RELEASE 2 (01/04 - 14/05)

Model e Controller dos requisitos:

Requisito 5: Realizar Vendas de Produtos.

Requisito 6: O comprador possui um histórico de pedidos/compras.

Tecnologias utilizadas:

1. Linguagem de programação: Java

2. Estruturação: Arquitetura MVC

3. Persistência de dados: JSON

Sprint 3 (01/04 - 14/04) - Gerente: LAURA BARBOSA VASCONCELOS (

Atribuições de DANILO PEDRO DA SILVA VALERIO

Carrinho de compras:

- Task 1.0 - adicionar atributo "carrinho de compras" na classe Comprador.

Histórico de compras:

- Task 3.0 - adicionar atributo "histórico de vendas" na classe Comprador.

Relacionamento entre lojas e vendas:

- Task 4.0 - criar método para adicionar venda ao histórico na classe LojaController.

Testes:

- Task 6.0 - testes unitários para a classe Comprador.
- Task 6.3 - testes unitários para a classe Loja.

Documentação:

- Registrar todas as atualizações e melhorias realizadas no sistema.

Atribuições de LAURA BARBOSA VASCONCELOS:

Carrinho de compras:

- Task 1.0 - adicionar atributo "carrinho de compras" na classe Comprador
- Task 1.2 - criar método de remover produtos do carrinho na classe Comprador

Testes:

- Task 6.4 - testes de integração entre Comprador VendaController
- Task 6.5 - testes de integração entre VendaController e Loja

Diagrama de Classe:

- Atualizar o Diagrama com as novas classes, métodos e relacionamentos.

Organização do azure:

- Manter o Azure bem estruturado e gerenciar as atividades desenvolvidas

Atribuições de LETICIA MARIA CAMPOS DE MEDEIROS

Processo de venda:

- Task 2.0 - criar classe Venda
- Task 2.1 - criar classe VendaController

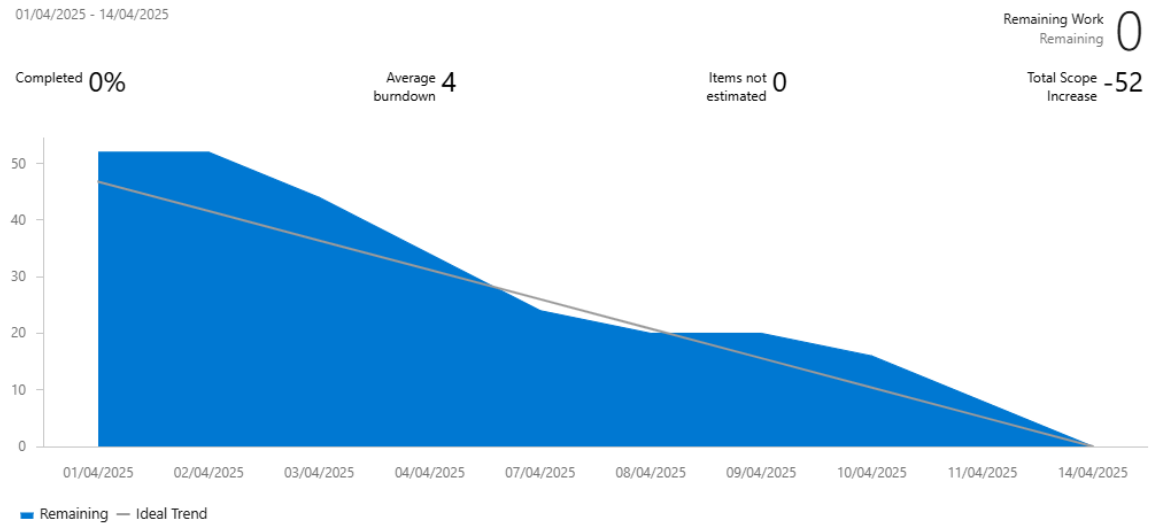
Testes:

- Task 6.1 - testes unitários para a classe VendaModel
- Task 6.2 - testes unitários para a classe VendaController

Documentação:

- Documentar o processo de desenvolvimento e atribuições de cada integrante da equipe

Gráfico de burndown (Sprint 3):



Sprint 4 (14/04 -14/05) - Gerente: DANILO PEDRO DA SILVA VALERIO

Atribuições de DANILO PEDRO DA SILVA VALERIO :

Relacionamento lojas e vendas:

- Task 5.0 - Criar método para loja visualizar histórico de pedidos para lojas

Carrinho de compras:

- Task 1.2 - Adicionar no front-end opção de remover produto no carrinho

Testes:

- Task 4.0 - Testar visualização de histórico de compras

Documentação:

- Task 6.0 - Documentar funcionalidade do histórico de pedidos da loja
- Task 6.4 - Criar novo Diagrama de Sequência para o fluxo completo de venda

Atribuições de LAURA BARBOSA VASCONCELOS :

Histórico de compras:

- Task 4.0 - Criar método para o comprador visualizar seu histórico de compras

Carrinho de compras:

- Task 1.1 - Adicionar no front-end opção de colocar produto no carrinho

Testes:

- Task de Teste 1.0 - Testar funcionalidades do menu de carrinho

Documentação:

- Task 6.2 - Documentar funcionalidade do carrinho de compras
- Task 6.5 - Atualizar Diagrama de Casos de Uso com os novos fluxos

Atribuições de LETICIA MARIA CAMPOS DE MEDEIROS :

Processo de venda:

- Task 3.0 - Criar método na view para realizar venda após o pedido ser finalizado

Carrinho de compras:

- Task 1.0 - Criar menu de carrinho no menu de produtos para comprador, com opção de adicionar, remover e finalizar pedido

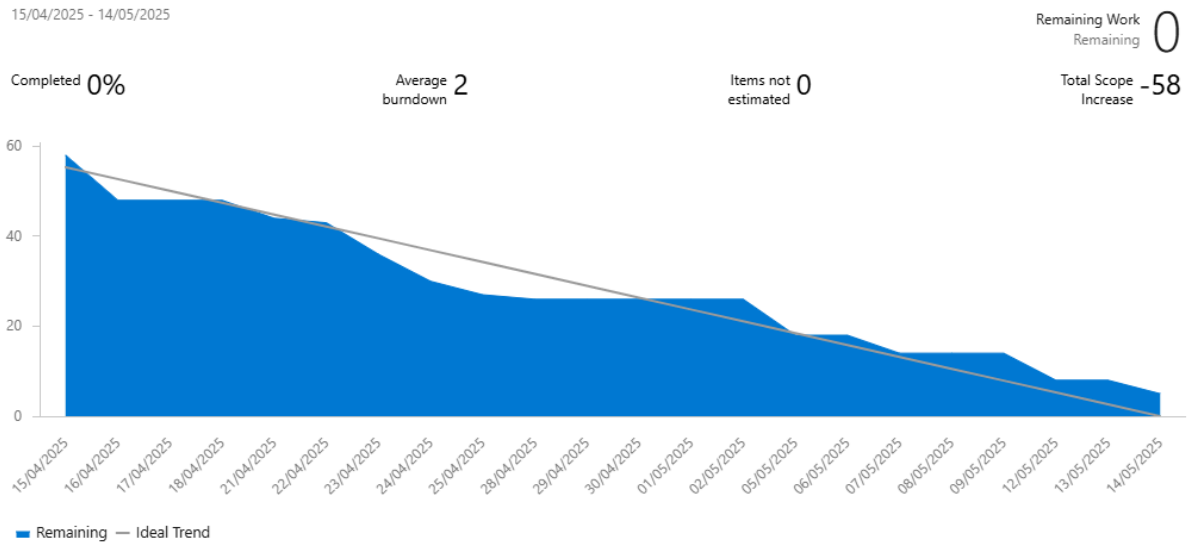
Testes:

- Task de Teste 3.0 - Testar processo de finalização de pedido

Documentação:

- Task 6.1 - Documentar o processo de vendas
- Task 6.3 - Atualizar Diagrama de Classes com os novos métodos

Gráfico de burndown (Sprint 4):



1. Carrinho de Compras no Comprador

Atributo Carrinho na Classe Comprador

Foi adicionado um novo atributo chamado carrinho na classe Comprador, responsável por armazenar temporariamente os produtos que o comprador deseja adquirir. Esse atributo é uma lista de objetos, cada um contendo os dados do produto e a quantidade desejada. A estrutura foi desenvolvida para permitir inclusão, modificação e remoção de itens, garantindo flexibilidade na composição da compra. O atributo pode ser acessado e manipulado por métodos da própria classe Comprador, assegurando encapsulamento e organização no código.

Método adicionarAoCarrinho(produto, quantidade)

Foi criado o método adicionarAoCarrinho para permitir que produtos sejam adicionados ao carrinho. O método realiza a verificação se o produto já existe no carrinho: se sim, atualiza a quantidade somando ao valor existente; se não, adiciona o produto com a quantidade informada. Além disso, impede a inserção de quantidades inválidas (zero ou

negativas). Após a adição, os dados atualizados do comprador são persistidos no arquivo compradores.json.

Método removerDoCarrinho(produto)

O método removerDoCarrinho possibilita a remoção de produtos previamente adicionados ao carrinho. Caso o produto esteja presente, ele é removido da lista; caso contrário, uma mensagem de erro é exibida. A ação também atualiza o arquivo compradores.json, garantindo que o estado do carrinho seja refletido corretamente nos dados persistidos.

Histórico de Compras/Pedidos

Além disso, a criação do vínculo de vendas com compradores e lojas permitiram a rastreabilidade completa das transações. Agora, cada compra realizada gera um registro detalhado que é incluído tanto no histórico da loja quanto no histórico do comprador. Isso possibilita ao usuário visualizar todas as suas compras anteriores, promovendo transparência, confiança e controle sobre seus pedidos. Essas implementações garantem o atendimento aos requisitos funcionais relacionados à realização de compras, gerenciamento do carrinho e visualização de históricos, consolidando uma base sólida para a expansão futura do sistema.

2. Venda

Classe VendaModel

Foi criada a classe VendaModel para representar uma venda no sistema. Essa classe contém os seguintes atributos: data da venda, lista de IDs dos produtos vendidos, valor unitário de cada produto, quantidade de cada item e valor total da venda. Um construtor inicializa todos os atributos corretamente, e métodos getters e setters foram implementados para permitir acesso e modificação controlada dos dados. Essa estrutura serve como base para o armazenamento das transações realizadas entre compradores e lojas.

Classe VendaController e Histórico de Vendas/Compras

A classe VendaController foi desenvolvida para gerenciar as operações de venda, incluindo o registro e armazenamento das transações no sistema. Com isso, foi implementado o **histórico de compras** para o comprador e o **histórico de vendas** para a loja, criando um vínculo entre as entidades envolvidas em cada venda. Cada venda realizada é registrada com

seus detalhes completos e associada ao comprador e à loja correspondente, permitindo que ambos visualizem o histórico de transações diretamente em seus perfis, a partir dos dados armazenados nos arquivos compradores.json, lojas.json e vendas.json.

Com as funcionalidades implementadas no back-end até o momento, o sistema evoluiu significativamente em relação à experiência do comprador dentro do marketplace. A introdução do carrinho de compras possibilitou que o usuário armazenasse produtos temporariamente antes de finalizar uma compra, oferecendo flexibilidade e controle sobre seus pedidos. A modelagem desse recurso foi pensada para garantir consistência nos dados, com validações de entrada e persistência adequada em arquivos JSON (Figura 1), mantendo a integridade da aplicação mesmo em operações repetidas ou simultâneas.


```
[ {
  "id_comprador" : "12345678900",
  "data" : "2025-04-08",
  "produtos" : [ [ "P001", 2, 10.0 ] ],
  "id_venda" : "V01",
  "id_loja" : "123.456.789-10",
  "id_vendaGeral" : "VTESTE",
  "valor total" : 20.0
}, {
  "id_comprador" : "12345678900",
  "data" : "2025-04-08",
  "produtos" : [ [ "P002", 1, 20.0 ] ],
  "id_venda" : "V02",
  "id_loja" : "123.456.789-12",
  "id_vendaGeral" : "VTESTE",
  "valor total" : 20.0
}, {
  "id_comprador" : "12345678900",
  "data" : "2025-04-08",
  "produtos" : [ [ "P003", 3, 15.0 ] ],
  "id_venda" : "V03",
  "id_loja" : "123.456.789-11",
  "id_vendaGeral" : "VTESTE",
  "valor total" : 45.0
} ]
```


Figura 1: Exemplo de vendas no arquivo Json.

Note que essa é uma venda só, todavia, como uma venda pode ter produtos de lojas diferentes, salvamos o arquivo json com subvendas. Note que o atributo “id_vendaGeral” é igual para todas as compras pertencentes a uma única venda e o que muda é o restante que

seria o id da loja do produto em específico. Dessa forma, garantimos um ID único para cada venda mesmo que incluam produtos de lojas diferentes.

3. Histórico de Vendas para lojas

O sistema de histórico de vendas foi desenvolvido para permitir que lojas acompanhem todas as transações realizadas, garantindo rastreabilidade e gestão eficiente de pedidos. Quando um comprador finaliza uma compra, o sistema não apenas registra a venda no perfil do cliente, mas também identifica automaticamente as lojas envolvidas e atualiza seus respectivos históricos.

A classe responsável por gerenciar esse processo assegura que cada venda seja armazenada de forma organizada no arquivo vendas.json, contendo informações essenciais como ID único, data da compra, detalhes do comprador, produtos adquiridos e status do pedido. Uma particularidade importante é o tratamento de vendas que envolvem múltiplas lojas: como um único pedido pode conter itens de diferentes estabelecimentos, o sistema divide a transação em subvendas, mantendo um id_vendaGeral comum para agrupar todos os itens pertencentes à mesma compra. Dessa forma, mesmo que um pedido inclua produtos de diversas lojas, cada uma receberá apenas as informações relevantes aos seus itens, sem perder a referência ao contexto completo da venda.

Assim que uma venda é concluída, o sistema percorre a lista de produtos adquiridos e, para cada um deles, recupera o ID da loja responsável. Em seguida, atualiza o campo historicoVendas no arquivo lojas.json, adicionando o ID da transação correspondente. Isso permite que cada estabelecimento visualize apenas suas próprias vendas, mantendo a organização e evitando sobreposição de dados.

Testes

Para garantir a qualidade e o funcionamento correto das novas funcionalidades, foram realizados testes complementares aos já existentes desde a release anterior, além da inserção de novos arquivos de teste próprios para as classes Venda e VendaController. A base de testes

já cobria os principais fluxos do sistema, como cadastro e visualização de lojas e compradores, o que permitiu focar diretamente nos novos métodos adicionados nesta etapa.

Foram implementados testes específicos (Figura 2) para validar o comportamento do carrinho de compras, incluindo a adição, atualização e remoção de produtos. Também foram testadas as condições de erro, como tentativa de adicionar quantidade inválida ou remover produtos inexistentes do carrinho. Além disso, testou-se o processo de criação de vendas e o correto armazenamento nos históricos de compradores e lojas. Esses testes asseguram que as novas funcionalidades estão integradas ao sistema de forma estável e coerente com os requisitos definidos.

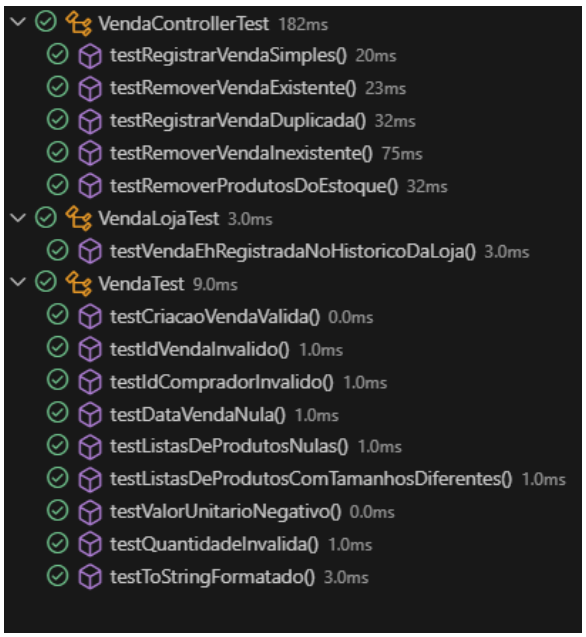
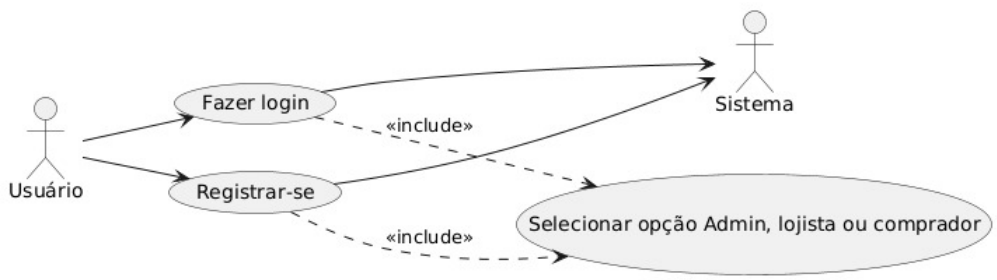


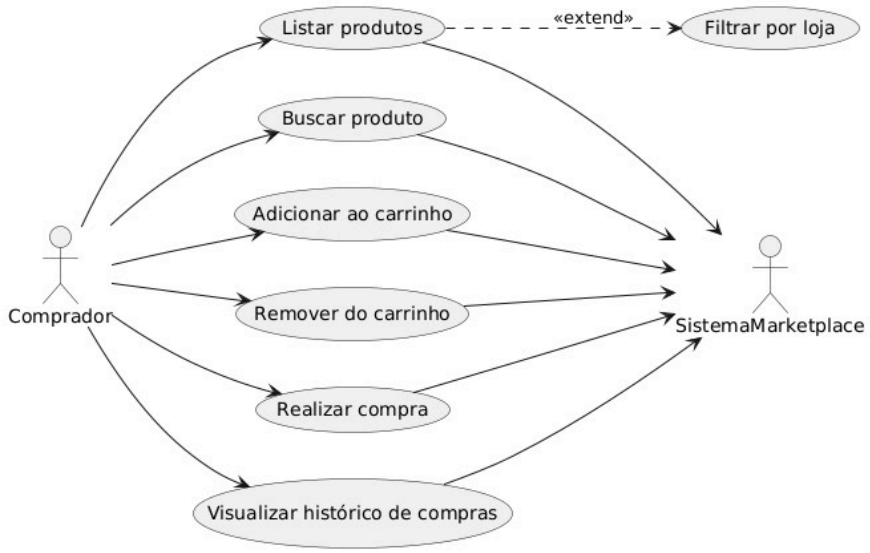
Figura 2: Testes implementados

Diagrama Caso de Uso

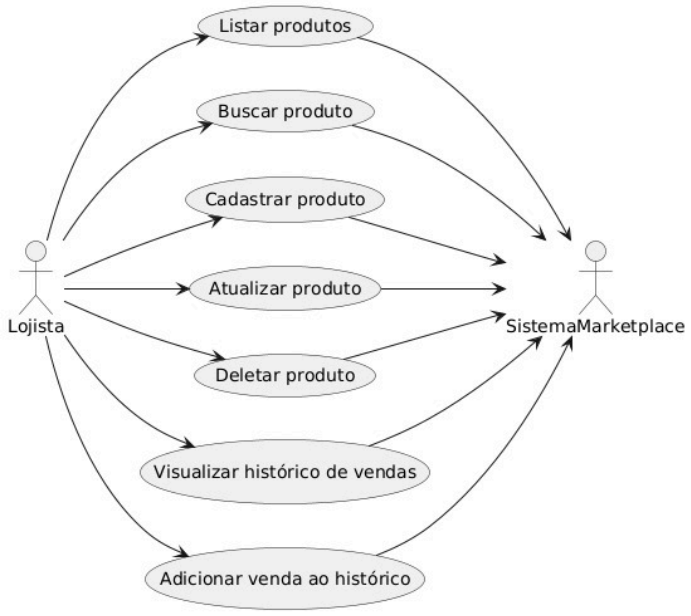
Usuário:



Comprador:



Lojista:



Administrador:

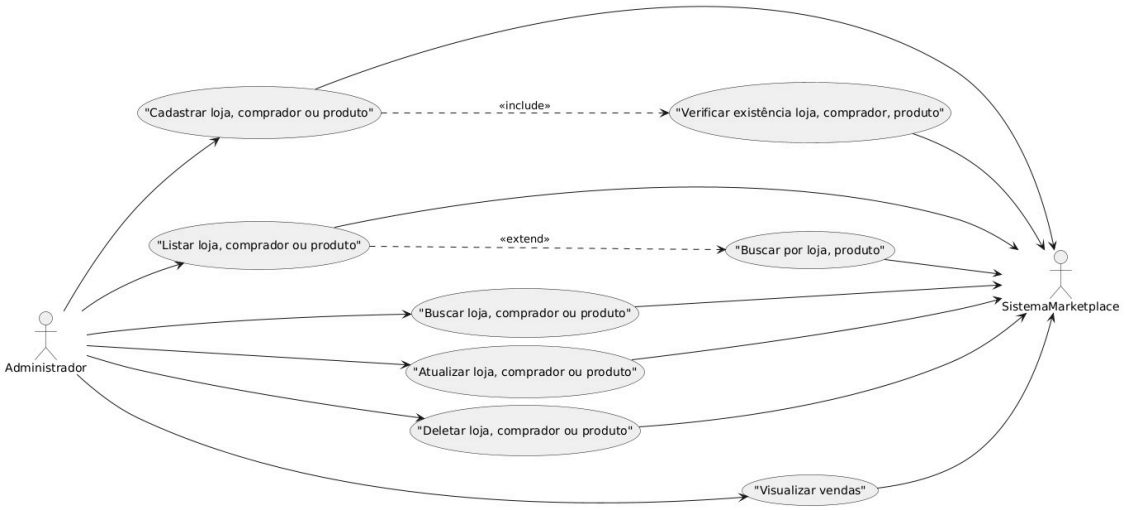


Diagrama de Classes

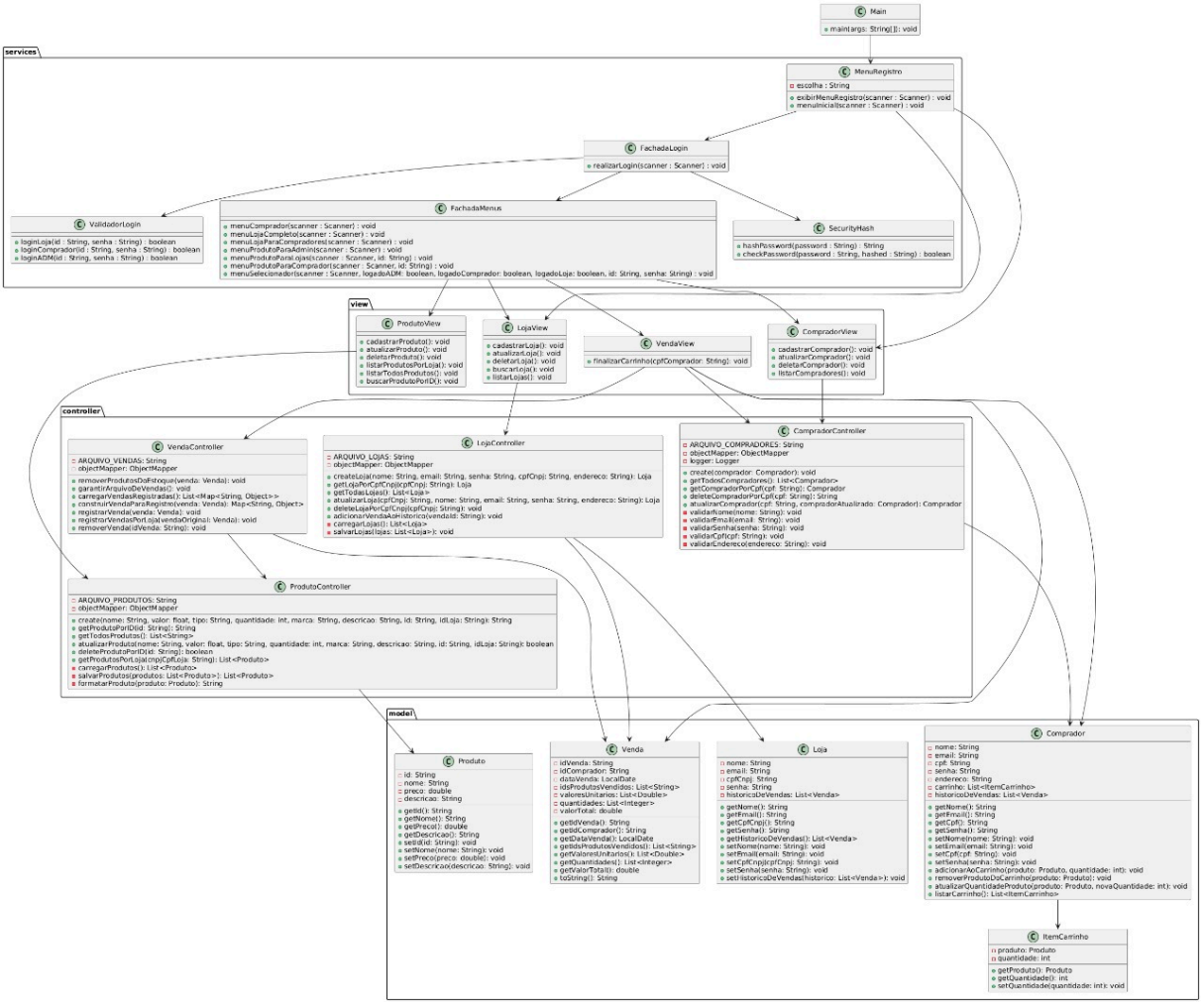
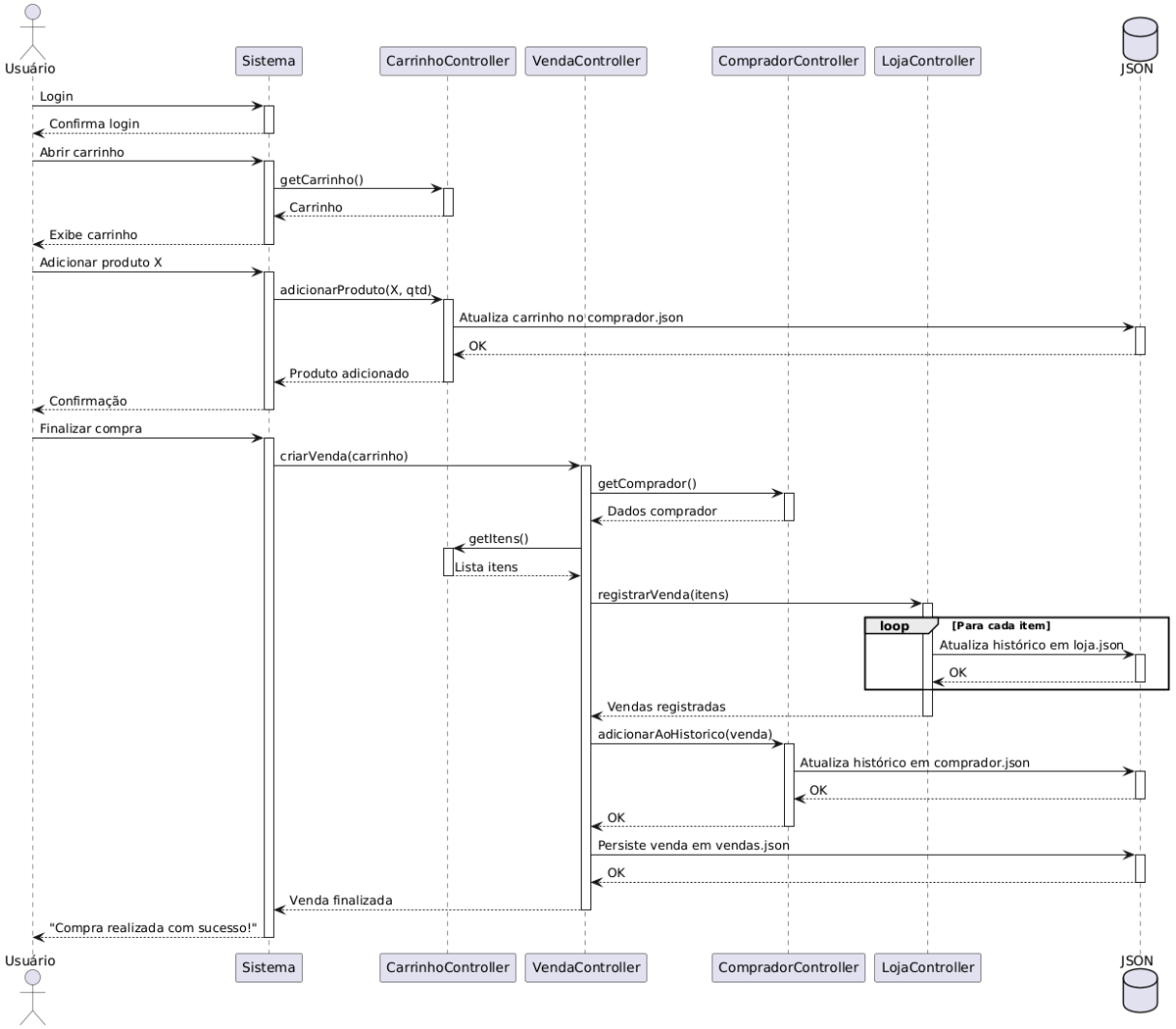


Diagrama de Sequência: Fluxo de venda



Resumo da Release 2 (01/04 a 14/05)

O período de desenvolvimento da Release 2 foi dedicado à implementação dos requisitos 5 (Realizar Vendas de Produtos) e 6 (Histórico de pedidos/compras do comprador). A equipe, composta por Danilo Pedro da Silva Valerio, Laura Barbosa Vasconcelos e Leticia Maria Campos de Medeiros, trabalhou em duas sprints consecutivas (Sprint 3: 01/04-14/04 e Sprint 4: 14/04-14/05), utilizando novamente Java como linguagem de programação, arquitetura MVC e persistência de dados em formato JSON.

Durante a Sprint 3, gerenciada por Laura Barbosa, foram criadas as estruturas fundamentais para o sistema de vendas, incluindo a implementação do carrinho de compras na classe Comprador, desenvolvimento da classe Venda e VendaController, e adição do histórico de

vendas/compras. Paralelamente, foram realizados testes unitários para validar o funcionamento das novas funcionalidades.

Na Sprint 4, sob a gerência de Danilo Pedro, o foco foi aprimorar o andamento do desenvolvimento do sistema com a inserção das views. Foram implementados métodos para visualização do histórico de pedidos tanto para compradores quanto para lojas, desenvolvimento de interfaces para gerenciamento do carrinho (adicionar/remover produtos) e finalização de compras, além de atualizações na documentação e nos diagramas do sistema.

O carrinho de compras foi implementado como um atributo na classe Comprador, permitindo o armazenamento temporário de produtos antes da finalização da compra. Foram criados métodos para adicionar e remover itens, com verificações para garantir a integridade dos dados. O sistema foi projetado para lidar com situações como a adição de produtos já existentes no carrinho e a validação de quantidades.

O processo de vendas foi estruturado através da classe VendaModel, que armazena informações como data, produtos, valores e quantidades, e da classe VendaController, responsável por gerenciar as transações. Uma característica importante implementada foi o tratamento de vendas envolvendo produtos de múltiplas lojas, onde o sistema cria "subvendas" associadas por um ID comum, permitindo que cada loja acesse apenas as informações pertinentes a seus produtos.

O histórico de compras/vendas foi implementado para ambos os atores do sistema: compradores podem visualizar todas as suas compras anteriores, enquanto lojas têm acesso ao registro completo de suas vendas. Os dados são persistidos em arquivos JSON (compradores.json, lojas.json e vendas.json), garantindo a rastreabilidade das transações.

Para assegurar a qualidade do software, foram realizados testes abrangentes das novas funcionalidades, incluindo testes unitários para as classes Venda, VendaController, Comprador e Loja, além de testes de integração entre os componentes do sistema. A documentação foi atualizada com diagramas de casos de uso, classes e sequência, refletindo as novas implementações.

O desenvolvimento da Release 2 cumpriu os objetivos estabelecidos, criando uma base sólida para a realização de vendas e o acompanhamento das transações, elementos fundamentais para o funcionamento eficaz do marketplace.