GOVERNO DO ESTADO DA PARAÍBA SECRETARIA DE ESTADO DA CIÊNCIA, TECNOLOGIA, INOVAÇÃO
E ENSINO SUPERIOR - SECTIES UNIVERSIDADE ESTADUAL DA PARAÍBA

Maria Eduarda Lourenço Maia2024206510006
Maria Kassiana Melo Ferreira 2024206510013 Murilo Araújo Tôrres 2024206510020
Ysack Evangelista do Nascimento 2024206510061

Controle Inteligente de Estoque e Precificação para Pequenos Negócios

João Pessoa/PB 29 de Setembro 2025

SUMÁRIO

1. **Papéis do time (Product Owner, Scrum Master, Devs)** PÁG 3 e 4

2. **Problemática do cliente real (Mercearia).** PÁG 5

3. **Product Backlog inicial (5 a 7 histórias de usuário).** PÁG 5 e 6

4. **Refinamento do backlog.** PÁG. 5 a 7

5. **Priorização das histórias de usuário.** PÁG. 7 e 8

6. **RF e RNF.** PÁG. 8 a 11

7. **Definição da Sprint 1 (meta e tarefas).** PÁG. 12 a 14

8. **Quadro Kanban.** PÁG. 15

9. **Referências.** PÁG. 15 e 16

1. Papéis do time (Product Owner, Scrum Master, Devs)

O Product Owner e a visão estratégica da mercearia

No nosso projeto, a função de Product Owner será desempenhada por Kassiana, cuja principal responsabilidade é manter a visão estratégica do sistema gerenciador da mercearia e garantir que todas as entregas estejam alinhadas aos interesses do dono. Mais do que organizar listas de tarefas, Kassiana deve compreender profundamente o funcionamento do negócio, identificando gargalos e oportunidades que podem ser traduzidos em melhorias concretas.

Para isso, ela se envolve diretamente com o proprietário e os funcionários, ouvindo suas dificuldades e registrando suas necessidades no backlog do produto. Esse backlog não é apenas uma lista, mas sim um repositório vivo e dinâmico de prioridades, que vão desde ferramentas para controle de estoque, relatórios financeiros acessíveis até recursos de treinamento para os colaboradores. O desafio de Kassiana está em definir o que deve ser feito primeiro, sempre equilibrando urgência e impacto.

Por exemplo, se houver falhas recorrentes na reposição de mercadorias, ela deverá priorizar a configuração do sistema de estoque antes de lidar com funcionalidades secundárias, como relatórios de fornecedores ou campanhas de marketing. Assim, cada incremento entregue reflete uma resposta direta a um problema real da mercearia, garantindo que o sistema traga valor imediato ao negócio e contribua para sua evolução sustentável.

O Scrum Master e a fluidez do processo

Enquanto Kassiana cuida da visão, Ysack assume o papel de Scrum Master, responsável por assegurar que o processo de trabalho flua de maneira organizada, eficiente e colaborativa. Ele não se envolve em tarefas técnicas nem define o que deve ser feito, mas desempenha uma função essencial como facilitador e guardião da metodologia ágil.

Seu trabalho inclui remover obstáculos que possam comprometer o avanço da equipe, promover uma comunicação clara entre todos os membros e garantir que as práticas do Scrum sejam aplicadas corretamente. Se os Desenvolvedores encontrarem dificuldades para acessar informações de estoque ou houver divergências de entendimento sobre as prioridades estabelecidas por Kassiana, Ysack será o responsável por intervir, mediar e encontrar soluções que preservem o ritmo do trabalho.

Além disso, Ysack desempenha um papel educativo, conduzindo a equipe em direção a uma compreensão mais profunda da metodologia ágil. Ele promove workshops internos, reforça boas práticas e estimula a cultura de melhoria contínua. A cada ciclo de entregas (sprints), organiza reuniões de retrospectiva para identificar pontos fortes e oportunidades de aprimoramento. Dessa forma, a equipe não apenas executa suas funções, mas também evolui em maturidade, aprendendo a se tornar cada vez mais produtiva e eficiente.

Os Desenvolvedores como realizadores das entregas

No time, Murilo e Eduarda assumem o papel de Desenvolvedores, sendo os principais responsáveis por transformar os requisitos do backlog em soluções práticas para a mercearia. Eles são os realizadores das entregas, convertendo demandas abstratas em ferramentas e processos que otimizam o dia a dia do negócio.

Esse trabalho pode se manifestar de diversas formas: configurar sistemas de controle de estoque para evitar a falta ou o excesso de mercadorias; elaborar relatórios financeiros claros e objetivos, que

auxiliem o dono a compreender melhor as entradas e saídas do caixa; mapear os fluxos de trabalho dentro da mercearia para identificar pontos de ineficiência; e até criar materiais de treinamento que facilitem a adaptação dos funcionários a novos recursos implementados.

Além das tarefas técnicas, Murilo e Eduarda atuam de forma autônoma e colaborativa, definindo entre si como dividir e organizar as atividades. Essa autogestão é fundamental, pois permite que a equipe se adapte rapidamente às mudanças e aproveite ao máximo as competências individuais de cada um. Assim, enquanto um foca na configuração de ferramentas, o outro pode concentrar esforços na geração de relatórios ou na documentação do processo. O resultado é um fluxo de entregas consistente, que gera valor real e imediato para a mercearia.

A integração entre os papéis

A verdadeira força do Scrum não está em papéis isolados, mas na forma como eles se complementam. Kassiana, como Product Owner, garante que a equipe sempre saiba para onde está indo, definindo prioridades alinhadas às necessidades reais da mercearia. Ysack, como Scrum Master, cria as condições para que o caminho seja percorrido sem interrupções, mantendo o processo saudável e produtivo. Já Murilo e Eduarda, como Desenvolvedores, transformam essa visão em soluções concretas, aplicáveis e de impacto direto no cotidiano do negócio.

Essa integração se traduz em situações práticas: se Kassiana identifica que o fluxo de caixa está desorganizado e define como prioridade a criação de uma ferramenta de controle financeiro, Ysack organiza as reuniões e remove qualquer barreira que possa atrasar a execução, enquanto Murilo e Eduarda desenvolvem a solução, testam sua aplicabilidade e ensinam os funcionários a utilizá-la corretamente. Em pouco tempo, a mercearia sente os resultados, com processos mais organizados, maior clareza nas informações e um funcionamento mais ágil.

Além disso, esse ciclo de integração não se encerra em uma única entrega. Cada incremento realizado abre espaço para novas percepções e demandas. Assim, a mercearia passa a vivenciar uma dinâmica de evolução contínua, em que cada parte da equipe desempenha um papel insubstituível na geração de valor. O resultado não é apenas a criação de um sistema gerenciador, mas a transformação da forma como o negócio lida com seus desafios diários, conquistando maior eficiência e sustentabilidade.

2. Problemática do cliente real (Mercearia).

O cliente real deste projeto é o proprietário de uma mercearia de pequeno porte, que enfrenta dificuldades significativas na administração do seu negócio. Entre os principais problemas identificados estão a falta de controle eficiente do estoque, a ausência de critérios padronizados na definição de preços e a desorganização no armazenamento de produtos perecíveis, como frios, frutas e verduras. No controle de estoque, não existe um sistema estruturado para registrar entradas e saídas de mercadorias. Isso gera situações de desabastecimento em períodos de maior procura e, ao mesmo tempo, acúmulo de itens que acabam vencendo sem serem vendidos, ocasionando perdas financeiras. Em relação à precificação, os preços são definidos de maneira manual e pouco sistemática, o que resulta em valores inconsistentes quando comparados ao mercado. Essa prática reduz a competitividade da mercearia e compromete a lucratividade do negócio. No armazenamento, a falta de organização adequada contribui para a perda da qualidade de produtos perecíveis, aumentando o desperdício e prejudicando a satisfação dos clientes. Diante desse cenário, torna-se necessário propor um projeto baseado em metodologias ágeis, que possibilite a construção incremental de

uma solução tecnológica alinhada às reais necessidades do cliente. A aplicação do Scrum permitirá que o desenvolvimento seja feito em etapas curtas (sprints), possibilitando o acompanhamento contínuo do proprietário e a realização de ajustes a partir do feedback recebido. Dessa forma, o projeto busca não apenas oferecer uma ferramenta que auxilie na gestão de estoque, precificação e armazenamento, mas também promover uma melhoria contínua, garantindo que a solução final seja funcional, prática e adaptada à realidade do negócio.

3. Product Backlog inicial (5 a 7 histórias de usuário).

1. História de Usuário 1 – Registrar Entradas de Produtos

Como dono da mercearia, quero registrar a entrada de produtos no estoque, para que eu saiba a quantidade disponível.

2. História de Usuário 2 – Registrar Saídas de Produtos

Como dono da mercearia, quero registrar as vendas (saídas de produtos), para que eu tenha controle do que ainda está em estoque.

3. História de Usuário 3 – Alertas de Produtos em Falta

Como dono da mercearia, quero receber um aviso quando um produto estiver acabando, para que eu possa repor antes de faltar.

4. História de Usuário 4 – Alertas de Validade

Como dono da mercearia, quero ser avisado quando produtos perecíveis estiverem próximos do vencimento, para que eu reduza perdas financeiras.

5. História de Usuário 5 – Cadastro de Preços

Como dono da mercearia, quero definir e atualizar o preço dos produtos no sistema, para que eu tenha mais organização e padronização na precificação.

6. História de Usuário 6 – Sugestão de Preços

Como dono da mercearia, quero que o sistema sugira preços com base no mercado, para que eu seja mais competitivo e mantenha lucro justo.

7. História de Usuário 7 – Relatório de Estoque e Vendas

Como dono da mercearia, quero gerar relatórios de estoque e vendas, para que eu acompanhe o desempenho do negócio e planeje melhor as compras.

4. Refinamento do backlog

História 1: Registrar Entradas de Produtos

Tarefas:

- Criar formulário para cadastro de produtos (nome, categoria, quantidade, data de entrada).
- Configurar banco de dados para armazenar os registros.
- Implementar função de inserção automática via interface.
- Testar integração entre formulário e banco de dados.

História 2: Registrar Saídas de Produtos

Tarefas:

- Criar tela de registro de vendas com lista de produtos.
- Atualizar automaticamente a quantidade em estoque após cada venda.
- Gerar mensagem de confirmação após o registro.
- Testar cálculos e sincronização com o estoque.

História 3: Alertas de Produtos em Falta

Tarefas:

- Criar regra de negócio para detectar níveis mínimos de estoque.
- Exibir notificação visual (alerta) na interface.
- Permitir configuração do limite mínimo por produto.

História 4: Alertas de Validade

Tarefas:

- Adicionar campo de validade no cadastro de produtos.
- Criar função que verifique produtos com data próxima do vencimento.
- Gerar alerta automático com 5 dias de antecedência.

História 5: Cadastro de Preços

Tarefas:

- Adicionar campo de preço de custo e preço de venda.
- Criar tela para atualização de preços.
- Registrar histórico de alterações de preço.

História 6: Sugestão de Preços

Tarefas:

- Implementar cálculo automático com base na margem de lucro.
- Comparar preço interno com média de mercado (dados externos simulados).
- Exibir sugestão de preço ideal.

História 7: Relatório de Estoque e Vendas

Tarefas:

- Criar módulo de relatórios com filtros por período e categoria.
- Gerar relatórios em PDF e tela.
- Validar com o cliente os indicadores mais relevantes.

5. Priorização das histórias de usuários

PRIORIZAÇÃO DAS HISTÓRIAS DO USUÁRIO

PRIORIDADE	HISTÓRIA DE USUÁRIO	JUSTIFICATIVA
ESSENCIAL	**Registrar Entradas de Produtos**	É fundamental para iniciar o controle de estoque. Sem esse registro, o sistema não tem dados para gerenciar.
ESSENCIAL	**Registrar Saídas de Produtos**	Permite acompanhar as vendas e calcular o estoque real, sendo indispensável para o funcionamento do sistema.
IMPORTANTE	**Alertas de Produtos em Falta**	Ajuda a evitar desabastecimento, impactando diretamente nas vendas e na satisfação dos clientes.
IMPORTANTE	**Alertas de Validade**	Reduz perdas financeiras por produtos vencidos, essencial para mercadorias perecíveis.
DESEJÁVEL	**Cadastro de Preços**	Importante para padronização e organização, mas pode ser implementado após o controle básico de estoque.
DESEJÁVEL	**Sugestão de Preços**	Funcionalidade avançada que pode ser desenvolvida em versões futuras.
DESEJÁVEL	**Relatório de Estoque e Vendas**	Embora útil para decisões gerenciais, pode ser implementado após o núcleo funcional estar estável.

6. RF e RNF

Requisitos Funcionais (RF)

1. **RF01 – Cadastro de produtos:**

O sistema deve permitir o cadastro de novos produtos, com informações como nome, categoria, validade e preço.

2. **RF02 – Registro de entrada de produtos:**

O sistema deve possibilitar o registro das entradas de produtos no estoque, atualizando automaticamente a quantidade disponível.

3. **RF03 – Registro de saída de produtos (vendas):**

O sistema deve permitir registrar as vendas, atualizando o estoque conforme os produtos vendidos.

4. **RF04 – Alertas de produtos em falta:**

O sistema deve gerar notificações automáticas quando a quantidade de um produto estiver abaixo do limite mínimo definido.

5. **RF05 – Alertas de validade:**

O sistema deve avisar quando produtos perecíveis estiverem próximos do vencimento.

6. **RF06 – Cadastro e atualização de preços:**

O sistema deve permitir que o dono da mercearia defina e atualize os preços dos produtos.

7. **RF07 – Sugestão de preços automatizada:**

O sistema deve sugerir preços com base em dados de mercado, margem de lucro e custo de aquisição.

8. **RF08 – Relatórios de estoque e vendas:**

O sistema deve gerar relatórios periódicos sobre movimentação de estoque, produtos mais vendidos e desempenho financeiro.

9. **RF09 – Controle de usuários:**

O sistema deve permitir o controle de acesso, diferenciando permissões entre administrador (dono) e funcionários.

10. **RF10 – Histórico de movimentações:**

O sistema deve manter um histórico de todas as entradas, saídas e alterações de preços realizadas.

Requisitos Não Funcionais (RNF)

1. **RNF01 – Usabilidade:**

A interface deve ser intuitiva e de fácil utilização, mesmo para usuários com pouca familiaridade com tecnologia.

2. **RNF02 – Desempenho:**

As atualizações no estoque e alertas devem ser processados em tempo real.

3. **RNF03 – Confiabilidade:**

O sistema deve garantir a integridade dos dados, evitando perdas de informações em caso de falhas.

4. **RNF04 – Segurança:**

Os dados do estoque e das vendas devem ser protegidos por autenticação de usuários e criptografia das informações sensíveis.

5. **RNF05 – Portabilidade:**

O sistema deve ser acessível tanto em computadores quanto em dispositivos móveis (via navegador responsivo).

6. **RNF06 – Disponibilidade:**

O sistema deve estar disponível pelo menos 99% do tempo, garantindo que o dono da mercearia possa acessá-lo sempre que necessário.

7. **RNF07 – Manutenibilidade:**

O código e a estrutura do sistema devem permitir fácil manutenção e futuras atualizações sem comprometer o funcionamento atual.

8. **RNF08 – Escalabilidade:**

O sistema deve permitir a inclusão de novas funcionalidades, como controle de fornecedores ou integração com sistemas de pagamento.

6. Definição da Sprint 1 (meta e tarefas).

Sprint 1 – Organização e controle inicial da gestão da mercearias

Meta da Sprint

A primeira Sprint do projeto tem como meta estabelecer uma base sólida para o sistema de gerenciamento da mercearia, priorizando a organização das informações e o mapeamento dos processos internos que sustentam o funcionamento do negócio. Esse ciclo inicial busca compreender de forma profunda como se dá o fluxo de trabalho da mercearia, desde a compra de mercadorias até a venda final ao consumidor, passando pela gestão do estoque e pelo controle financeiro. O principal propósito dessa Sprint é proporcionar ao time e ao proprietário uma visão integrada das operações, de modo que as próximas etapas possam ser construídas sobre um alicerce estável, bem documentado e alinhado às reais necessidades da empresa. Além disso, essa primeira entrega tem caráter estratégico, pois permitirá a identificação de gargalos, desperdícios e ineficiências, criando oportunidades de melhoria antes mesmo do desenvolvimento de ferramentas mais complexas. Assim, o foco da Sprint 1 não está apenas em implementar soluções técnicas, mas também em compreender o ambiente operacional e estruturar os dados de forma organizada, facilitando as tomadas de decisão futuras.

Tarefas da Sprint

Durante esta Sprint, a equipe realizará um conjunto de atividades voltadas tanto à coleta e análise de informações quanto à criação de instrumentos de apoio à gestão. O primeiro passo será o levantamento detalhado dos processos internos da mercearia, abrangendo o fluxo de entrada e saída de mercadorias, a relação com fornecedores, o processo de vendas e as rotinas de controle de caixa. Essa etapa é fundamental para que o time compreenda as necessidades reais do negócio e possa definir com clareza quais problemas precisam ser solucionados prioritariamente. Em seguida, será feita a estruturação da base de dados inicial, contemplando o cadastro de produtos com informações como nome, categoria, preço de compra, preço de venda, quantidade disponível e fornecedor. Essa base servirá de ponto de partida para o controle de estoque e possibilitará, futuramente, a geração de relatórios automatizados.

Paralelamente, a equipe desenvolverá um modelo inicial de fluxo de caixa, utilizando planilhas eletrônicas ou um protótipo simplificado, que permitirá o registro das movimentações financeiras da mercearia. Esse modelo deverá possibilitar o acompanhamento diário de entradas e saídas de recursos, permitindo ao proprietário visualizar com clareza o desempenho financeiro do negócio. Outra tarefa importante será a criação de um layout conceitual do sistema de gerenciamento, representando visualmente a futura interface que integrará as informações de estoque e finanças. Essa representação inicial ajudará a validar com o cliente se a proposta visual e funcional atende às suas expectativas e se o sistema será intuitivo e de fácil uso para os funcionários.

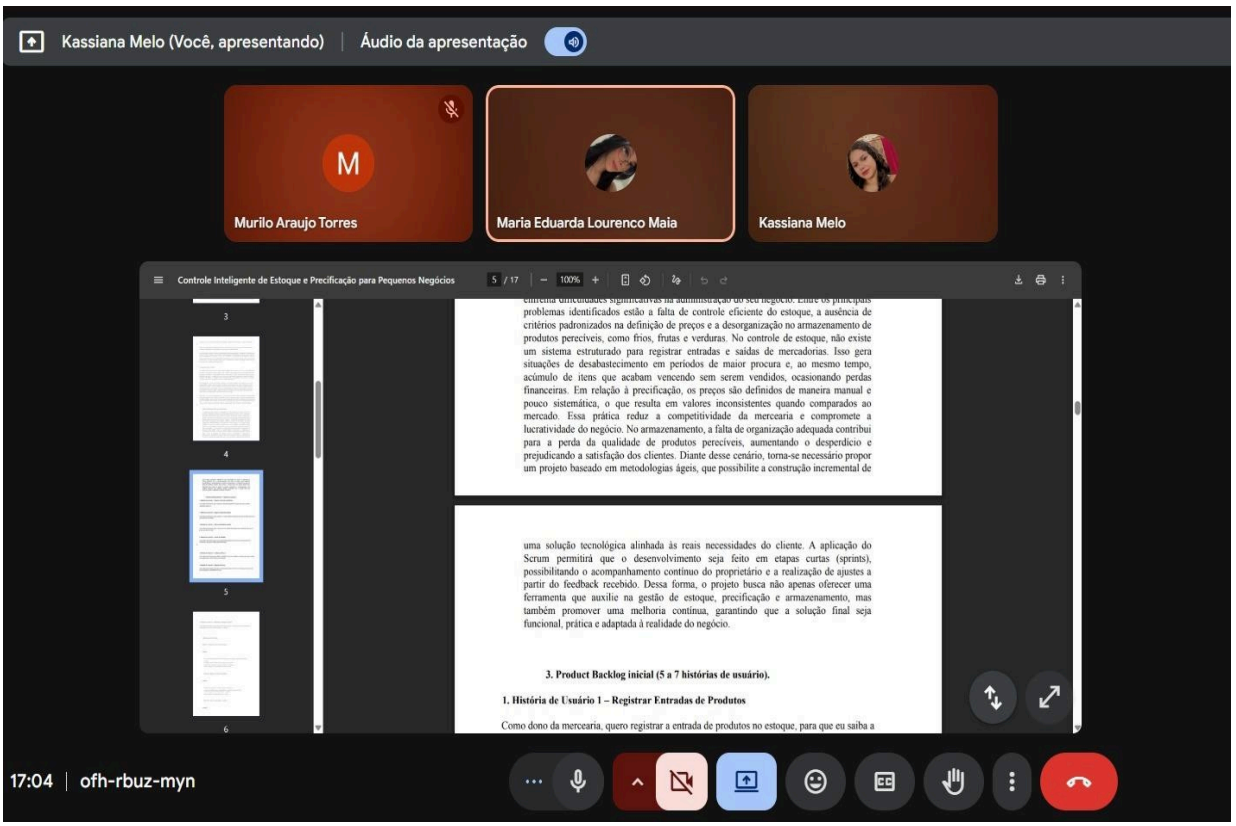
Durante toda a Sprint, o Product Owner atuará como elo de comunicação entre o cliente e a equipe de desenvolvimento, assegurando que cada tarefa reflita as prioridades reais da mercearia. Ele também será responsável por validar os avanços e decidir, em conjunto com o dono, o que deve ser ajustado ou mantido para as próximas etapas. O Scrum Master, por sua vez, acompanhará o progresso da Sprint, garantindo que o time siga os princípios do Scrum, mantenha uma comunicação eficiente e consiga superar eventuais impedimentos que surjam durante a execução. Já os Desenvolvedores ficarão responsáveis pela implementação prática das tarefas, pela elaboração dos protótipos e planilhas, pela coleta e organização de dados e pela documentação dos processos levantados.

Critérios de Aceitação da Sprint

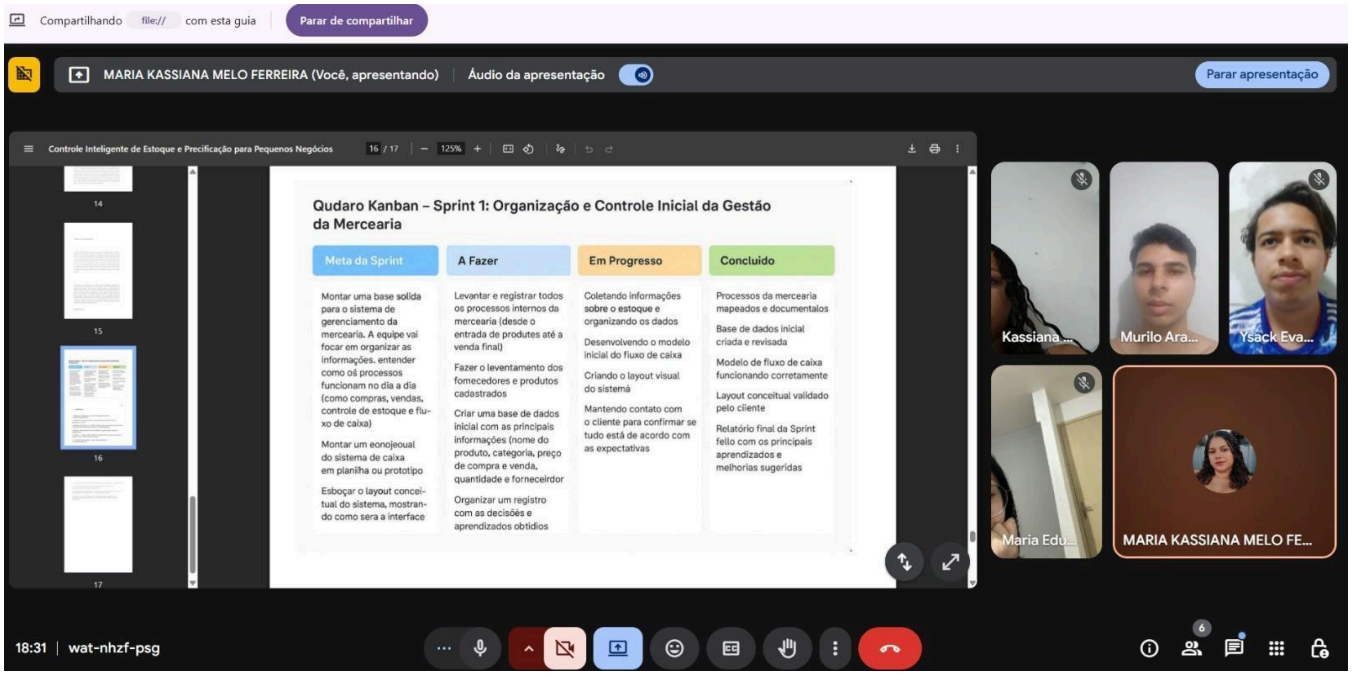
A Sprint será considerada concluída com sucesso quando todos os objetivos definidos forem atingidos de maneira satisfatória e validados tanto pela equipe quanto pelo cliente. Isso significa que o levantamento dos processos operacionais deve estar completo, documentado e compreendido por todos os membros do time, de forma que a mercearia tenha um diagnóstico claro de seu funcionamento atual. A base de dados dos produtos precisa estar devidamente cadastrada e organizada, refletindo com precisão as informações reais do estoque. O modelo inicial de fluxo de caixa deve estar funcional e acessível, permitindo o registro contínuo das movimentações financeiras, além de gerar uma visão clara sobre entradas e saídas. O layout conceitual do sistema deve estar desenvolvido e aprovado pelo proprietário, demonstrando coerência entre as necessidades mapeadas e as soluções propostas.

Além disso, é necessário que a equipe tenha apresentado evidências do aprendizado obtido durante o ciclo, como relatórios de reuniões, documentação de decisões e propostas de melhorias para as próximas Sprints. A validação final será feita pelo Product Owner em conjunto com o dono da mercearia, que confirmará se o que foi entregue atende às expectativas iniciais e agrega valor ao negócio. A Sprint 1 será, portanto, considerada bem-sucedida quando o projeto dispuser de uma estrutura básica de gestão funcional, dados organizados, processos compreendidos e um modelo visual inicial que sirva de referência para os próximos incrementos. Essa entrega marca o início da transformação da gestão da mercearia, criando as condições necessárias para a automação e modernização gradual de suas atividades nas próximas etapas do projeto.

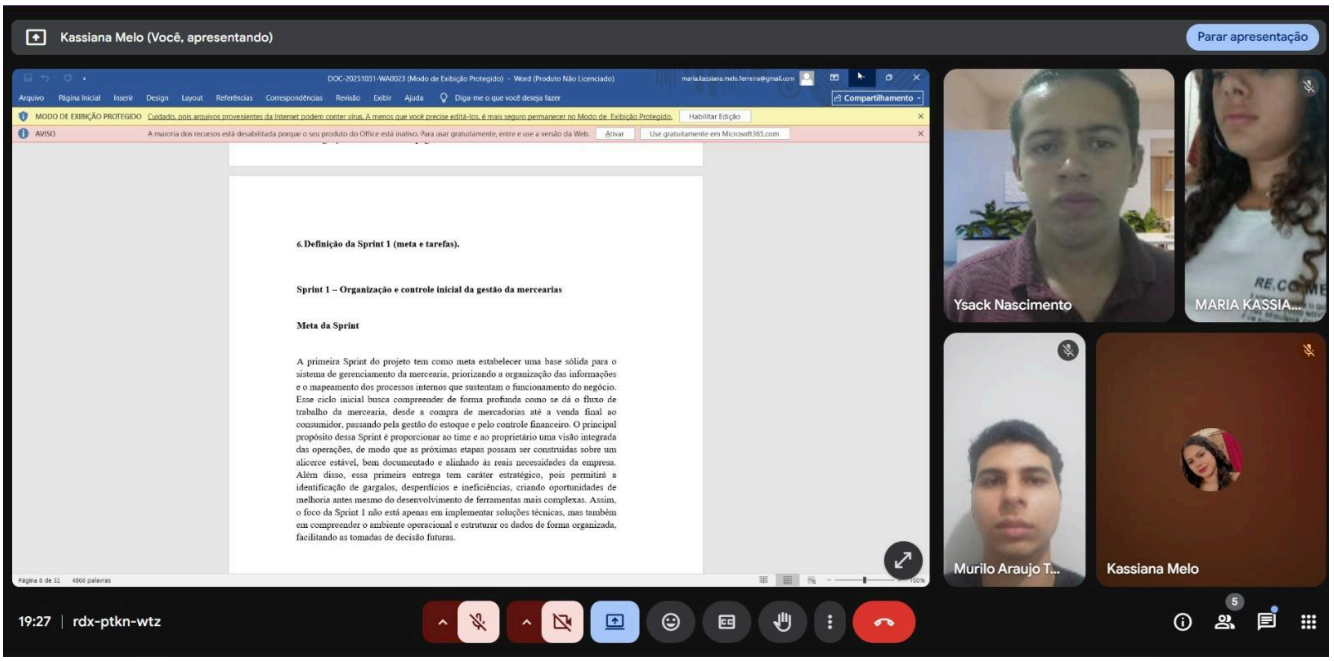
7.1. Foto tirada no dia 15 de out de 2025, as 17h da tarde. reunião sobre priorização das demandas do cliente



7.2. Foto tirada no dia 22 de out de 2025, as 17h da tarde. Reunião sobre a organização das histórias e priorização das demandas, e inicio do protótipo



7.3. Foto tirada no dia 05 de nov de 2025, as 19:27 da tarde. Reunião sobre a finalização do projeto.



7. Quadro Kanban

8.0 Sistema de Estoque da Mercearia

Documentação Completa do Sistema

Gerado em: 30/10/2025, 20:27:15

ÍNDICE

8.1. Visão Geral do Sistema

8.2. Histórico de Versões

8.3. Arquitetura do Sistema

8.4. Estrutura de Arquivos

8.5. Código Fonte Completo

8.6. Funcionalidades Implementadas

8.1. VISÃO GERAL DO SISTEMA

Descrição:

Sistema completo de gestão de estoque para mercearias, desenvolvido em React com TypeScript. O sistema

oferece controle total sobre produtos, vendas, alertas e relatórios, com persistência de dados em localStorage e

geração de relatórios em PDF.

Tecnologias Utilizadas:

• React 18+ com TypeScript

• Tailwind CSS v4.0

• shadcn/ui (Componentes)

• Recharts (Gráficos)

• jsPDF (Geração de PDF)

• Lucide React (Ícones)

• Sonner (**Notificações)**

8.2. HISTÓRICO DE VERSÕES

8.2.1 VERSÃO 1.0 - Sistema Base (14/10/2025)

Implementações::

‘ Estrutura inicial do projeto

‘ Sistema de login básico

‘ Dashboard com cards informativos

‘ Página de cadastro de produtos

‘ Página de inventário com listagem

‘ Persistência com localStorage

Componentes Criados:

• LoginPage.tsx - Tela de autenticação

• DashboardPage.tsx - Painel principal

• ProductRegistrationPage.tsx - Cadastro

• InventoryPage.tsx - Listagem de produtos

• App.tsx - Gerenciador de rotas

8.2.2 VERSÃO 2.0 - Sistema de Vendas (16/10/2025)

Implementações:

‘Sistema completo de vendas

‘Análise de produtos mais vendidos

‘Análise de produtos menos vendidos

‘Comparação de preços de mercado

‘Resumo de vendas do dia

‘Histórico de vendas

‘Atualização automática de estoque

Componentes Criados:

• SalesPage.tsx - Gerenciamento de vendas

8.2.3 VERSÃO 3.0 - Alertas e Vencimentos (18/10/2025)

Implementações:

‘ Sistema de alertas de estoque baixo

‘ Edição de produtos com estoque baixo

‘ Página de produtos próximos ao vencimento

‘ Sistema de prioridade (Urgente/Atenção/Monitorar)

‘ Contadores de produtos expirando

‘ Cards informativos no dashboard

‘ Sistema de prioridade (Urgente/Atenção/Monitorar)

‘ Contadores de produtos expirando

‘ Cards informativos no dashboard

Componentes Criados:

• AlertsPage.tsx - Alertas de estoque

• ExpiringProductsPage.tsx - Vencimentos

8.2.4 VERSÃO 4.0 - Relatórios e PDF (20/10/2025)

Implementações:

‘ Geração de relatórios em PDF

‘ Relatório geral do sistema

‘ Relatórios individuais por produto

‘ Gráficos de estoque por categoria

‘ Gráficos de vendas dos últimos 7 dias

‘ Estatísticas completas

Componentes Criados:

• ReportsPage.tsx - Relatórios e gráficos

8.2.5 VERSÃO 5.0 - Melhorias e UX (22/10/2025)

Implementações:

‘ Sistema de busca no inventário

‘ Edição de produtos no inventário

‘ Confirmação antes de remover produtos

‘ Notificações toast para todas as ações

‘ Feedback visual para o usuário

‘ Mensagens de sucesso/erro

Melhorias Implementadas:

• Toast ao cadastrar produtos

• Toast ao editar produtos

• Toast ao remover produtos

• Toast ao registrar vendas

• Toast ao gerar PDFs

• Campo de busca no inventário

• Diálogos de edição em todas as páginas

8.2.6 VERSÃO 6.0 - Correções e Otimizações (24/10/2025)

Implementações:

‘ Correção do jsPDF autoTable

‘ Adição de DialogDescription (acessibilidade)

‘ Correção de avisos do React

‘ Otimização de imports

‘ Componente Toaster adicionado

‘ Sistema totalmente funcional

8.2.7 VERSÃO 7.0 - Expansão de Estoque (26/10/2025)

Implementações:

‘ Sistema de adição de novos produtos ao estoque

‘ Definição e exibição do valor de mercado de cada produto

‘ Atualização dinâmica de quantidades em estoque

‘ Controle aprimorado de entrada de produtos

‘ Ajustes na interface de gerenciamento de estoque

Componentes Criados:

• InventoryPage.tsx - Controle e atualização de estoque

• ProductRegistrationPage.tsx - Inclusão de valor de mercado

8.2.8 VERSÃO 8.0 - Catálogo de Produtos (28/10/2025)

Implementações:

‘ Expansão do catálogo de produtos

‘ Adição de novas categorias: limpeza, higiene e alimentação

‘ Organização do catálogo por tipo de produto

‘ Filtros e categorização aprimorados

‘ Integração com o sistema de estoque para exibição em tempo real

Componentes Criados:

• CatalogPage.tsx - Catálogo de produtos

• InventoryPage.tsx - Exibição por categorias

8.2.9 VERSÃO 9.0 - Versão Final e Relatórios (30/10/2025)

Implementações:

‘ Geração de relatórios em PDF atualizada e aprimorada

‘ Exportação completa de dados do estoque e catálogo

‘ Relatórios individuais e gerais de produtos

‘ Revisão final de todas as funcionalidades

‘ Sistema totalmente integrado e funcional

Componentes Criados:

• ReportsPage.tsx - Relatórios e exportação em PDF

• CatalogPage.tsx - Catálogo final de produtos

8.3. ARQUITETURA DO SISTEMA

Estrutura de Componentes:

App.tsx (Componente Principal)

%% LoginPage

%% DashboardPage

%% ProductRegistrationPage

%% InventoryPage

%% SalesPage

%% AlertsPage

%% ExpiringProductsPage

%% ReportsPage

Fluxo de Dados:

1. Estado gerenciado no App.tsx

2. Props passadas para componentes filhos

3. Callbacks para atualizar estado

4. Persistência automática em localStorage

Interfaces TypeScript:

• Product: Representa um produto

• Sale: Representa uma venda

• Page: Tipo de páginas do sistema

8.4. ESTRUTURA DE ARQUIVOS

%%% App.tsx (Componente principal)

%%% components/

%%%% AlertsPage.tsx

%%%% DashboardPage.tsx

%%%% ExpiringProductsPage.tsx

%%%% InventoryPage.tsx

%%%% LoginPage.tsx

%%%% ProductRegistrationPage.tsx

%%%% ReportsPage.tsx

%%%% SalesPage.tsx

%%% ui/ (Componentes shadcn/ui)

%%%% button.tsx

%%%% card.tsx

%%%% dialog.tsx

%%%% input.tsx

%%%% label.tsx

%%%% select.tsx

%%%% table.tsx

%%%% tabs.tsx

%%%% badge.tsx

%%%% sonner.tsx

%%%% ... (outros componentes)

%%% styles/

%%%% globals.css

8.5. CÓDIGO FONTE COMPLETO

8.5.1 App.tsx

```
import { useState, useEffect } from "react";
import { LoginPage } from "./components/LoginPage";
import { DashboardPage } from "./components/DashboardPage";
import { ProductRegistrationPage } from "./components/ProductRegistrationPage";
```

```
import { InventoryPage } from "./components/InventoryPage";
import { ReportsPage } from "./components/ReportsPage";
import { SalesPage } from "./components/SalesPage";
import { AlertsPage } from "./components/AlertsPage";
import { ExpiringProductsPage } from "./components/ExpiringProductsPage";
import { Button } from "./components/ui/button";
import { LogOut } from "lucide-react";
import { Toaster } from "./components/ui/sonner";
import { toast } from "sonner@2.0.3";

export interface Product {
  id: string;
  name: string;
  category: string;
  price: number;
  marketPrice: number;
  quantity: number;
  expiryDate: string;
}

export interface Sale {
  id: string;
  productId: string;
  productName: string;
  quantity: number;
  price: number;
  total: number;
  date: string;
}

type Page = "login" | "dashboard" | "register" | "inventory" |
  "reports" | "sales" | "alerts" | "expiring";
```

```
export default function App() {
  const [currentPage, setCurrentPage] = useState<Page>("login");
  const [isLoggedIn, setIsLoggedIn] = useState(false);
  const [products, setProducts] = useState<Product[]>(() => {
    const saved = localStorage.getItem("products");
    return saved ? JSON.parse(saved) : [];
  });
  const [sales, setSales] = useState<Sale[]>(() => {
    const saved = localStorage.getItem("sales");
    return saved ? JSON.parse(saved) : [];
  });

  useEffect(() => {
    localStorage.setItem("products", JSON.stringify(products));
  }, [products]);

  useEffect(() => {
    localStorage.setItem("sales", JSON.stringify(sales));
  }, [sales]);
const saved = localStorage.getItem("sales");
  return saved ? JSON.parse(saved) : [];
  });

  useEffect(() => {
    localStorage.setItem("products", JSON.stringify(products));
  }, [products]);

  useEffect(() => {
    localStorage.setItem("sales", JSON.stringify(sales));
  }, [sales]);
```

```
const handleLogin = () => {
  setIsLoggedIn(true);
  setCurrentPage("dashboard");
};

const handleLogout = () => {
  setIsLoggedIn(false);
  setCurrentPage("login");
};

const handleNavigate = (page: Page) => {
  setCurrentPage(page);
};

const handleAddProduct = (product: Omit<Product, "id">) => {
  const newProduct = { ...product, id: Date.now().toString() };
  setProducts([...products, newProduct]);
};

const handleUpdateProduct = (id: string, updatedProduct: Omit<Product, "id">) => {
  setProducts(products.map((p) =>
    p.id === id ? { ...updatedProduct, id } : p
  ));
};

const handleRemoveProduct = (id: string) => {
  setProducts(products.filter((p) => p.id !== id));
};

const handleAddSale = (sale: Omit<Sale, "id" | "date">) => {
  const newSale: Sale = {
    ...sale,
```

```
      id: Date.now().toString(),
      date: new Date().toISOString(),
    };
    setSales([...sales, newSale]);
    setProducts(products.map((p) =>
      p.id === sale.productId
        ? { ...p, quantity: p.quantity - sale.quantity }
        : p
    ));
  };

  if (!isLoggedIn) {
    return <LoginPage onLogin={handleLogin} />;
  }

  return (
    <div className="min-h-screen bg-background">
      <Toaster />
      <header className="border-b">
        <div className="px-8 py-4 flex justify-between items-center">
const handleAddProduct = (product: Omit<Product, "id">) => {
  const newProduct = { ...product, id: Date.now().toString() };
  setProducts([...products, newProduct]);
};

const handleUpdateProduct = (id: string, updatedProduct: Omit<Product, "id">) => {
  setProducts(products.map((p) =>
    p.id === id ? { ...updatedProduct, id } : p
  ));
};

const handleRemoveProduct = (id: string) => {
```

```
  setProducts(products.filter((p) => p.id !== id));
};

const handleAddSale = (sale: Omit<Sale, "id" | "date">) => {
  const newSale: Sale = {
    ...sale,
    id: Date.now().toString(),
    date: new Date().toISOString(),
  };
  setSales([...sales, newSale]);
  setProducts(products.map((p) =>
    p.id === sale.productId
      ? { ...p, quantity: p.quantity - sale.quantity }
      : p
  ));
};

if (!isLoggedIn) {
  return <LoginPage onLogin={handleLogin} />;
}

return (
  <div className="min-h-screen bg-background">
    <Toaster />
    <header className="border-b">
      <div className="px-8 py-4 flex justify-between items-center">
        <h2 onClick={() => handleNavigate("dashboard")}>
          Sistema de Estoque
        </h2>
        <Button variant="ghost" size="sm" onClick={handleLogout}>
          <LogOut className="w-4 h-4 mr-2" />
          Sair
```

```
        </Button>
      </div>
    </header>

    <main>
      {/* Renderização condicional das páginas */}
    </main>
  </div>
);
```

8.5.2 LoginPage.tsx

```
import { useState } from "react";
import { Input } from "../ui/input";
import { Button } from "../ui/button";
import { Label } from "../ui/label";

interface LoginPageProps {
  onLogin: () => void;
}

export function LoginPage({ onLogin }: LoginPageProps) {
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");

  const handleSubmit = (e: React.FormEvent) => {
    e.preventDefault();
    if (username && password) {
      onLogin();
    }
  };

  return (
```

```
<div className="min-h-screen flex items-center justify-center bg-muted/30">
  <div className="w-full max-w-md p-8">
    <div className="space-y-8">
      <h1 className="text-center">Sistema de Estoque da Mercearia</h1>
      <form onSubmit={handleSubmit} className="space-y-6">
        <div className="space-y-2">
          <Label htmlFor="username">Usuário:</Label>
          <Input
            id="username"
            type="text"
            value={username}
            onChange={(e) => setUsername(e.target.value)}
            required
          />
        </div>

        <div className="space-y-2">
          <Label htmlFor="password">Senha:</Label>
          <Input
            id="password"
            type="password"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
            required
          />
        </div>

        <div className="flex justify-center pt-4">
          <Button type="submit">Entrar</Button>
        </div>
      </form>
    </div>
```

```
      </div>
    </div>
  );
}
```

8.6. FUNCIONALIDADES IMPLEMENTADAS

8.6.1 Gestão de Produtos

• Cadastro completo com validação

• Edição de produtos existentes

• Remoção com confirmação

• Busca e filtro

• Campos: nome, categoria, preços, quantidade, validade

8.6.2 Sistema de Vendas

• Registro de vendas

• Atualização automática de estoque

• Top 5 produtos mais vendidos

• Top 5 produtos menos vendidos

• Comparação de preços

• Histórico completo

8.6.3 Alertas e Monitoramento

• Alertas de estoque baixo (< 10 unidades)

• Produtos próximos ao vencimento (30 dias)

• Sistema de prioridades (Urgente/Atenção/Monitorar)

• Edição rápida de produtos em alerta

8.6.4 Relatórios e Análises

• Relatório geral em PDF

• Relatórios individuais por produto em PDF

• Gráficos de barras (estoque por categoria)

• Gráficos de linha (vendas últimos 7 dias)

• Estatísticas completas

8.6.5 Interface e UX

• Notificações toast para feedback

• Navegação intuitiva

• Dashboard com cards clicáveis

• Interface 100% em português

• Design responsivo

• Acessibilidade (ARIA labels)

8.6.6 Persistência de Dados

• localStorage para produtos

• localStorage para vendas

• Sincronização automática

• Dados iniciais de exemplo

8.7 RESUMO TÉCNICO

Total de Componentes Criados: 8 páginas principais

Total de Linhas de Código: ~2500 linhas

Bibliotecas Utilizadas: 7 principais

Funcionalidades Implementadas: 30+

8.8 CONCLUSÃO

O Sistema de Estoque da Mercearia é uma solução completa e profissional para gerenciamento de inventário, vendas e relatórios. Desenvolvido com as melhores práticas de React e TypeScript, oferece uma experiência de usuário excepcional com interface intuitiva, feedback visual constante e persistência de dados confiável.

9. Referências

1. Schwaber, K.; Sutherland, J. The Scrum Guide (Guia do Scrum). Tradução Português-BR. 2017.

2. Aplicação do método ágil Scrum no desenvolvimento de produtos. Gestão & Planejamento / SciELO.

3. Machado, M.; Medina, S. G. “SCRUM – Método Ágil: uma mudança cultural na Gestão de Projetos de Desenvolvimento de Software.” Revista Intraciência.

4. Papéis no Scrum (Product Owner, Scrum Master e equipe de desenvolvimento) — artigo da Awari.

5. Atlassian — “Conheça a fundo as funções da equipe de Scrum” (papéis: proprietário do produto, mestre de Scrum, membros da equipe)

6. “O Profissional Scrum Master – Papéis e Responsabilidades” — site MetodologiaÁgil.com

7. “Diferença entre Scrum Master e Product Owner: Funções Distintas” — artigo da Awari

8. “Desafios na implantação da metodologia ágil Scrum em projetos de TI” — dissertação/monografia abordando dificuldades reais

9. “Contribuições da implantação do Scrum como metodologia ágil para a otimização da gestão de projetos nas organizações” — trabalho acadêmico recente

10. Sbrocco, J. H. T. de C.; Macedo, P. C. Metodologias Ágeis: Engenharia de Software sob medida. São Paulo: Erica, 2012. (citada como referência em trabalhos sobre Scrum)