremamente úteis.
- Screenshots são Essenciais: Imagens da interface do usuário e do console do navegador (especialmente a aba "Console" e "Network" se relevante para chamadas de API) são inestimáveis para depuração.
- Compartilhe Código Existente: Se você modificou um arquivo, ou se o problema parece estar em um arquivo específico, me mencione o nome do arquivo (SalesPage.js, CashRegisterPage.js, etc.). Se o arquivo já foi "carregado" na nossa conversa, eu posso acessá-lo pelo nome.
- Expectativas de Output:
    - Código: Sempre será fornecido em um bloco code immersive, completo e autocontido, com comentários detalhados.

    - Explicações/Discussões: Serão em formato de texto/Markdown, como este documento.
    - Iteração: O processo é iterativo. Pode ser necessário várias rodadas de ajuste e teste para resolver problemas complexos ou adicionar novas funcionalidades.
- Evite Alertas Nativos: No código, evite alert() ou confirm(). Prefira usar modais ou mensagens na interface do usuário, pois alert() pode não funcionar como esperado no ambiente de visualização.
- Foco em Componentes Autocontidos: Os componentes React que eu forneço são pensados para serem autocontidos e fáceis de integrar.

Espero que este resumo ajude a outra IA (e a você!) a entender o projeto e a continuar o desenvolvimento sem problemas. Estou à disposição para qualquer dúvida!