

Arquitetura do software

visão geral em camadas;

componentes e responsabilidades;

modelo de dados essencial (tabelas principais);

fluxos críticos (alertas de validade, cadastro em lote, auditoria);

requisitos não-funcionais e decisões tecnológicas recomendadas;

diagrama lógico textual (para usar como base ao desenhar diagramas UML/arquitetura gráficos);

checklist de entregáveis e próximos passos;

explicação resumida no final.

1. Visão geral (camadas)




1. Apresentação (UI Desktop)

Aplicação desktop em VB.NET (WPF ou WinForms conforme necessidade de UI rica).

Telas principais: Dashboard, Lista de Produtos, Cadastro de Produto/Lote, Consulta/Busca Avançada, Relatórios, Backup/Configurações, Ajuda/Manual.

Indicadores visuais por cor (verde / amarelo / laranja / vermelho) conforme o documento.

2. Camada de Aplicação / Lógica de Negócio

Regras de cálculo de status de validade (15 dias =  Amarelo; 7 dias =  Laranja; vencido =  Vermelho).

Regras de lote e gerenciamento de quantidade (fardos/lotos).

Motor de alertas (verificação periódica local).

Serviços de relatório, importação em lote, exportação, auditoria.

3. Persistência (Banco de Dados Local)

Banco SQL local (SQLite para implantação simples ou SQL Server Express se houver requisitos futuros de replicação).

Acesso via ORM leve (ex.: Dapper) ou Entity Framework (se preferir abstração).

4. Infra / Utilitários

Agendador local (timer ou Windows Service opcional) para verificação e geração de alertas.

Backup local (rotina periódica sobre arquivo .bak / cópia do .db).

Logs de auditoria e de aplicação.

Módulo de exportação (CSV/PDF) para relatórios e auditoria.

5. Segurança & Controle de Acesso

Login com permissões (Usuário e senhas já adicionados ao banco).

Registro de quem alterou/excluiu registros (audit trail).

2. Componentes detalhados e responsabilidades

2.1 UI / Frontend (Desktop)

Dashboard: resumo de produtos próximos ao vencimento, entradas/saídas do dia, indicadores rápidos por cor.

Módulo de Cadastro de Produto: campos: nome, código de barras, categoria, fornecedor, unidade, quantidade (fardo/lote), data entrada, data validade, local armazenamento. (Prioridade Alta no PDF).

Cadastro Por Lote: tela para inserir lotes múltiplos sem repetir unidade por unidade (Alta prioridade).

Consulta/Busca Avançada: filtros por fornecedor, categoria, lote, local, intervalo de datas. (Alta).

Relatórios: perdas por vencimento, movimentação (entrada/saída), histórico por fornecedor.

Administração: permissões, parâmetros de alerta (15 dias / 7 dias ajustáveis).

2.2 Lógica de Negócio / Serviços

Serviço de Validação/Status: calcula status de cada lote (verde/amarelo/laranja/vermelho).

Agendador de Alertas: roda localmente (ex.: a cada 24h ou ao iniciar a aplicação) e gera notificações visuais e relatórios de alerta.

Controle de Estoque: funções para entrada/saída de produtos por lote, decremento automático ao vender/retirar.

Auditoria: gravar operações de criação, alteração, exclusão com usuário e timestamp. (Média prioridade no documento).

2.3 Persistência / Dados

Tables principais (ver seção 3 para esquema).

Índices: sobre código de barras, data de validade, lote e fornecedor para buscas rápidas.

Backup: rotina programada com opção manual.

Exportação/importação: CSV para lote de cadastros.

2.4 Infra & Deploy

Executável desktop + arquivo de banco (SQLite) ou instância local SQL Server Express.

Requisitos de máquina conforme PDF: Windows 10 (64 bits), 4 GB RAM, Intel Core i5 (ou similar).

3. Modelo de dados (tabelas principais — proposta)

-- Observação: sintaxe genérica; ajustar para SQLite / SQL Server

```
CREATE TABLE Fornecedores (  
    fornecedor_id INTEGER PRIMARY KEY,  
    nome TEXT NOT NULL,  
    contato TEXT,  
    telefone TEXT,  
    observacoes TEXT  
);
```

```
CREATE TABLE Categorias (  
    categoria_id INTEGER PRIMARY KEY,  
    nome TEXT NOT NULL  
);
```

```
CREATE TABLE Locais (  
    local_id INTEGER PRIMARY KEY,  
    nome TEXT NOT NULL, -- ex: prateleira A, freezer 1
```

```
tipo TEXT -- prateleira, freezer, geladeira, depósito  
);
```

```
CREATE TABLE Produtos (  
    produto_id INTEGER PRIMARY KEY,  
    nome TEXT NOT NULL,  
    codigo_barras TEXT UNIQUE,  
    categoria_id INTEGER,  
    fornecedor_id INTEGER,  
    descricao TEXT,  
    unidade_medida TEXT, -- fardo, unidade...  
    FOREIGN KEY(categoria_id) REFERENCES Categorias(categoria_id),  
    FOREIGN KEY(fornecedor_id) REFERENCES Fornecedores(fornecedor_id)  
);
```

```
CREATE TABLE Lotes (  
    lote_id INTEGER PRIMARY KEY,  
    produto_id INTEGER NOT NULL,  
    codigo_lote TEXT,  
    quantidade INTEGER NOT NULL, -- quantidade total no lote (fardos ou unidades)  
    data_entrada DATE,  
    data_validade DATE,  
    local_id INTEGER,  
    status INTEGER, -- opcional cache de status (0=verde,1=amarelo,2=laranja,3=vermelho)  
    ativo BOOLEAN DEFAULT 1,  
    FOREIGN KEY(produto_id) REFERENCES Produtos(produto_id),  
    FOREIGN KEY(local_id) REFERENCES Locais(local_id)  
);
```

```
CREATE TABLE Movimentacoes (  
    movimentacao_id INTEGER PRIMARY KEY,
```

```
lote_id INTEGER,  
tipo TEXT, -- entrada | saída  
quantidade INTEGER,  
data_movimentacao DATETIME,  
usuario TEXT,  
observacao TEXT,  
FOREIGN KEY(lote_id) REFERENCES Lotes(lote_id)  
);
```

```
CREATE TABLE Usuarios (  
    usuario_id INTEGER PRIMARY KEY,  
    username TEXT UNIQUE,  
    senha_hash TEXT,  
    nivel_acesso TEXT -- admin | user  
);
```

```
CREATE TABLE Auditoria (  
    auditoria_id INTEGER PRIMARY KEY,  
    entidade TEXT, -- ex: Lotes  
    entidade_id INTEGER,  
    acao TEXT, -- CREATE | UPDATE | DELETE  
    usuario TEXT,  
    data_hora DATETIME,  
    detalhes TEXT  
);
```

```
CREATE TABLE Configuracoes (  
    chave TEXT PRIMARY KEY,  
    valor TEXT  
);
```

Essas tabelas cobrem os requisitos funcionais do PDF: cadastro, lote, local de armazenamento, filtros e auditoria.

4. Lógica de alertas (pseudocódigo)

Para cada lote ativo:

dias_para_vencimento = data_validade - hoje

se dias_para_vencimento < 0:

status = VERMELHO

senao se dias_para_vencimento <= 7:

status = LARANJA

senao se dias_para_vencimento <= 15:

status = AMARELO

senao:

status = VERDE

salvar status (opcional cache em Lotes.status)

se status mudou para LARANJA/VERMELHO ou AMARELO:

registrar alerta (exibir no dashboard + gerar relatório)

Agendador roda esta rotina ao iniciar a aplicação e uma vez por dia (ou com periodicidade configurável).

5. Regras e UI para cadastro em lote (Alta prioridade)

Tela de importação via CSV com colunas mínimas: código_barras/produto, codigo_lote, quantidade, data_entrada, data_validade, local_id.

Importação cria o produto caso não exista (opcional) ou associa ao produto existente.

Confirmação de lote com resumo antes de gravar.

6. Auditoria / Controle de acesso

Registrar em Auditoria toda criação/alteração/exclusão com usuário e timestamp. (Média prioridade no documento).

Acesso por usuário/senha. Perfis: Admin (config, backup, restauração, usuários) e Operador (cadastro/consulta/movimentação). (Alta prioridade).

7. Relatórios essenciais (prioritários)

Produtos por status de validade (tabela e gráfico no dashboard).

Itens expirados (detalhe por lote).

Movimentação por período (entradas/saídas).

Perdas por vencimento (valor aproximado se tabela de custo estiver presente).

Relatório por fornecedor / categoria / local.

8. Requisitos Não-Funcionais e decisões

Banco local: suportar SQLite (simples) ou SQL Server Express (se multiusuário local). O documento especifica banco SQL local — atender.

Offline: funcionalidade completa sem internet. Sincronização futura opcional.

UI: design limpo, ícones visuais, contraste para as cores de status.

Desempenho: consultas indexadas por validade e código de barras; cargas de lote processadas em transações.

Segurança: senhas armazenadas com hash; backups encriptados se necessário.

Requisitos mínimos: Windows 10 64-bit, 4GB RAM, Core i5 — seguir o PDF.

9. Diagrama lógico (texto — fluxo de alto nível)

[Usuario Desktop] --> [UI (VB.NET)] --> [Serviços de Aplicação (Validações, Alertas, Movimentação, Relatórios)] --> [DB Local (SQLite/SQL Server Express)]

\

--> [Módulo Backup / Exportação / Importação CSV]

--> [Log/Auditoria]

Opcional: se houver necessidade futura, substituir DB local por camada de sincronização para servidor central.

10. Checklist mínimo para desenvolvimento (entregáveis)

1. Protótipo de telas (wireframes) — Dashboard, Cadastro Produto/Lote, Importação em lote, Relatórios, Usuários.

2. Esquema do banco e scripts de criação.
3. Implementação da camada de persistência (repositórios).
4. Serviço de verificação de validade + agendador.
5. Módulo de auditoria e controle de acesso.
6. Relatórios principais exportáveis (PDF/CSV).
7. Testes unitários para regras de cálculo de validade.
8. Manual de usuário + telas de ajuda (documento exigido no PDF).

11. Possíveis extensões futuras (arquitetura escalável)

Sincronização com servidor (multi-loja), API REST para integrar com PDV/ERP.

App mobile para leitura de código de barras e registro de entrada/saída.

Integração com leitores de código de barras USB.

Notificações por e-mail/SMS quando houver muitos itens próximos ao vencimento.

12. Riscos e Mitigações

Perda de dados: implementar backups automáticos e testes de restauração.

Busca lenta em grandes volumes: índices e paginação nas consultas.

Erro humano no cadastro em lote: validar CSV com pré-visualização antes de inserir.

Alteração indevida: auditoria e logs com rollback quando aplicável.

13. Exemplo de rotina de entrega (metodologia sugerida)

Sprint 0: wireframes, schema DB, ambiente de desenvolvimento.

Sprint 1: CRUD de produtos/lotes, UI básica.

Sprint 2: Agendador/alertas + dashboard status.

Sprint 3: Importação em lote + relatórios.

Sprint 4: Auditoria, controle de acesso, testes, manual.

14. Explicação resumida do que eu fiz (pedido final)

1. Li e usei o PDF que você enviou como fonte principal dos objetivos e requisitos (funcionais e não-funcionais).

2. Projetei uma arquitetura em camadas (UI desktop, lógica de negócio, persistência local, utilitários) alinhada com as exigências do documento (alertas 15/7 dias, cores de status, cadastro em lote, banco SQL local, offline, requisitos de hardware).
3. Propus um modelo de dados com as tabelas essenciais (Produtos, Lotes, Fornecedores, Locais, Movimentações, Usuários, Auditoria) para suportar os fluxos descritos no PDF.
4. Descrevi fluxos críticos (verificação de validade/alertas, importação em lote, auditoria) com pseudocódigo para implementação direta.
5. Listei decisões tecnológicas, riscos, mitigações, relatórios e entregáveis necessários para transformar os requisitos em software funcional.

ARQUITETURA DO SISTEMA GESTCOMTEC — ORGANIZADA POR ÁREA

1. FRONTEND (Camada de Apresentação / Interface Desktop)

Responsabilidade

Exibir as informações do sistema para o usuário e receber suas interações.

É a camada que o operador e o administrador do armazém irão utilizar no dia a dia.

Principais componentes

Dashboard (Painel Principal): exibe alertas de produtos prestes a vencer com cores (verde, amarelo, laranja, vermelho).

Tela de Cadastro de Produto e Lote: campos essenciais — nome, código de barras, quantidade, validade, fornecedor, local.

Consulta e Busca Avançada: filtros por categoria, validade, local e fornecedor.

Relatórios: perdas por vencimento, movimentação, histórico de estoque.

Administração de Usuários: controle de permissões e senhas.

Configurações/Ajuda: definição de parâmetros (dias de alerta, cores, backup, idioma, etc).

Tecnologias sugeridas

VB.NET (WinForms ou WPF) — conforme o documento original.

Framework de UI moderno: WPF + MVVM (caso queira separar lógica e apresentação).

Biblioteca de gráficos (para relatórios): LiveCharts ou OxyPlot.

Orientação para o desenvolvedor

1. Inicie com protótipos (wireframes) de cada tela.
2. Use cores fixas para os status (verde, amarelo, laranja, vermelho).
3. Crie um menu lateral fixo e uma barra superior com atalhos para o Dashboard e Relatórios.
4. Cada tela deve se conectar a uma camada de serviço (backend) por meio de métodos simples (ex.: GetProdutos(), SalvarLote(), GerarRelatorio()).
5. Evite lógica de negócio no código da interface — ela deve apenas exibir e enviar dados.

2. BACKEND (Lógica de Negócio + Persistência de Dados)

Responsabilidade

Processar as regras do sistema, validar dados, calcular alertas, armazenar informações e gerenciar transações com o banco de dados.

Principais módulos

Validação e Alertas de Validade: calcula status de cada lote conforme dias até o vencimento (15 = amarelo, 7 = laranja, 0 = vermelho).

Controle de Estoque: gerencia entrada e saída de produtos (quantidade por lote).

Cadastro e Importação em Lote: inserção em massa de produtos via CSV.

Relatórios: geração de relatórios de movimentação, perdas e vencimentos.

Auditoria: registra toda modificação de dados com usuário, data e ação.

Autenticação e Permissões: valida login e define níveis de acesso.

Agendador: rotina automática diária para verificar produtos próximos ao vencimento.

Backup e Exportação: gera cópia de segurança e exporta relatórios para CSV ou PDF.

Tecnologias sugeridas

Linguagem: VB.NET ou C# (.NET Framework / .NET 6+).

Banco de Dados: SQLite (para implantação simples) ou SQL Server Express (se multiusuário).

ORM: Dapper (leve e rápido) ou Entity Framework (para CRUDs automáticos).

Agendador: System.Threading.Timer ou Quartz.NET (para tarefas diárias).

Relatórios: FastReport.NET ou iTextSharp (para PDF).

Orientação para o desenvolvedor

1. Centralize toda a lógica de negócio em classes separadas (ex.: EstoqueService, AlertaService, AuditoriaService).
2. Crie uma camada de repositórios (ex.: ProdutoRepository, LoteRepository) que faz toda a comunicação com o banco.
3. Utilize DTOs ou ViewModels para transferir dados entre o backend e a UI, evitando expor entidades diretamente.

4. Mantenha transações atômicas ao alterar quantidades (para não duplicar ou perder estoque).
5. Gere logs de erros e operações críticas (arquivo .log ou tabela Logs no BD).
6. Faça backup automático diário e uma opção manual no menu de Configurações.

3. BANCO DE DADOS (Camada de Persistência)

Responsabilidade

Armazenar de forma estruturada e confiável todos os dados da aplicação: produtos, lotes, usuários, movimentações, auditoria e configurações.

Tabelas principais

Produtos

Lotes

Fornecedores

Locais

Movimentacoes

Usuarios

Auditoria

Configuracoes

⚙ Boas práticas

1. Utilize chaves estrangeiras para relacionar Produtos, Lotes e Fornecedores.
2. Crie índices em data_validade, codigo_barras e fornecedor_id para buscas rápidas.
3. Padronize datas em formato UTC.

4. Configure backup automático em local seguro (ex.: /backups/gestcomtec/).
5. Crie views SQL para relatórios de vencimentos e movimentações.

Orientação para o desenvolvedor

Se for usar SQLite, crie o arquivo gestcomtec.db junto com a aplicação.

Se usar SQL Server Express, disponibilize um script .sql para criação automática das tabelas na primeira execução.

Adicione um campo status nas tabelas de lotes para armazenar o código de cor atual (0=verde, 1=amarelo, 2=laranja, 3=vermelho).





4. UI / DESIGN (Camada Visual e Experiência do Usuário)

Responsabilidade

Garantir que o sistema seja intuitivo, legível e agradável de usar, mesmo para usuários com pouca experiência.

Diretrizes visuais

Cores de alerta:

-  Verde = normal
-  Amarelo = 15 dias para vencer
-  Laranja = 7 dias para vencer
-  Vermelho = vencido

Layout:

Menu lateral fixo com ícones (Produtos, Lotes, Relatórios, Configurações).

Painel principal com gráfico e lista de produtos próximos ao vencimento.

Pop-ups de alerta automático ao abrir o sistema.

Tipografia: clara e legível (Segoe UI / Roboto).

Ícones: simples e padronizados (Material Icons).

Orientação para o designer/UI developer

1. Priorize a usabilidade — o operador precisa identificar rapidamente produtos críticos.
2. Evite sobrecarregar telas; prefira dados resumidos com botão de “Ver Detalhes”.
3. Adote uma identidade visual coerente (ex.: azul + cinza + branco).
4. Crie um guia de cores e fontes (mini design system).
5. Teste contraste e legibilidade em diferentes monitores.

RESUMO FINAL (Fluxo de Desenvolvimento Recomendado)

1. **Planejar:** leia o documento de requisitos (PDF) e defina escopo inicial.
2. **Modelar Banco:** crie o banco conforme as tabelas acima.
3. **Implementar Backend:** serviços e repositórios primeiro (sem interface).
4. **Criar UI/Frontend:** construa telas e conecte aos serviços do backend.
5. **Aplicar Design:** aplique cores, ícones e validações visuais.
6. **Testar e Documentar:** valide alertas, cadastros e backups.
7. **Gerar Manual de Usuário (solicitado no documento original).**

(Alta importância)