RELEASE 2 - SPRINT 3 (01/04 - 14/04)

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

Processo de desenvolvimento e atribuições:

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

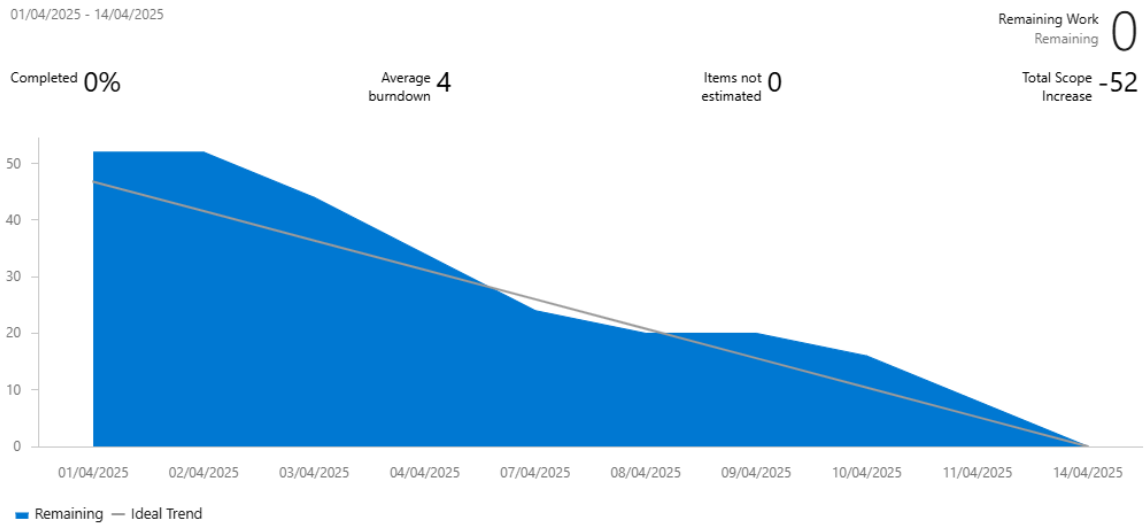
Testes:

- 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:



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.

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.

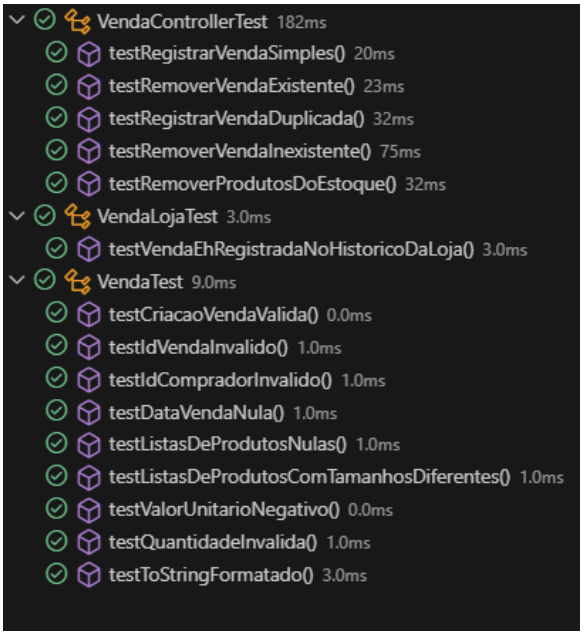


Figura 2: Testes implementados