PdC
RESTAURANT

PEDACINHO DO CÉU

Restaurante & Gastronomia

GESTÃOZE SYSTEM

Sistema de Gestão de Estoque Inteligente

Manual Técnico e Guia de Utilização

Especificações Técnicas

Frontend:	Vue.js 3.5.21 + TypeScript
Backend:	Supabase (PostgreSQL)
UI Framework:	Components Customizados
Charts:	Chart.js 4.5.0
AI Integration:	Google Gemini AI
Versão:	1.0.0

Desenvolvido especialmente para
Gestão de Estoque em Restaurantes

25 de setembro de 2025

Conteúdo

1 Introdução **4**
1.1 Visão Geral do Sistema . . . 4
1.2 Tecnologias Utilizadas . . . 4
1.2.1 Frontend . . . 4
1.2.2 Backend e Banco de Dados . . . 4
1.2.3 Ferramentas Auxiliares . . . 4
1.3 Arquitetura do Sistema . . . 5
1.3.1 Padrão de Arquitetura . . . 5
1.3.2 Estrutura de Diretórios . . . 5

2 Configuração e Instalação **5**
2.1 Requisitos do Sistema . . . 5
2.1.1 Requisitos de Software . . . 5
2.1.2 Configuração do Ambiente . . . 6
2.2 Instalação . . . 6

3 Banco de Dados e Supabase **6**
3.1 Arquitetura do Banco de Dados . . . 6
3.1.1 Tabelas Principais . . . 6
3.1.2 Relacionamentos e Constraints . . . 7
3.2 Configuração do Supabase . . . 7
3.2.1 Row Level Security (RLS) . . . 7
3.2.2 Triggers e Funções . . . 8

4 Sistema de Rotas **8**
4.1 Configuração de Roteamento . . . 8
4.1.1 Estrutura de Rotas . . . 8
4.1.2 Guards de Navegação . . . 9

5 Guia de Utilização das Rotas **9**
5.1 Rota /login - Autenticação . . . 9
5.1.1 Funcionalidades . . . 9
5.1.2 Uso da Rota . . . 9
5.1.3 Serviços Utilizados . . . 10
5.2 Rota /dashboard - Painel Principal . . . 10
5.2.1 Funcionalidades . . . 10
5.2.2 Estrutura da Página . . . 10
5.2.3 Serviços Utilizados . . . 10
5.3 Rota /inventory - Gestão de Estoque . . . 11
5.3.1 Funcionalidades Principais . . . 11
5.3.2 Operações de Produto . . . 11
5.3.3 Filtros e Busca . . . 12
5.3.4 Serviços Utilizados . . . 12
5.4 Rota /suppliers - Gestão de Fornecedores . . . 12
5.4.1 Funcionalidades . . . 12
5.4.2 Cadastro de Fornecedor . . . 12
5.4.3 Serviços Utilizados . . . 12

5.5 Rota /menu - Gestão de Cardápio . . . 13
5.5.1 Funcionalidades . . . 13
5.5.2 Cadastro de Prato . . . 13
5.5.3 Planejamento de Cardápio . . . 13
5.5.4 Serviços Utilizados . . . 13
5.6 Rota /reports - Relatórios e Análises . . . 14
5.6.1 Funcionalidades Avançadas . . . 14
5.6.2 Tipos de Relatórios . . . 14
5.6.3 Gráficos e Visualizações . . . 14
5.6.4 Exportação de Dados . . . 15
5.6.5 Serviços Utilizados . . . 15
5.7 Rota /ai - Inteligência Artificial . . . 15
5.7.1 Funcionalidades de IA . . . 15
5.7.2 Análise Preditiva . . . 16
5.7.3 Otimização de Estoque . . . 16
5.7.4 Serviços Utilizados . . . 16
5.8 Rota /logs - Auditoria e Logs . . . 16
5.8.1 Funcionalidades de Auditoria . . . 16
5.8.2 Tipos de Log . . . 17
5.8.3 Filtros de Log . . . 17
5.8.4 Serviços Utilizados . . . 17
5.9 Rota /settings - Configurações . . . 17
5.9.1 Configurações Disponíveis . . . 17
5.9.2 Categorias de Configuração . . . 17
5.9.3 Serviços Utilizados . . . 18
5.10 Rota /profile - Perfil do Usuário . . . 18
5.10.1 Funcionalidades . . . 18
5.10.2 Configurações de Perfil . . . 18
5.10.3 Segurança . . . 19
5.10.4 Serviços Utilizados . . . 19

6 Arquitetura de Serviços **19**
6.1 Camada de Serviços . . . 19
6.1.1 Estrutura de Serviços . . . 19
6.2 Padrões de Implementação . . . 19
6.2.1 Service Layer Pattern . . . 19
6.2.2 Error Handling . . . 20
6.3 Integração com Supabase . . . 20
6.3.1 Cliente Supabase . . . 20
6.3.2 Operações de Banco de Dados . . . 21

7 Funcionalidades Avançadas **21**
7.1 Sistema de Analytics Avançado . . . 21
7.1.1 Análises Estatísticas . . . 21
7.1.2 Business Intelligence . . . 22
7.2 Inteligência Artificial . . . 22
7.2.1 Análise Preditiva . . . 22
7.2.2 Integração com APIs de IA . . . 23
7.3 Sistema de Relatórios Avançados . . . 23

7.3.1 Exportação Multi-Formato . . . 23
7.3.2 Customização de Relatórios . . . 24
7.4 Visualizações Avançadas . . . 24
7.4.1 Tipos de Gráfico Suportados . . . 24
7.4.2 Configuração de Gráficos . . . 24

8 Segurança e Performance **25**
8.1 Medidas de Segurança . . . 25
8.1.1 Autenticação e Autorização . . . 25
8.1.2 Proteção de Dados . . . 26
8.1.3 Políticas RLS . . . 26
8.2 Otimizações de Performance . . . 26
8.2.1 Frontend . . . 26
8.2.2 Backend/Database . . . 26
8.2.3 Monitoramento . . . 27

9 Deployment e Manutenção **27**
9.1 Processo de Build . . . 27
9.1.1 Scripts de Build . . . 27
9.1.2 Configuração de Build . . . 28
9.2 Deployment em Produção . . . 28
9.2.1 Vercel (Recomendado) . . . 28
9.2.2 Netlify . . . 29
9.3 Monitoramento e Logs . . . 29
9.3.1 Monitoramento de Aplicação . . . 29
9.3.2 Logs Estruturados . . . 29
9.3.3 Logs de Debug . . . 32
9.4 FAQ - Perguntas Frequentes . . . 32
9.4.1 Funcionalidades . . . 32
9.4.2 Técnicas . . . 33

10 API e Integrações **33**
10.1 API Supabase . . . 33
10.1.1 Endpoints Principais . . . 33
10.1.2 Autenticação API . . . 33
10.2 Integrações Externas . . . 33
10.2.1 Google Gemini AI . . . 33
10.2.2 Webhooks . . . 34

11 Conclusão **35**
11.1 Resumo das Capacidades . . . 35
11.2 Roadmap de Desenvolvimento . . . 35
11.2.1 Próximas Funcionalidades . . . 35
11.3 Suporte e Comunidade . . . 35
11.3.1 Recursos de Apoio . . . 35
11.3.2 Contribuição . . . 36

1 Introdução

1.1 Visão Geral do Sistema

O **GestãoZe System** é uma aplicação web moderna de gestão de estoque desenvolvida com tecnologias de ponta para atender às necessidades de pequenas e médias empresas do setor alimentício. O sistema combina funcionalidades tradicionais de controle de inventário com recursos avançados de inteligência artificial, análise preditiva e relatórios automatizados.

Características Principais

- Interface moderna e responsiva construída em Vue.js 3
- Backend robusco com Supabase (PostgreSQL)
- Análise de dados avançada com inteligência artificial
- Relatórios dinâmicos e exportação em múltiplos formatos
- Sistema de autenticação e autorização seguro
- Gerenciamento completo de produtos, fornecedores e cardápios
- Análise preditiva de demanda e otimização de estoque

1.2 Tecnologias Utilizadas

1.2.1 Frontend

- **Vue.js 3.5.21**: Framework JavaScript progressivo para construção da interface
- **TypeScript 5.2.2**: Superset do JavaScript com tipagem estática
- **Pinia 2.3.1**: Gerenciamento de estado reativo
- **Vue Router 4.5.1**: Roteamento SPA (Single Page Application)
- **Chart.js 4.5.0**: Biblioteca para gráficos e visualizações
- **Lucide Vue Next**: Biblioteca de ícones moderna

1.2.2 Backend e Banco de Dados

- **Supabase**: Backend-as-a-Service com PostgreSQL
- **PostgreSQL**: Banco de dados relacional robusto
- **Row Level Security (RLS)**: Segurança em nível de linha
- **Real-time subscriptions**: Atualizações em tempo real

1.2.3 Ferramentas Auxiliares

- **Vite**: Build tool e servidor de desenvolvimento
- **Axios**: Cliente HTTP para requisições
- **Date-fns**: Manipulação de datas

- **jsPDF**: Geração de relatórios PDF
- **html2canvas**: Captura de elementos HTML
- **XLSX**: Manipulação de planilhas Excel

1.3 Arquitetura do Sistema

1.3.1 Padrão de Arquitetura

O sistema segue uma arquitetura de **Single Page Application (SPA)** baseada em componentes, implementando os seguintes padrões:

- **Model-View-ViewModel (MVVM)**: Implementado através do Vue.js
- **Service Layer Pattern**: Separação da lógica de negócio em serviços
- **Store Pattern**: Gerenciamento de estado centralizado com Pinia
- **Component-Based Architecture**: Interface modular e reutilizável

1.3.2 Estrutura de Diretórios

```
src/
          components/          # Componentes reutiliz veis
          views/               # P ginas da aplica  o
          services/            # Camada de servi os
          stores/              # Gerenciamento de estado (Pinia)
          router/              # Configura  o de rotas
          config/              # Configura  es (Supabase, etc.)
          types/               # Defini  es de tipos TypeScript
          utils/               # Fun  es utilit rias
          assets/              # Recursos est ticos
```

Listing 1: Estrutura do Projeto

2 Configuração e Instalação

2.1 Requisitos do Sistema

2.1.1 Requisitos de Software

- **Node.js**: Versão 18.x ou superior
- **NPM**: Versão 9.x ou superior
- **Navegador**: Chrome 90+, Firefox 88+, Safari 14+, Edge 90+

2.1.2 Configuração do Ambiente

Variáveis de Ambiente Necessárias

Crie um arquivo `.env` na raiz do projeto com as seguintes variáveis:

```
VITE_SUPABASE_URL=sua_url_do_supabase
VITE_SUPABASE_ANON_KEY=sua_chave_anonima_supabase
VITE_GEMINI_API_KEY=sua_chave_api_gemini
```

2.2 Instalação

1. **Clone o repositório**:

```
git clone <repository_url>
cd gestaozesystem-web
```

2. **Instale as dependências**:

```
npm install
```

3. **Configure as variáveis de ambiente**:

```
cp .env.example .env
# Edite o arquivo .env com suas credenciais
```

4. **Execute em modo de desenvolvimento**:

```
npm run dev
```

5. **Build para produção**:

```
npm run build
```

3 Banco de Dados e Supabase

3.1 Arquitetura do Banco de Dados

O sistema utiliza PostgreSQL através do Supabase, implementando uma estrutura relacional otimizada para gestão de estoque de estabelecimentos alimentícios.

3.1.1 Tabelas Principais

Tabela	Propósito	Campos Principais
`admin_users`	Usuários do sistema	id, email, password_hash, role, created_at
`produtos`	Produtos em estoque	id, nome, categoria, current_stock, min_stock, max_stock, unit_cost, sale_price, supplier_id

Tabela	Propósito	Campos Principais
`categorias`	Categorias de produtos	id, nome, description, color_tag
`movements`	Movimentações de estoque	id, product_id, type, quantity, reason, user_id, created_at
`menu_items`	Itens do cardápio	id, nome, category, price, description, is_available
`menu_item_ingredientes`	Ingredientes dos pratos	id, menu_item_id, product_id, quantity_needed
`menu_diario`	Cardápio diário	id, date, menu_item_id, quantity_planned
`planejamento_semanal`	Planejamento semanal	id, week_start, menu_item_id, quantities_by_day
`suppliers`	Fornecedores	id, name, contact_info, email, phone, address
`reports`	Relatórios salvos	id, name, type, data, user_id, created_at
`logs`	Logs do sistema	id, action, details, user_id, ip_address, created_at
`app_settings`	Configurações	key, value, description, updated_at

3.1.2 Relacionamentos e Constraints

- **produtos.supplier_id** → **suppliers.id**: Relacionamento many-to-one
- **movements.product_id** → **produtos.id**: Relacionamento many-to-one
- **menu_item_ingredientes.product_id** → **produtos.id**: Relacionamento many-to-one
- **menu_item_ingredientes.menu_item_id** → **menu_items.id**: Relacionamento many-to-one

3.2 Configuração do Supabase

3.2.1 Row Level Security (RLS)

O sistema implementa políticas de segurança em nível de linha para garantir o acesso controlado aos dados:

```
-- Pol tica para tabela de produtos
CREATE POLICY "Users can view products" ON produtos
    FOR SELECT USING (auth.role() = 'authenticated');

CREATE POLICY "Admin can insert products" ON produtos
    FOR INSERT WITH CHECK (auth.jwt() ->> 'role' = 'admin');

CREATE POLICY "Admin can update products" ON produtos
    FOR UPDATE USING (auth.jwt() ->> 'role' = 'admin');
```

Listing 2: Exemplo de Política RLS

3.2.2 Triggers e Funções

```sql
-- Função para atualizar estoque após movimentação
CREATE OR REPLACE FUNCTION update_product_stock()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.type = 'entrada' THEN
        UPDATE produtos
        SET current_stock = current_stock + NEW.quantity
        WHERE id = NEW.product_id;
    ELSIF NEW.type = 'saida' THEN
        UPDATE produtos
        SET current_stock = current_stock - NEW.quantity
        WHERE id = NEW.product_id;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Trigger
CREATE TRIGGER trigger_update_stock
    AFTER INSERT ON movements
    FOR EACH ROW
    EXECUTE FUNCTION update_product_stock();
```

Listing 3: Trigger para Atualização de Estoque

4 Sistema de Rotas

4.1 Configuração de Roteamento

O sistema utiliza Vue Router 4 com roteamento baseado em história (History Mode), proporcionando URLs limpas e navegação SPA otimizada.

4.1.1 Estrutura de Rotas

Rota	Componente	Autenticação	Descrição
/	-	-	Redirecionamento para /dashboard
/login	LoginView	Convidado	Página de autenticação
/dashboard	DashboardView	Obrigatória	Painel principal com métricas
/inventory	InventoryView	Obrigatória	Gestão de produtos e estoque
/ai	AIView	Obrigatória	Análises com inteligência artificial
/reports	ReportsView	Obrigatória	Relatórios e análises avançadas
/suppliers	SuppliersView	Obrigatória	Gestão de fornecedores

Rota	Componente	Autenticação	Descrição
/menu	MenuView	Obrigatória	Gestão de cardápio e planejamento
/logs	LogsView	Obrigatória	Auditoria e logs do sistema
/settings	SettingsView	Obrigatória	Configurações do sistema
/profile	ProfileView	Obrigatória	Perfil do usuário
/about	AboutView	Obrigatória	Informações sobre o sistema

4.1.2 Guards de Navegação

O sistema implementa guards de navegação para controle de acesso:

```
router.beforeEach(async (to) => {
  const authStore = useAuthStore()

  // Verificar sess o armazenada
  const stored = localStorage.getItem('userSession')
  if (stored && !authStore.user) {
    authStore.user = JSON.parse(stored)
  }

  const isAuthenticated = authStore.isAuthenticated

  if (to.meta.requiresAuth && !isAuthenticated) {
    return '/login'
  }

  if (to.meta.requiresGuest && isAuthenticated) {
    return '/dashboard'
  }
})
```

Listing 4: Guard de Autenticação

5 Guia de Utilização das Rotas

5.1 Rota /login - Autenticação

5.1.1 Funcionalidades

- Autenticação de usuários via email e senha
- Validação de credenciais com Supabase Auth
- Persistência de sessão local
- Redirecionamento automático após login

5.1.2 Uso da Rota

1. Acesse `http://localhost:5173/login`

2. Insira credenciais válidas (email e senha)
3. Clique em "Entrar"
4. Sistema redireciona automaticamente para `/dashboard`

Segurança

As senhas são criptografadas pelo Supabase Auth. Nunca são armazenadas em texto plano.

5.1.3 Serviços Utilizados

- `authService.ts`: Gerencia autenticação
- `auth.ts` (store): Estado de autenticação global

5.2 Rota /dashboard - Painel Principal

5.2.1 Funcionalidades

- Visão geral de métricas importantes
- Gráficos de vendas e movimentação
- Alertas de estoque baixo
- Resumo de produtos críticos
- Cards informativos com KPIs

5.2.2 Estrutura da Página

1. **Header**: Métricas rápidas (total de produtos, valor do estoque)
2. **Gráficos**: Visualizações de dados de vendas e movimentação
3. **Alertas**: Produtos com estoque baixo ou zerado
4. **Atividades Recentes**: Últimas movimentações

5.2.3 Serviços Utilizados

- `productService.ts`: Dados de produtos
- `salesService.ts`: Dados de vendas
- `reportsService.ts`: Métricas e análises

5.3 Rota /inventory - Gestão de Estoque

5.3.1 Funcionalidades Principais

- CRUD completo de produtos
- Controle de movimentação (entrada/saída)
- Gestão de categorias
- Filtros e busca avançada
- Importação/exportação de dados
- Código de barras e QR Code

5.3.2 Operações de Produto

Cadastro de Produto

1. Clique em "Novo Produto"
2. Preencha os campos obrigatórios:
    - Nome do produto
    - Categoria
    - Estoque atual
    - Estoque mínimo
    - Preço de custo
    - Preço de venda
    - Fornecedor
3. Clique em "Salvar"

Movimentação de Estoque

1. Selecione o produto na lista
2. Clique em "Movimentar"
3. Escolha o tipo (Entrada/Saída)
4. Informe a quantidade
5. Adicione observações (opcional)
6. Confirme a movimentação

5.3.3 Filtros e Busca

- **Busca por nome**: Campo de pesquisa em tempo real
- **Filtro por categoria**: Dropdown com categorias disponíveis
- **Filtro por status**: Estoque normal, baixo, zerado
- **Filtro por fornecedor**: Produtos de fornecedor específico

5.3.4 Serviços Utilizados

- `productService.ts`: Operações CRUD de produtos
- `movementService.ts`: Controle de movimentações
- `categoryService.ts`: Gestão de categorias

5.4 Rota /suppliers - Gestão de Fornecedores

5.4.1 Funcionalidades

- Cadastro completo de fornecedores
- Controle de contatos e informações comerciais
- Histórico de compras por fornecedor
- Avaliação de performance
- Gestão de contratos e prazos

5.4.2 Cadastro de Fornecedor

1. Clique em "Novo Fornecedor"
2. Preencha as informações:
    - Razão social
    - Nome fantasia
    - CNPJ/CPF
    - Endereço completo
    - Telefone e email
    - Informações bancárias
    - Condições de pagamento
3. Salve o cadastro

5.4.3 Serviços Utilizados

- `suppliersService.ts`: Gestão de fornecedores
- `purchaseService.ts`: Histórico de compras

5.5 Rota /menu - Gestão de Cardápio

5.5.1 Funcionalidades

- Criação e gestão de pratos
- Definição de ingredientes e quantidades
- Cálculo automático de custos
- Planejamento de cardápio diário/semanal
- Análise de rentabilidade por prato
- Controle de sazonalidade

5.5.2 Cadastro de Prato

1. Acesse "Novo Prato"
2. Defina nome, categoria e preço
3. Adicione ingredientes:
    - Selecione produto do estoque
    - Defina quantidade necessária
    - Sistema calcula custo automaticamente
4. Configure informações nutricionais (opcional)
5. Salve o prato

5.5.3 Planejamento de Cardápio

1. Selecione período (dia/semana/mês)
2. Escolha pratos disponíveis
3. Defina quantidades estimadas
4. Sistema verifica disponibilidade de ingredientes
5. Confirme o planejamento

5.5.4 Serviços Utilizados

- `menuService.ts`: Gestão de cardápio
- `recipeService.ts`: Receitas e ingredientes
- `planningService.ts`: Planejamento de produção

5.6 Rota /reports - Relatórios e Análises

5.6.1 Funcionalidades Avançadas

- Relatórios financeiros detalhados
- Análises de vendas e margem
- Relatórios de movimentação
- Análise ABC de produtos
- Relatórios de fornecedores
- Análise preditiva com IA
- Exportação em múltiplos formatos (PDF, Excel, CSV)

5.6.2 Tipos de Relatórios

Relatórios Padrão

- **Estoque Atual**: Lista completa com valores
- **Movimentações**: Histórico de entradas/saídas
- **Produtos em Falta**: Itens com estoque zerado
- **Estoque Baixo**: Produtos abaixo do mínimo
- **Análise de Vendas**: Performance por período

Relatórios Avançados

- **Análise ABC**: Classificação por importância
- **Giro de Estoque**: Rotatividade de produtos
- **Margem de Contribuição**: Análise financeira
- **Sazonalidade**: Padrões de consumo
- **Previsão de Demanda**: IA preditiva

5.6.3 Gráficos e Visualizações

- Gráficos de linha para tendências
- Gráficos de barras para comparações
- Gráficos de pizza para distribuições
- Mapas de calor para correlações
- Gráficos radar para análises multi-dimensionais

5.6.4 Exportação de Dados

1. Selecione o relatório desejado
2. Configure filtros e período
3. Escolha formato de exportação:
    - PDF com gráficos
    - Excel com dados e tabelas dinâmicas
    - CSV para análise externa
    - PowerBI para dashboards
4. Clique em "Exportar"

5.6.5 Serviços Utilizados

- `reportsService.ts`: Geração de relatórios
- `advancedAnalyticsService.ts`: Análises estatísticas
- `advancedChartsService.ts`: Visualizações avançadas
- `advancedExportService.ts`: Exportação múltiplos formatos
- `aiAnalyticsService.ts`: Análises com IA
- `predictiveAnalyticsService.ts`: Análise preditiva

5.7 Rota /ai - Inteligência Artificial

5.7.1 Funcionalidades de IA

- Análise preditiva de demanda
- Otimização de compras
- Detecção de anomalias
- Sugestões inteligentes de precificação
- Análise de padrões de consumo
- Insights automatizados
- Chatbot para consultas

5.7.2 Análise Preditiva

1. Selecione produtos para análise
2. Defina horizonte temporal
3. Configure parâmetros:
    - Sazonalidade
    - Tendências históricas
    - Fatores externos
4. Execute análise
5. Visualize previsões e recomendações

5.7.3 Otimização de Estoque

O sistema analisa histórico e sugere:

- Pontos ótimos de reposição
- Quantidades econômicas de compra
- Produtos com potencial de descontinuação
- Oportunidades de cross-selling

5.7.4 Serviços Utilizados

- `aiService.ts`: Integração com APIs de IA
- `aiAnalyticsService.ts`: Análises inteligentes
- `predictiveAnalyticsService.ts`: Modelos preditivos

5.8 Rota /logs - Auditoria e Logs

5.8.1 Funcionalidades de Auditoria

- Rastreamento completo de ações
- Logs de acesso e segurança
- Histórico de alterações
- Análise de performance
- Monitoramento de erros
- Relatórios de conformidade

5.8.2 Tipos de Log

- **Sistema**: Inicialização, erros, performance
- **Usuário**: Login, logout, ações realizadas
- **Estoque**: Movimentações, alterações
- **Segurança**: Tentativas de acesso, falhas
- **API**: Requisições, respostas, erros

5.8.3 Filtros de Log

- Por usuário
- Por período
- Por tipo de ação
- Por nível de severidade
- Por módulo do sistema

5.8.4 Serviços Utilizados

- `logsService.ts`: Gestão de logs
- `auditService.ts`: Trilha de auditoria

5.9 Rota /settings - Configurações

5.9.1 Configurações Disponíveis

- Parâmetros gerais do sistema
- Configurações de estoque (pontos de reposição)
- Configurações financeiras (moedas, impostos)
- Configurações de notificação
- Integrações externas
- Backup e restauração

5.9.2 Categorias de Configuração

Sistema Geral

- Nome da empresa
- Logo e identidade visual
- Fuso horário
- Idioma padrão
- Formato de data/hora

Estoque

- Política de estoque mínimo global
- Configurações de alerta
- Métodos de valoração (FIFO, LIFO, Médio)
- Categorias padrão

Financeiro

- Moeda padrão
- Configurações de impostos
- Margem de lucro padrão
- Formas de pagamento

5.9.3 Serviços Utilizados

- `settingsService.ts`: Gestão de configurações
- `configService.ts`: Parâmetros do sistema

5.10 Rota /profile - Perfil do Usuário

5.10.1 Funcionalidades

- Edição de dados pessoais
- Alteração de senha
- Configurações de notificação
- Histórico de atividades
- Preferências de interface
- Configurações de segurança (2FA)

5.10.2 Configurações de Perfil

- Nome completo
- Email de contato
- Telefone
- Foto de perfil
- Cargo/função
- Departamento

5.10.3 Segurança

- Alteração de senha
- Autenticação de dois fatores
- Sessões ativas
- Dispositivos autorizados

5.10.4 Serviços Utilizados

- `profileService.ts`: Gestão de perfil
- `authService.ts`: Segurança e autenticação

6 Arquitetura de Serviços

6.1 Camada de Serviços

O sistema implementa uma arquitetura em camadas com serviços especializados para cada domínio de negócio, garantindo separação de responsabilidades e facilidade de manutenção.

6.1.1 Estrutura de Serviços

Serviço	Responsabilidades
`authService`	Autenticação, autorização, gestão de sessões
`productService`	CRUD de produtos, controle de estoque, categorias
`suppliersService`	Gestão de fornecedores, contratos, avaliações
`menuService`	Cardápio, receitas, ingredientes, planejamento
`salesService`	Vendas, faturamento, análise de performance
`reportsService`	Relatórios padrão, métricas, KPIs
`advancedAnalyticsService`	Análises estatísticas avançadas, correlações
`aiAnalyticsService`	Análises com inteligência artificial
`predictiveAnalyticsService`	Modelos preditivos, forecasting
`advancedChartsService`	Visualizações avançadas, gráficos customizados
`advancedExportService`	Exportação em múltiplos formatos
`logsService`	Auditoria, logs, rastreamento
`settingsService`	Configurações do sistema
`profileService`	Perfil do usuário, preferências

6.2 Padrões de Implementação

6.2.1 Service Layer Pattern

Cada serviço segue o padrão Service Layer, implementando:

```
export class BaseService {
  protected supabase = supabase

  // Opera  es CRUD padr o
  async getAll(): Promise<T[]>
  async getById(id: string): Promise<T>
  async create(data: Partial<T>): Promise<T>
  async update(id: string, data: Partial<T>): Promise<T>
  async delete(id: string): Promise<void>

  // Opera  es espec ficas do dom nio
  // ...
}
```

Listing 5: Estrutura Base de Serviço

6.2.2 Error Handling

Sistema padronizado de tratamento de erros:

```
export interface ServiceError {
  code: string
  message: string
  details?: any
  timestamp: Date
}

export class ServiceException extends Error {
  constructor(
    public code: string,
    public message: string,
    public details?: any
  ) {
    super(message)
  }
}
```

Listing 6: Tratamento de Erros

6.3 Integração com Supabase

6.3.1 Cliente Supabase

```
import { createClient } from '@supabase/supabase-js'

const supabaseUrl = import.meta.env.VITE_SUPABASE_URL
const supabaseAnonKey = import.meta.env.VITE_SUPABASE_ANON_KEY

export const supabase = createClient(supabaseUrl,
   supabaseAnonKey, {
  auth: {
```

```
    autoRefreshToken: true,
    persistSession: true,
    detectSessionInUrl: false
  }
})
```

Listing 7: Configuração Supabase

6.3.2 Operações de Banco de Dados

```
export class ProductService {
  async getAllProducts(): Promise<Product[]> {
    const { data, error } = await supabase
      .from('produtos')
      .select(`
        *,
        categoria:categorias(*),
        fornecedor:suppliers(*)
      `)
      .order('nome')

    if (error) throw new ServiceException('DB_ERROR', error.
       message)
    return data
  }

  async createProduct(product: Partial<Product>): Promise<
     Product> {
    const { data, error } = await supabase
      .from('produtos')
      .insert(product)
      .select()
      .single()

    if (error) throw new ServiceException('CREATE_ERROR', error.
       message)
    return data
  }
}
```

Listing 8: Exemplo de Operações CRUD

7 Funcionalidades Avançadas

7.1 Sistema de Analytics Avançado

7.1.1 Análises Estatísticas

O sistema implementa análises estatísticas robustas através do `advancedAnalyticsService`:

- **Estatísticas Descritivas**: Média, mediana, moda, desvio padrão

- **Análise de Tendências**: Regressão linear, correlações
- **Detecção de Sazonalidade**: Padrões cíclicos, autocorrelação
- **Forecasting**: Previsão usando suavização exponencial
- **Detecção de Anomalias**: Baseada em Z-score e desvios
- **Análise de Distribuição**: Histogramas, bondade de ajuste

7.1.2 Business Intelligence

Funcionalidades de BI integradas:

```
interface BusinessIntelligence {
  kpis: {
    revenue: {
      current: number
      growth: number
      target: number
      achievement: number
    }
    efficiency: {
      inventoryTurnover: number
      stockoutRate: number
      fillRate: number
      cycleTime: number
    }
    quality: {
      accuracyRate: number
      errorRate: number
      customerSatisfaction: number
    }
  }
  benchmarks: {
    industry: IndustryBenchmark
    internal: InternalBenchmark
  }
  recommendations: Recommendation[]
}
```

Listing 9: KPIs Automatizados

7.2 Inteligência Artificial

7.2.1 Análise Preditiva

O `predictiveAnalyticsService` implementa modelos preditivos:

- **Previsão de Demanda**: Modelos ARIMA e suavização exponencial
- **Otimização de Estoque**: Cálculo de pontos de reposição ótimos
- **Análise de Churn**: Identificação de produtos em declínio

- **Segmentação ABC**: Classificação automática por importância
- **Análise de Cestas**: Market basket analysis

7.2.2 Integração com APIs de IA

```
export class AIAnalyticsService {
  private readonly GEMINI_API_KEY = import.meta.env.
     VITE_GEMINI_API_KEY

  async generateInsights(data: any[]): Promise<string[]> {
    const prompt = this.buildAnalysisPrompt(data)

    const response = await this.callGeminiAPI(prompt)
    return this.parseInsights(response)
  }

  private async callGeminiAPI(prompt: string) {
    const response = await fetch(
      `https://generativelanguage.googleapis.com/v1beta/models/
         gemini-pro:generateContent?key=${this.GEMINI_API_KEY}`,
      {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          contents: [{ parts: [{ text: prompt }] }]
        })
      }
    )

    return response.json()
  }
}
```

Listing 10: Integração Gemini AI

7.3 Sistema de Relatórios Avançados

7.3.1 Exportação Multi-Formato

O `advancedExportService` suporta múltiplos formatos:

- **PDF**: Relatórios formatados com gráficos
- **Excel**: Planilhas com dados e tabelas dinâmicas
- **CSV**: Dados estruturados para análise externa
- **PowerBI**: Datasets otimizados para dashboards
- **Imagem**: Screenshots de dashboards

7.3.2 Customização de Relatórios

```
interface ExportOptions {
  format: 'pdf' | 'excel' | 'csv' | 'json' | 'powerbi' | 'image'
  includeAI: boolean
  includePredictive: boolean
  includeCharts: boolean
  includeRawData: boolean
  customSections?: string[]
  branding?: {
    logo?: string
    companyName?: string
    colors?: {
      primary: string
      secondary: string
    }
  }
}
```

Listing 11: Opções de Exportação

7.4 Visualizações Avançadas

7.4.1 Tipos de Gráfico Suportados

O `advancedChartsService` implementa visualizações sofisticadas:

- **Line Charts**: Tendências temporais
- **Bar/Column Charts**: Comparações categóricas
- **Pie/Doughnut Charts**: Distribuições percentuais
- **Radar Charts**: Análises multi-dimensionais
- **Scatter Plots**: Correlações entre variáveis
- **Bubble Charts**: Três dimensões de dados
- **Heatmaps**: Matrizes de correlação
- **Gauge Charts**: Indicadores de performance

7.4.2 Configuração de Gráficos

```
interface AdvancedChartData {
  type: 'line' | 'bar' | 'doughnut' | 'radar' | 'bubble' | '
     scatter' | 'polarArea' | 'heatmap'
  data: ChartData
  options: ChartOptions
  config?: ChartConfiguration
}

```

```
export class AdvancedChartsService {
  generateSalesPerformanceChart(
    salesData: any[],
    period: string
  ): AdvancedChartData {
    const labels = salesData.map(item => item.date)
    const sales = salesData.map(item => item.total)
    const movingAverage = this.calculateMovingAverage(sales, 7)

    return {
      type: 'line',
      data: {
        labels,
        datasets: [
          {
            label: 'Vendas Di rias',
            data: sales,
            borderColor: '#667eea',
            backgroundColor: 'rgba(102, 126, 234, 0.1)',
            fill: true,
            tension: 0.4
          },
          {
            label: 'M dia M vel (7 dias)',
            data: movingAverage,
            borderColor: '#f093fb',
            backgroundColor: 'transparent',
            borderWidth: 2,
            borderDash: [5, 5],
            pointRadius: 0
          }
        ]
      },
      options: this.getAdvancedLineOptions('Vendas por Per odo
        ', 'R$')
    }
  }
}
```

Listing 12: Configuração de Gráfico

8 Segurança e Performance

8.1 Medidas de Segurança

8.1.1 Autenticação e Autorização

- **Supabase Auth**: Sistema robusto de autenticação
- **JWT Tokens**: Tokens seguros com expiração automática
- **Row Level Security**: Controle granular de acesso

- **HTTPS Only**: Comunicação criptografada
- **Session Management**: Controle de sessões ativas

8.1.2 Proteção de Dados

- **Criptografia**: Senhas hasheadas com bcrypt
- **Sanitização**: Prevenção de SQL injection
- **Validação**: Validação rigorosa de inputs
- **Auditoria**: Log completo de ações
- **Backup**: Backup automático de dados

8.1.3 Políticas RLS

```
-- Pol tica para acesso a produtos
CREATE POLICY "authenticated_users_products" ON produtos
    FOR ALL USING (auth.role() = 'authenticated');

-- Pol tica para logs (somente leitura)
CREATE POLICY "read_logs" ON logs
    FOR SELECT USING (auth.role() = 'authenticated');

-- Pol tica para configura  es (somente admin)
CREATE POLICY "admin_settings" ON app_settings
    FOR ALL USING (auth.jwt() ->> 'role' = 'admin');
```

Listing 13: Políticas de Segurança

8.2 Otimizações de Performance

8.2.1 Frontend

- **Code Splitting**: Carregamento sob demanda
- **Lazy Loading**: Componentes carregados quando necessário
- **Virtual Scrolling**: Listas grandes otimizadas
- **Caching**: Cache inteligente de dados
- **Compression**: Assets comprimidos

8.2.2 Backend/Database

- **Indexação**: Índices otimizados para consultas
- **Connection Pooling**: Pool de conexões eficiente
- **Query Optimization**: Consultas SQL otimizadas

- **Caching Strategy**: Cache de resultados frequentes
- **Real-time Subscriptions**: Atualizações em tempo real

8.2.3 Monitoramento

```
export class PerformanceMonitor {
  static measureExecutionTime<T>(
    fn: () => Promise<T>,
    operation: string
  ): Promise<T> {
    const start = performance.now()

    return fn().finally(() => {
      const end = performance.now()
      const duration = end - start

      console.log(`${operation} took ${duration.toFixed(2)}ms`)

      // Log performance metrics
      this.logMetric({
        operation,
        duration,
        timestamp: new Date()
      })
    })
  }
}
```

Listing 14: Monitoramento de Performance

9 Deployment e Manutenção

9.1 Processo de Build

9.1.1 Scripts de Build

```
{
  "scripts": {
    "dev": "vite",
    "build": "vue-tsc && vite build",
    "preview": "vite preview",
    "lint": "eslint src --ext .vue,.ts,.tsx",
    "type-check": "vue-tsc --noEmit"
  }
}
```

Listing 15: Package.json Scripts

9.1.2 Configuração de Build

```
import { defineConfig } from 'vite'
import vue from '@vitejs/plugin-vue'
import { resolve } from 'path'

export default defineConfig({
  plugins: [vue()],
  resolve: {
    alias: {
      '@': resolve(__dirname, 'src')
    }
  },
  build: {
    target: 'es2015',
    outDir: 'dist',
    assetsDir: 'assets',
    sourcemap: false,
    rollupOptions: {
      output: {
        manualChunks: {
          vendor: ['vue', 'vue-router', 'pinia'],
          charts: ['chart.js', 'vue-chartjs'],
          utils: ['axios', 'date-fns']
        }
      }
    }
  }
})
```

Listing 16: Vite Config

9.2 Deployment em Produção

9.2.1 Vercel (Recomendado)

1. Configure as variáveis de ambiente no painel da Vercel
2. Conecte o repositório GitHub
3. A build será executada automaticamente
4. Configure domínio personalizado (opcional)

```
{
  "builds": [
    {
      "src": "package.json",
      "use": "@vercel/static-build",
      "config": {
        "distDir": "dist"
      }
```

```
    }
  ],
  "routes": [
    {
      "src": "/(.*)",
      "dest": "/index.html"
    }
  ]
}
```

Listing 17: vercel.json

9.2.2 Netlify

```
[build]
  command = "npm run build"
  publish = "dist"

[[redirects]]
  from = "/*"
  to = "/index.html"
  status = 200

[build.environment]
  NODE_VERSION = "18"
```

Listing 18: netlify.toml

9.3 Monitoramento e Logs

9.3.1 Monitoramento de Aplicação

- **Error Tracking**: Sentry ou similar
- **Performance Monitoring**: Web Vitals
- **Analytics**: Google Analytics 4
- **Uptime Monitoring**: Pingdom ou similar

9.3.2 Logs Estruturados

```
export enum LogLevel {
  DEBUG = 0,
  INFO = 1,
  WARN = 2,
  ERROR = 3
}

export interface LogEntry {
  timestamp: Date
```

```
  level: LogLevel
  message: string
  metadata?: any
  userId?: string
  sessionId?: string
}

export class Logger {
  static log(level: LogLevel, message: string, metadata?: any) {
    const entry: LogEntry = {
      timestamp: new Date(),
      level,
      message,
      metadata,
      userId: this.getCurrentUserId(),
      sessionId: this.getSessionId()
    }

    // Send to logging service
    this.sendToSupabase(entry)
  }
}
\end{lstlisting>

\subsection{Backup e Recupera  o}

\subsubsection{Estrat gia de Backup}

\begin{itemize}
    \item \textbf{Backup Di rio}: Supabase autom tico
    \item \textbf{Backup Semanal}: Export completo de dados
    \item \textbf{Point-in-time Recovery}: At  7 dias
    \item \textbf{Backup de C digo}: Git com m ltiplos remotes
\end{itemize}

\subsubsection{Procedimento de Recupera  o}

\begin{enumerate}
    \item Identificar ponto de falha
    \item Selecionar backup apropriado
    \item Restaurar via Supabase Dashboard
    \item Verificar integridade dos dados
    \item Executar testes de valida  o
    \item Comunicar usu rios sobre restaura  o
\end{enumerate}

% ========================================
% SE  O 10: TROUBLESHOOTING
% ========================================

\section{Troubleshooting e FAQ}
```

```
61
62 \subsection{Problemas Comuns}
63
64 \subsubsection{Erros de Autentica  o}
65
66 \begin{warningbox}{Erro: "Invalid credentials"}
67 \textbf{Causa}: Credenciais incorretas ou sess o expirada \\
68 \textbf{Solu  o}:
69 \begin{itemize}
70     \item Verificar email/senha
71     \item Limpar cache do navegador
72     \item Verificar configura  es Supabase
73     \item Renovar tokens de sess o
74 \end{itemize}
75 \end{warningbox}
76
77 \subsubsection{Problemas de Performance}
78
79 \begin{warningbox}{Lentid o na Aplica  o}
80 \textbf{Causas Poss veis}:
81 \begin{itemize}
82     \item Consultas n o otimizadas
83     \item Cache desatualizado
84     \item Muitos dados na mem ria
85     \item Conex o lenta com BD
86 \end{itemize}
87 \textbf{Solu  es}:
88 \begin{itemize}
89     \item Otimizar queries SQL
90     \item Implementar pagina  o
91     \item Usar lazy loading
92     \item Verificar  ndices  do BD
93 \end{itemize}
94 \end{warningbox}
95
96 \subsubsection{Erros de Build}
97
98 \begin{warningbox}{TypeScript Compilation Errors}
99 \textbf{Comando de Diagn stico}: \texttt{npm run type-check} \\
100 \textbf{Solu  es Comuns}:
101 \begin{itemize}
102     \item Verificar tipos importados
103     \item Atualizar depend ncias
104     \item Limpar cache: \texttt{rm -rf node\_modules package-
        lock.json}
105     \item Reinstalar: \texttt{npm install}
106 \end{itemize}
107 \end{warningbox}
108
109 \subsection{Comandos de Diagn stico}
110
```

```
\subsubsection{Verifica  o de Sistema}

\begin{lstlisting}[language=bash, caption=Scripts de
   Diagn stico]
# Verificar vers es
node --version
npm --version

# Verificar depend ncias
npm ls

# Verificar tipos TypeScript
npm run type-check

# Verificar lint
npm run lint

# Teste de build
npm run build

# Verificar configura  o Supabase
npx supabase status
```

Listing 19: Sistema de Logs

9.3.3 Logs de Debug

```
// Ativar modo debug
localStorage.setItem('DEBUG_MODE', 'true')

// Debug de queries Supabase
const { data, error } = await supabase
  .from('produtos')
  .select('*')
  .explain({ analyze: true, verbose: true })
```

Listing 20: Debug Mode

9.4 FAQ - Perguntas Frequentes

9.4.1 Funcionalidades

- **P**: Como importar produtos em massa?
  R: Use a funcionalidade de importação Excel na rota /inventory
- **P**: É possível customizar relatórios?
  R: Sim, na rota /reports há opções avançadas de customização
- **P**: Como configurar alertas de estoque baixo?
  R: Configure limites mínimos em /settings e ative notificações

9.4.2 Técnicas

- **P**: Como fazer backup dos dados?
  R: Use o painel Supabase ou a exportação automática em /reports
- **P**: É possível integrar com outros sistemas?
  R: Sim, através da API REST do Supabase ou webhooks customizados
- **P**: Como otimizar performance para muitos produtos?
  R: Use filtros, paginação e índices adequados no banco

10 API e Integrações

10.1 API Supabase

10.1.1 Endpoints Principais

O Supabase gera automaticamente uma API REST para todas as tabelas:

```
# Produtos
GET    /rest/v1/produtos
POST   /rest/v1/produtos
PATCH  /rest/v1/produtos?id=eq.{id}
DELETE /rest/v1/produtos?id=eq.{id}

# Movimenta  es
GET    /rest/v1/movements
POST   /rest/v1/movements

# Relat rios
GET    /rest/v1/reports
POST   /rest/v1/reports
```

Listing 21: Exemplos de Endpoints

10.1.2 Autenticação API

```
curl -X GET 'https://seu-projeto.supabase.co/rest/v1/produtos' \
  -H "apikey: SUA_ANON_KEY" \
  -H "Authorization: Bearer SEU_JWT_TOKEN"
```

Listing 22: Headers de Autenticação

10.2 Integrações Externas

10.2.1 Google Gemini AI

```
export class AIService {
  private readonly apiKey = import.meta.env.VITE_GEMINI_API_KEY
  private readonly baseUrl = 'https://generativelanguage.
     googleapis.com/v1beta'
```

```

  async generateContent(prompt: string): Promise<string> {
    const response = await fetch(
      `${this.baseUrl}/models/gemini-pro:generateContent?key=${
        this.apiKey}`,
      {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          contents: [{ parts: [{ text: prompt }] }]
        })
      }
    )

    const data = await response.json()
    return data.candidates[0].content.parts[0].text
  }
}
```

Listing 23: Integração Gemini

10.2.2 Webhooks

Configurar webhooks para eventos importantes:

```
-- Fun   o para webhook
CREATE OR REPLACE FUNCTION notify_low_stock()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW.current_stock <= NEW.min_stock THEN
    PERFORM net.http_post(
      url := 'https://seu-webhook-url.com/low-stock',
      headers := '{"Content-Type": "application/json"}'::jsonb,
      body := json_build_object(
        'product_id', NEW.id,
        'product_name', NEW.nome,
        'current_stock', NEW.current_stock,
        'min_stock', NEW.min_stock
      )::jsonb
    );
  END IF;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Trigger
CREATE TRIGGER low_stock_webhook
  AFTER UPDATE OF current_stock ON produtos
  FOR EACH ROW
  EXECUTE FUNCTION notify_low_stock();
```

Listing 24: Webhook para Estoque Baixo

11 Conclusão

11.1 Resumo das Capacidades

O **GestãoZe System** representa uma solução completa e moderna para gestão de estoque, combinando:

Pontos Fortes do Sistema

- **Arquitetura Moderna**: Vue.js 3 com TypeScript
- **Backend Robusto**: Supabase com PostgreSQL
- **Análises Avançadas**: IA e análise preditiva
- **Interface Intuitiva**: UX otimizada para produtividade
- **Segurança Robusta**: RLS e autenticação segura
- **Escalabilidade**: Arquitetura preparada para crescimento
- **Relatórios Ricos**: Múltiplos formatos de exportação
- **Integração Flexível**: APIs e webhooks

11.2 Roadmap de Desenvolvimento

11.2.1 Próximas Funcionalidades

- **Mobile App**: Aplicativo nativo com React Native
- **E-commerce Integration**: Integração com plataformas de venda
- **Advanced AI**: Modelos de deep learning customizados
- **Multi-tenancy**: Suporte para múltiplas empresas
- **IoT Integration**: Sensores de estoque automáticos
- **Blockchain**: Rastreabilidade de produtos

11.3 Suporte e Comunidade

11.3.1 Recursos de Apoio

- **Documentação**: Sempre atualizada
- **Issues GitHub**: Relatório de bugs e sugestões
- **Comunidade**: Fórum de discussão

- **Tutoriais**: Vídeos e guias passo a passo
- **API Reference**: Documentação completa da API

11.3.2 Contribuição

O sistema é desenvolvido seguindo boas práticas de código aberto:

- **Code Reviews**: Revisão rigorosa de código
- **Testing**: Cobertura de testes abrangente
- **CI/CD**: Pipeline automatizado de deployment
- **Documentation**: Documentação viva e atualizada

GestãoZe System v1.0.0

Sistema de Gestão de Estoque Inteligente

Desenvolvido com Vue.js 3, TypeScript e Supabase

gestao.restpedacinhodoceu.com.br