RELEASE 1

Tecnologias utilizadas:

1. Linguagem de programação: Java
2. Estrutura do projeto:
- Arquitetura MVC
- Padrão Fachada

3. Bibliotecas e APIs: BCrypt
4. Persistência de dados: JSON

Processo de desenvolvimento e atribuições

Organização da equipe - Release 1 (25/02 - 31/03)

Sprint 1 - Gerente: DANILO PEDRO DA SILVA VALERIO

Duração: 25/02 - 17/03

Atribuições de DANILO PEDRO DA SILVA VALERIO :

1. Project Resources - criação e configuração do repositório do projeto
2. Organização de PBIs e tasks e suas respectivas distribuições para os integrantes do grupo
3. CRUD de Lojas - backend (Model e Controller)
4. Testes de unidade para as classes do CRUD de Lojas

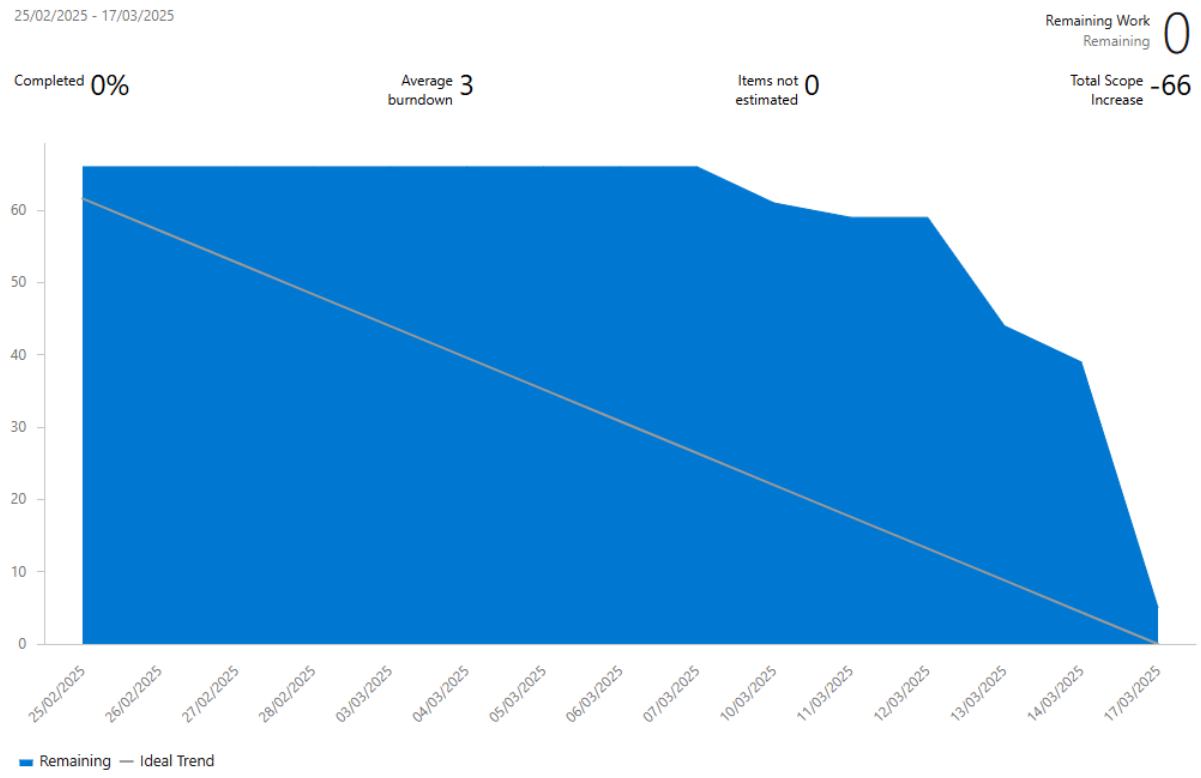
Atribuições de LAURA BARBOSA VASCONCELOS :

1. CRUD de Compradores - backend (Model e Controller)
2. Testes de unidade para as classes do CRUD de Compradores

Atribuições de LETICIA MARIA CAMPOS DE MEDEIROS :

1. CRUD de Produtos - backend (Model e Controller)
2. Testes de unidade para as classes do CRUD de Produtos

Gráfico de Burndown da Sprint 1:



Sprint 2 - Gerente: LETICIA MARIA CAMPOS DE MEDEIROS

Duração: 18/03 - 31/03

Atribuições de DANILO PEDRO DA SILVA VALERIO:

1. CRUD de Lojas - frontend (View)
2. CRUD de Produtos - frontend (View): tasks 3.3 e 3.4
3. Controle de acesso - menu principal - DevOps: tasks 0.2, 0.3 e 0.4
4. Testes de integração: tasks 4.1, 4.2 e 4.3

Atribuições de LAURA BARBOSA VASCONCELOS:

1. CRUD de Compradores - frontend (View)
2. Controle de acesso - menu principal - DevOps: tasks 0.1 e 0.5
3. Testes de integração: tasks 4.4, 4.5, 4.6, 4.7, 4.8 e 4.9
4. Project Resourses: task 5.1 - criação do diagrama de sequência para gerenciamento de lojas, compradores e produtos.

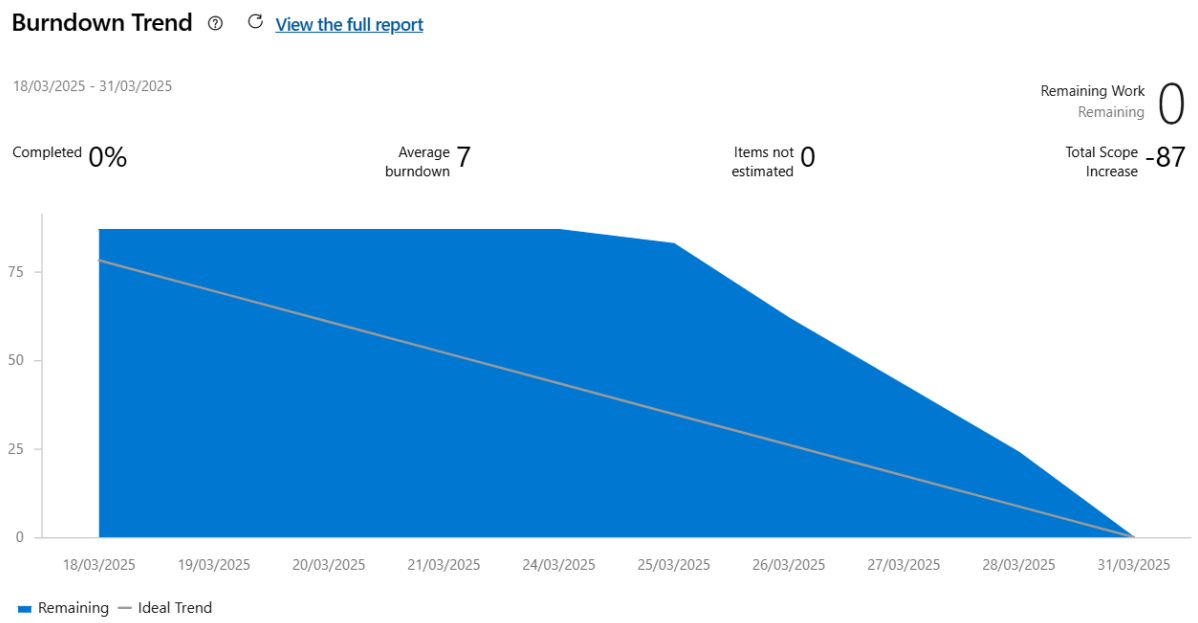
Atribuições LETICIA MARIA CAMPOS DE MEDEIROS :

1. CRUD de produtos - frontend (View): tasks 3.1 e 3.2

2. Organização de PBIs e tasks e suas respectivas distribuições para os integrantes do grupo

3. Project Resources - documentação da release 1 como um todo; em relação a task 5.1, foi responsável pela criação de diagramas de sequência para login.

Gráfico de BurnDown da Sprint 2:



Sprint 1 - Registro de funcionalidades de desenvolvimento do Model e Controller

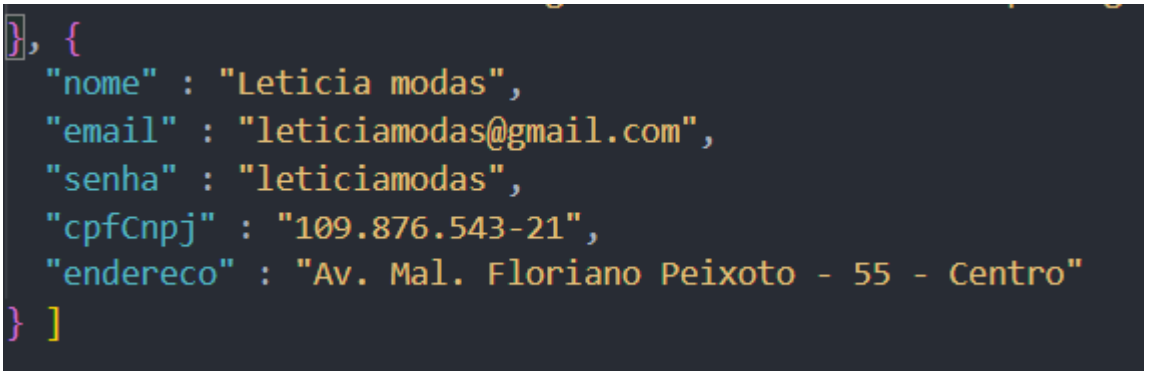
1. CRUD de Lojas

Cadastro de Lojas

Foi desenvolvido um sistema para cadastrar lojas no marketplace, com foco na validação dos dados e no armazenamento seguro das informações em um arquivo JSON. Para o cadastro, são solicitados os seguintes dados: nome da loja, que não pode ser vazio; e-mail, que deve conter "@" e ".com"; senha, com um mínimo de seis caracteres; CNPJ, que deve incluir os caracteres especiais (traço e ponto); e endereço, que deve ser escrito em uma única linha, com as informações separadas por hífen. A validação desses dados é realizada na classe Loja por meio de expressões regulares (regex), garantindo que apenas informações no formato correto sejam aceitas.

A classe LojaController gerencia o processo de cadastro e assegura a persistência dos dados no arquivo lojas.json O método createLoja realiza as seguintes etapas: carrega a lista de lojas já cadastradas, verifica se o CNPJ já existe para evitar duplicações, adiciona a nova loja à lista e salva os dados no arquivo JSON, retornando uma confirmação de sucesso no cadastro. Esse mecanismo garante a integridade dos cadastros e mantém a organização das informações no sistema.

Loja salva no arquivo lojas.json após o cadastro:



Visualização de lojas

A funcionalidade de visualização de lojas foi implementada para permitir a exibição de todas as lojas cadastradas ou a consulta detalhada de uma loja específica, incluindo seu nome e CNPJ.

A classe Loja disponibiliza métodos **getters**, que permitem acessar individualmente os atributos de um objeto loja. Além disso, o método toString foi sobrescrito para fornecer uma representação legível da loja ao ser exibida.

Já a classe LojaController é responsável por recuperar e exibir os dados das lojas registradas no sistema. O método getTodasAsLojas() retorna uma lista contendo todas as lojas cadastradas, organizadas alfabeticamente pelo nome. Para isso, ele primeiro carrega os dados do arquivo JSON, realiza a ordenação das lojas e, por fim, retorna a lista formatada para exibição.

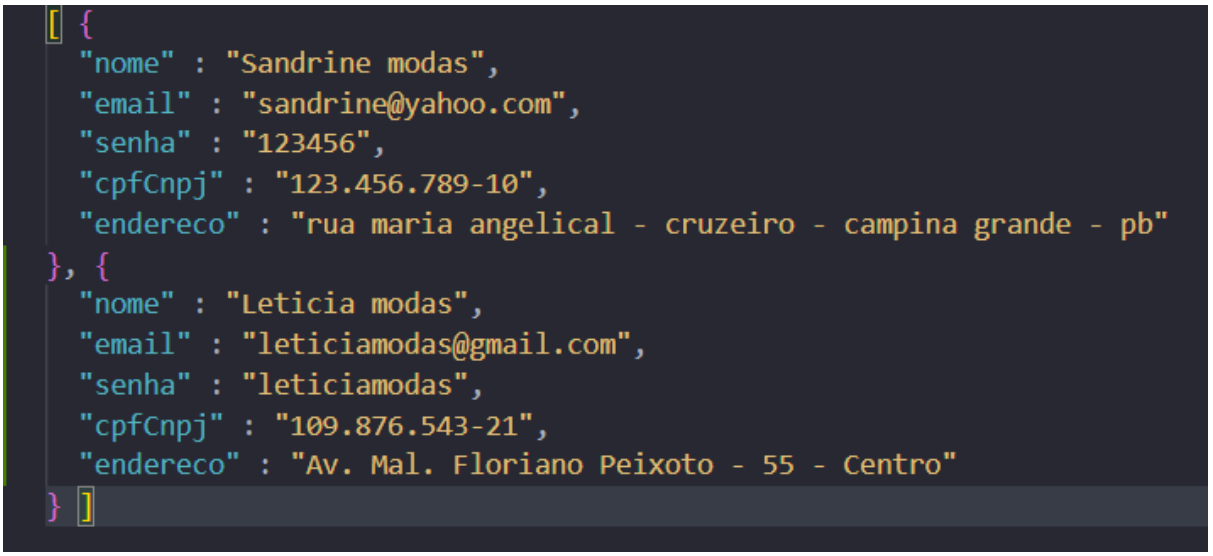
Lojas cadastradas salvas no arquivo lojas.json:

```
[ {
  "nome" : "Sandrine modas",
  "email" : "sandrine@outlook.com",
  "senha" : "123456",
  "cpfCnpj" : "123.456.789-10",
  "endereco" : "rua maria angelical - cruzeiro - campina grande - pb"
} ]
```

Exclusão de lojas:

Foi implementado um sistema para remover lojas do sistema, com validação da existência da loja e atualização do arquivo JSON. O método deleteLojaPorCpfCnpj(String cpfCnpj) da classe LojaController é responsável por excluir uma loja utilizando o CPF/CNPJ. O processo de exclusão envolve: verificar a validade do CPF/CNPJ, remover a loja da lista de lojas carregadas no arquivo e, por fim, atualizar o arquivo JSON para refletir a lista sem a loja excluída. Essa funcionalidade garante que a loja seja removida do sistema de maneira eficiente, mantendo a integridade e persistência dos dados.

Arquivo lojas.json antes da loja ser deletada:



```
[ {
  "nome" : "Sandrine modas",
  "email" : "sandrine@yahoo.com",
  "senha" : "123456",
  "cpfCnpj" : "123.456.789-10",
  "endereco" : "rua maria angelical - cruzeiro - campina grande - pb"
}, {
  "nome" : "Leticia modas",
  "email" : "leticiamodas@gmail.com",
  "senha" : "leticiamodas",
  "cpfCnpj" : "109.876.543-21",
  "endereco" : "Av. Mal. Floriano Peixoto - 55 - Centro"
} ]
```

Arquivo lojas.json após a exclusão da loja cujo CNPJ é 109.876.543-21:

Atualização de Lojas:

Implementação de um método para atualizar os dados de uma loja existente, com validação e persistência das alterações no arquivo JSON.

A classe Loja oferece métodos setters que permitem modificar individualmente os atributos de uma loja. Cada setter realiza a atualização de um atributo específico, com validações aplicadas quando necessário para garantir a integridade dos dados.

A classe LojaController desempenha um papel central na atualização de lojas dentro do sistema. Ela oferece a funcionalidade de editar os dados de uma loja existente através do método atualizarLoja, que segue um fluxo de validação e persistência. As principais funcionalidades envolvidas na atualização de lojas são: validação do CNPJ, que garante que a loja seja identificada corretamente no sistema; busca da loja, que localiza a loja a ser atualizada, com base no CPF/CNPJ fornecido; alteração dos dados, que utiliza os setters da classe Loja para modificar os atributos da loja, validando cada campo; persistência, que salva as alterações no arquivo JSON, garantindo a integridade dos dados; retorno da loja atualizada, que retorna uma confirmação de que os dados da loja foram atualizados.

Essa funcionalidade facilita a atualização dos dados das lojas, assegurando a consistência e a persistência das informações no sistema.

Arquivo lojas.json antes da atualização:

Arquivo lojas.json após a atualização:

2. CRUD de Compradores

Cadastro de compradores

A funcionalidade de cadastro de compradores foi implementada, permitindo inserir dados como nome, e-mail, CPF e endereço, com validação de dados (CPF e e-mail).

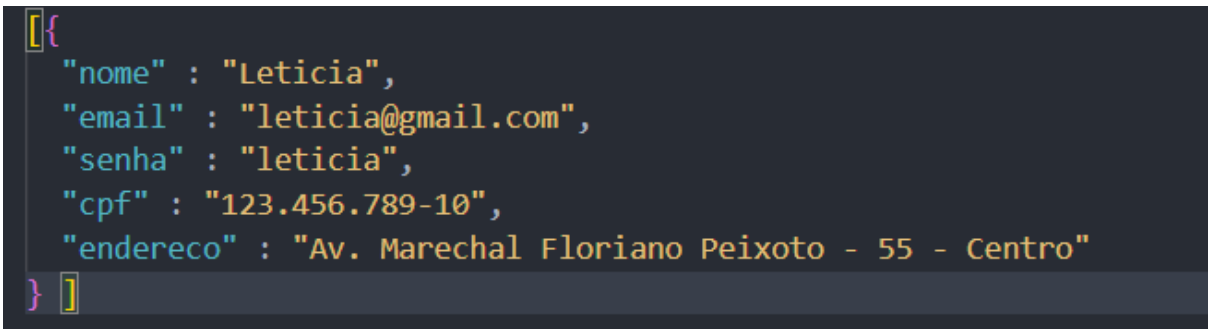
Foi desenvolvido um sistema para cadastrar compradores no marketplace, com foco na validação dos dados e no armazenamento seguro das informações em um arquivo JSON. Para o cadastro, são solicitados os seguintes dados: nome do comprador, que não pode ser vazio; e-mail, que deve conter um formato válido (com "@" e um domínio); senha, com um mínimo de seis caracteres; CPF, que deve incluir os caracteres especiais (ponto e traço); e endereço, que deve ser informado corretamente. A validação desses dados é realizada na classe Comprador por meio de expressões regulares (regex) e verificações específicas, garantindo que apenas informações no formato correto sejam aceitas.

A classe CompradorController gerencia o processo de cadastro e assegura a persistência dos dados no arquivo compradores.json. O método create(Comprador comprador) realiza as seguintes etapas: carrega a lista de compradores já cadastrados, verifica se o CPF já está

registrado no sistema para evitar duplicações, valida os dados fornecidos, adiciona o novo comprador à lista e salva os dados no arquivo JSON. Caso algum dado seja inválido ou o CPF já exista, são lançadas exceções que impedem o cadastro, garantindo a integridade e consistência dos registros. Ao final, o método retorna uma confirmação de sucesso no cadastro ou lança erros específicos se algo estiver errado.

Esse mecanismo assegura a organização e a confiabilidade dos cadastros de compradores, mantendo as informações bem estruturadas e acessíveis no sistema.

Comprador salvo no arquivo compradores.json:



Visualização de compradores

Foi implementado um sistema para listar todos os compradores cadastrados e a possibilidade de visualizar um comprador específico.

A funcionalidade de visualização de compradores foi implementada para permitir a exibição de todos os compradores cadastrados ou a consulta detalhada de um comprador específico, incluindo seu nome, e-mail, CPF e endereço.

A classe Comprador disponibiliza métodos getters, como getNome(), getEmail(), getCpf() e getEndereco(), que permitem acessar individualmente os atributos de um objeto comprador. Além disso, o método toString() foi sobrescrito para fornecer uma representação legível do comprador ao ser exibido, incluindo todos os principais dados de forma estruturada, como nome, e-mail, CPF e endereço.

Já a classe CompradorController é responsável por recuperar e exibir os dados dos compradores registrados no sistema. O método getTodosCompradores() retorna uma lista contendo todos os compradores cadastrados, organizados alfabeticamente pelo nome. Para isso, ele primeiro carrega os dados do arquivo JSON, realiza a ordenação dos compradores e,

por fim, retorna a lista formatada para exibição. Isso garante que os compradores sejam listados de maneira clara e ordenada, facilitando a visualização e consulta dos dados no sistema.

Além disso, também é possível fazer a consulta detalhada de um comprador específico com base no número de exibição, retornando as informações completas de um único comprador quando solicitado. Isso é útil para acessar os dados de um comprador de forma direta e sem a necessidade de exibir toda a lista.

Compradores cadastrados salvos no arquivo compradores.json:


```
[ {
  "nome" : "Leticia Medeiros",
  "email" : "leticia@gmail.com",
  "senha" : "leticia",
  "cpf" : "123.456.789-10",
  "endereco" : "Av. Brasilia - 55 - Catole"
} ]
```

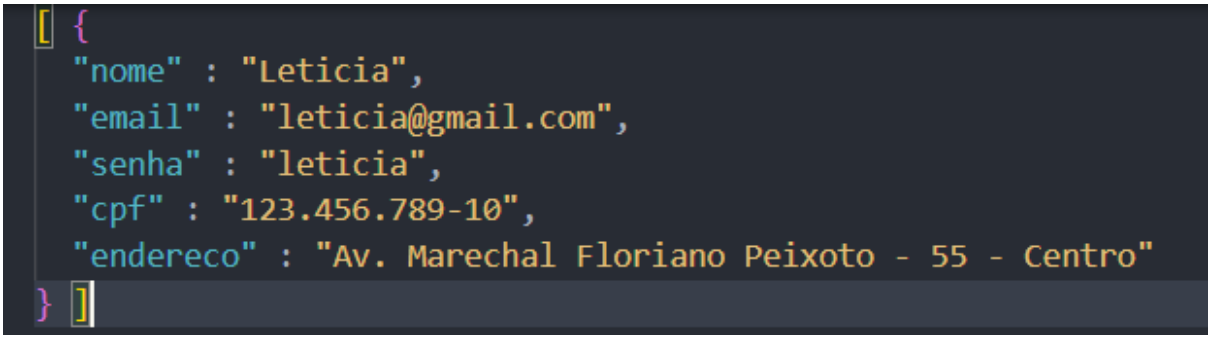

Exclusão de Compradores:

Foi criado um método para remover compradores do sistema, validando a existência do comprador no arquivo JSON antes da exclusão.

Foi implementado um sistema para remover compradores do sistema, com validação da existência do comprador e atualização do arquivo JSON. O método deleteCompradorPorCpf(String cpf) da classe CompradorController é responsável por excluir um comprador utilizando o CPF. O processo de exclusão começa com a verificação da validade do CPF, garantindo que o valor fornecido não seja nulo ou vazio. Em seguida, a classe carrega a lista de todos os compradores por meio do método getTodosCompradores(), que recupera os dados do arquivo JSON. A busca pelo comprador é feita na lista, identificando o comprador correto pelo CPF informado. Caso o CPF seja encontrado, o comprador correspondente é removido da lista utilizando o método removeIf(), o que garante a remoção eficiente do registro. Por fim, a lista atualizada de compradores é salva de volta no arquivo JSON, utilizando o ObjectMapper e um FileWriter. Esse processo de atualização

mantém a integridade e persistência dos dados, assegurando que a exclusão seja realizada de forma segura e eficaz.

Arquivo compradores.json antes do comprador ser deletado:



Arquivo compradores.json após o comprador com CPF 123.456.789-10 ser deletado:

Atualização de Compradores:

A atualização dos dados dos compradores foi implementada, com validação dos novos dados e a persistência das alterações no arquivo JSON.

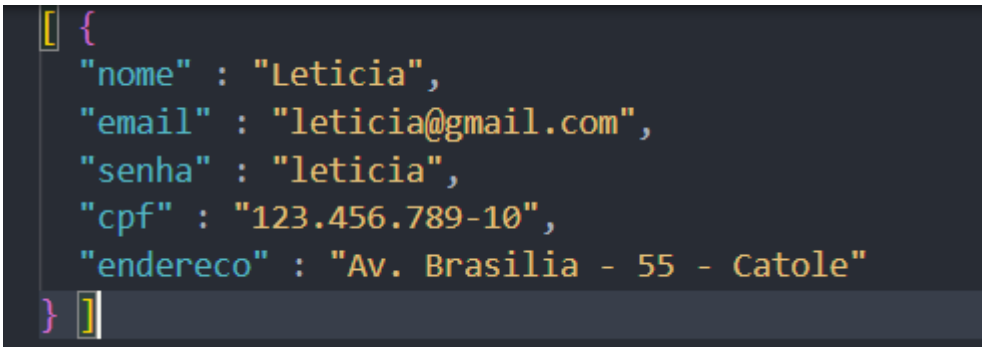
A classe Comprador e a classe CompradorController desempenham papéis fundamentais na atualização dos dados dos compradores no sistema. A classe Comprador oferece métodos setters que permitem modificar individualmente os atributos de um comprador, como nome, e-mail, senha, CPF e endereço. Cada setter é responsável por validar e atualizar um atributo específico, garantindo que os dados do comprador permaneçam consistentes e válidos.

Já a classe CompradorController gerencia o processo de atualização dos dados dos compradores. O método principal responsável por essa funcionalidade é o atualizarComprador. Esse processo começa com a validação dos dados fornecidos, como nome, e-mail, senha, CPF e endereço, para garantir que os dados estejam corretos antes de serem atualizados. Após a validação, a classe CompradorController utiliza o método getTodosCompradores() para carregar a lista de todos os compradores cadastrados e localiza o comprador pelo CPF.

Uma vez encontrado o comprador, a classe CompradorController usa os setters da classe Comprador para modificar os dados desejados, validando cada campo conforme necessário. Depois de alterar os dados, a lista de compradores é salva novamente no arquivo JSON, utilizando o ObjectMapper para garantir que as alterações sejam persistidas no sistema. Por fim, o método atualizarComprador retorna uma confirmação de atualização, confirmando que a operação foi realizada com sucesso.

Esse processo assegura que os dados dos compradores sejam mantidos consistentes e atualizados, garantindo a integridade das informações no sistema.

Arquivo compradores.json antes da atualização:



Arquivo compradores.json após a atualização:

3. CRUD de Produtos

Cadastro de produtos

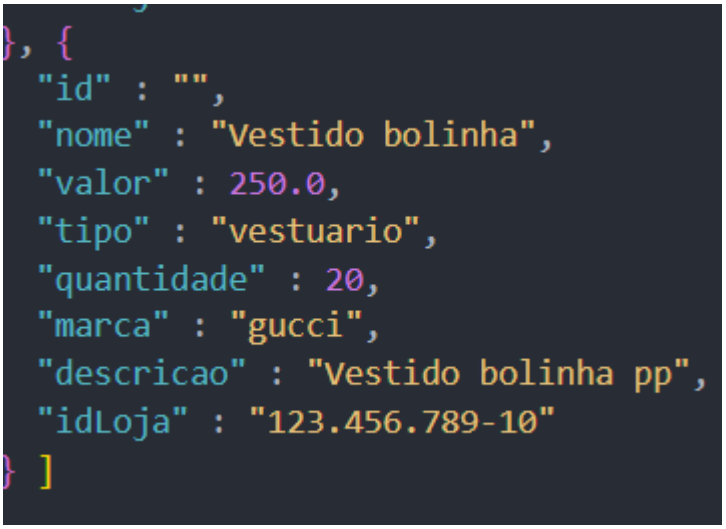
A funcionalidade de cadastro de produtos foi desenvolvida para permitir o cadastro de novos produtos no marketplace, garantindo a validação dos dados e a persistência segura das informações em um arquivo JSON. Para o cadastro, são solicitados os seguintes dados: nome

do produto, descrição, preço, categoria, id do produto, CNPJ da loja a qual esse produto pertence e a marca do produto. A validação desses dados é realizada na classe Produto, utilizando expressões regulares (regex) e verificações lógicas, assegurando que apenas informações no formato correto sejam cadastradas no sistema.

A classe ProdutoController gerencia o processo de cadastro e garante a persistência dos dados no arquivo produtos.json. O método createProduto segue um fluxo estruturado para realizar o cadastro de um novo produto: primeiro, carrega a lista de produtos já cadastrados; em seguida, verifica se o código do produto já existe para evitar duplicações; depois, adiciona o novo produto à lista e salva os dados atualizados no arquivo JSON. Ao final do processo, o sistema retorna uma confirmação de sucesso no cadastro.

Esse mecanismo assegura que os produtos sejam cadastrados corretamente, preservando a integridade das informações no sistema e organizando os dados de maneira eficiente para futuras consultas.

Produto salvo no arquivo produtos.json:



Visualização de Produtos:

Foi criado um método para listar todos os produtos ou visualizar os detalhes de um produto específico, permitindo a exibição das informações.

A funcionalidade de visualização de produtos foi implementada para permitir a exibição de todos os produtos cadastrados no sistema, bem como a consulta detalhada de um produto específico, incluindo atributos como nome, preço e categoria.

A classe Produto disponibiliza métodos *getters*, que permitem acessar individualmente os atributos de um objeto produto. Além disso, o método toString foi sobrescrito para fornecer uma representação legível do produto ao ser exibido.

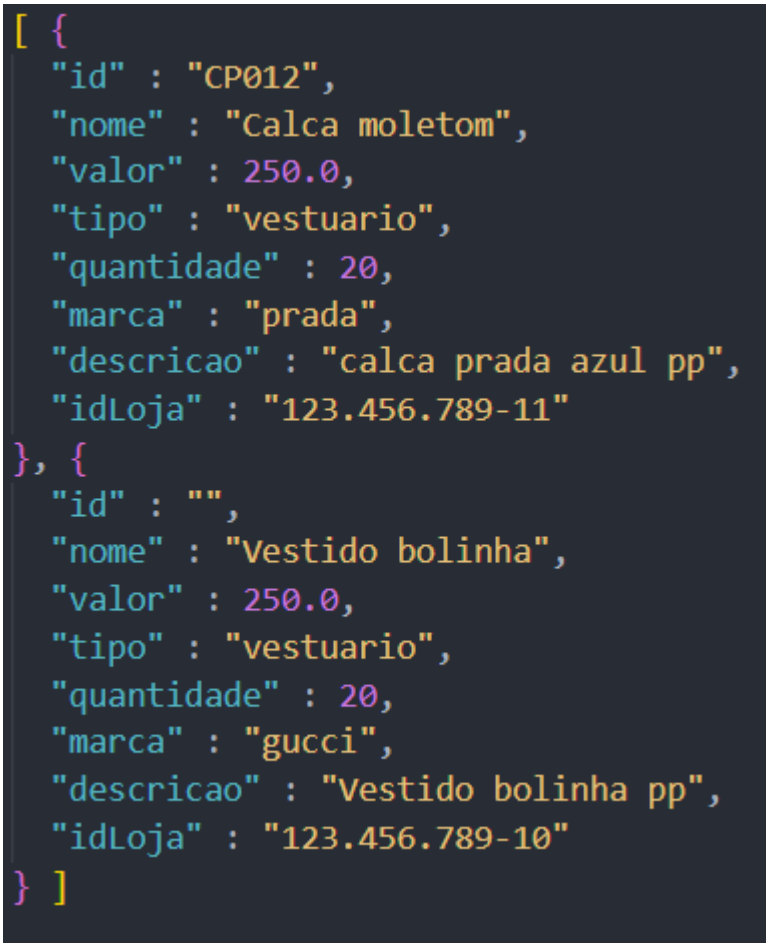
Já a classe ProdutoController é responsável por recuperar e exibir os dados dos produtos registrados no sistema. O método listarProdutos() retorna uma lista contendo todos os produtos cadastrados, permitindo a exibição geral. Para facilitar a organização, o método ordenarProdutosPorNome() ordena os produtos alfabeticamente antes de exibi-los.

A funcionalidade de busca de produtos cadastrados no sistema também foi implementada para permitir a consulta de produtos tanto por ID quanto por loja. A classe Produto oferece métodos getters que possibilitam o acesso aos atributos individuais de um produto, como o ID e a loja associada. Além disso, o método toString foi sobrescrito para fornecer uma representação legível do produto ao ser exibido.

Já a classe ProdutoController é responsável por recuperar e exibir os produtos registrados no sistema. O método buscarProdutoPorId(int id) permite localizar um produto específico com base no seu ID. Ele percorre a lista de produtos carregados do sistema e, ao encontrar o produto com o ID correspondente, retorna esse produto. Já o método buscarProdutosPorLoja(String lojaId) possibilita a busca de todos os produtos associados a uma loja específica. Ele filtra os produtos pela loja a que pertencem e retorna a lista de produtos dessa loja.

Para realizar essas buscas, o ProdutoController carrega os dados dos produtos a partir de uma fonte persistente, como um arquivo JSON. A classe realiza a filtragem dos produtos de forma eficiente, garantindo que os resultados sejam retornados com precisão. Essas funcionalidades permitem a visualização detalhada dos produtos, seja por ID ou pela loja, oferecendo uma experiência de consulta rápida e precisa no sistema.

Produtos cadastrados no sistema salvos no arquivo produtos.json:



Exclusão de Produtos

A exclusão de produtos no sistema é realizada por meio da classe ProdutoController, que oferece um método específico para esse processo. O método removerProduto(int id) é responsável por excluir um produto, utilizando seu identificador único (ID). O processo começa com a verificação da existência do produto na lista cadastrada, onde o ID fornecido é utilizado para localizar o item. Caso o produto seja encontrado, ele é removido da lista. Em seguida, o arquivo JSON que armazena os dados dos produtos é atualizado, refletindo a remoção do item. Essa abordagem garante que a exclusão do produto ocorra de maneira eficiente e que os dados no sistema permaneçam consistentes e atualizados.

Arquivo produtos.json antes do produto ser deletado:

```
[ {
  "id" : "CP01",
  "nome" : "Calca jeans",
  "valor" : 250.0,
  "tipo" : "vestuario",
  "quantidade" : 20,
  "marca" : "prada",
  "descricao" : "calca prada azul pp",
  "idLoja" : "123.456.789-10"
}, {
  "id" : "CP012",
  "nome" : "Calca jeans",
  "valor" : 250.0,
  "tipo" : "vestuario",
  "quantidade" : 20,
  "marca" : "prada",
  "descricao" : "calca prada azul pp",
  "idLoja" : "123.456.789-11"
}, {
  "id" : "",
  "nome" : "Vestido bolinha",
  "valor" : 250.0,
  "tipo" : "vestuario",
  "quantidade" : 20,
  "marca" : "gucci",
  "descricao" : "Vestido bolinha pp",
  "idLoja" : "123.456.789-10"
} ]
```


Arquivo produtos.json após a exclusão do produto cujo ID é CP01:


```
[ {
  "id" : "CP012",
  "nome" : "Calca jeans",
  "valor" : 250.0,
  "tipo" : "vestuario",
  "quantidade" : 20,
  "marca" : "prada",
  "descricao" : "calca prada azul pp",
  "idLoja" : "123.456.789-11"
}, {
  "id" : "",
  "nome" : "Vestido bolinha",
  "valor" : 250.0,
  "tipo" : "vestuario",
  "quantidade" : 20,
  "marca" : "gucci",
  "descricao" : "Vestido bolinha pp",
  "idLoja" : "123.456.789-10"
} ]
```

Atualização de Produtos

A implementação de um método para atualizar os dados de um produto existente, com validação e persistência das alterações no arquivo JSON, é realizada através da interação das classes Produto e ProdutoController. A classe Produto oferece métodos setters que permitem modificar individualmente os atributos de um produto, como nome, valor, quantidade, marca, tipo, descrição e ID. Esses setters aplicam validações para garantir que os dados inseridos sejam consistentes e válidos, como a verificação do valor e da quantidade de estoque.

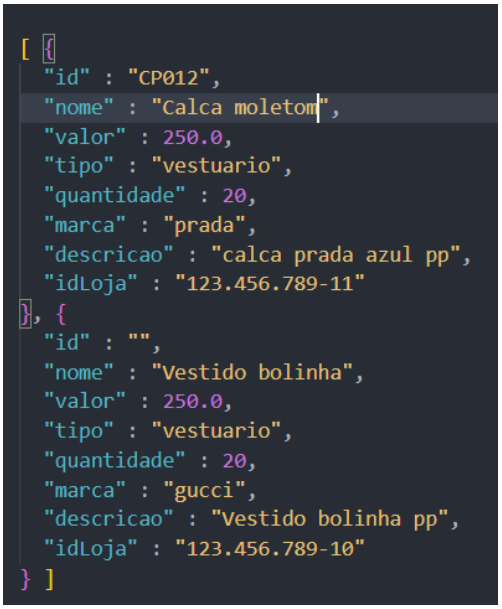
Por sua vez, a classe ProdutoController desempenha um papel central na atualização de produtos dentro do sistema. A principal funcionalidade que ela oferece é a capacidade de editar os dados de um produto existente, por meio do método atualizarProduto. Esse processo envolve um fluxo de validação e persistência: primeiramente, é feita a validação dos dados fornecidos, como o valor e a quantidade, para garantir que estão dentro dos parâmetros esperados. Em seguida, o produto é localizado, geralmente utilizando o ID, e os dados são alterados por meio dos setters da classe Produto, com a devida validação de cada campo. Após as modificações, o produto atualizado é salvo novamente no arquivo JSON, garantindo que as mudanças sejam refletidas de forma permanente no sistema. Por fim, uma confirmação é retornada, assegurando que os dados do produto foram atualizados com sucesso.

Essa funcionalidade assegura a atualização eficiente dos dados dos produtos, mantendo a consistência e integridade das informações no sistema.

Arquivo produtos.json antes da atualização:


```
[ {
  "id" : "CP012",
  "nome" : "Calca jeans",
  "valor" : 250.0,
  "tipo" : "vestuario",
  "quantidade" : 20,
  "marca" : "prada",
  "descricao" : "calca prada azul pp",
  "idLoja" : "123.456.789-11"
}, {
  "id" : "",
  "nome" : "Vestido bolinha",
  "valor" : 250.0,
  "tipo" : "vestuario",
  "quantidade" : 20,
  "marca" : "gucci",
  "descricao" : "Vestido bolinha pp",
  "idLoja" : "123.456.789-10"
} ]
```


Arquivo produtos.json após a atualização do nome do produto cujo ID é CP012:



4. Testes

Testes de Métodos das Classes Foram criados testes unitários para os métodos das classes Loja, Comprador e Produto, incluindo testes para os métodos getters e setters, construtores, validação de dados (CPF, e-mail, etc.) e o método de representação como string. Os testes garantiram que os atributos das classes fossem inicializados corretamente e que as validações, como a verificação de CPF, CNPJ, e-mail, e dados numéricos (valor e quantidade), estivessem funcionando corretamente.

Além disso, foram implementados testes para verificar o comportamento dos métodos de controle nas classes LojaController, CompradorController e ProdutoController, garantindo que operações como criação, atualização, exclusão e consulta de registros fossem realizadas de maneira eficaz. Os testes verificaram se as exceções adequadas eram lançadas em caso de dados inválidos ou tentativas de operações ilegais, como a criação de registros com valores negativos ou a tentativa de excluir registros inexistentes.

A criação, por exemplo, foi testada para garantir que as lojas, compradores e produtos fossem registrados corretamente no sistema, com todas as informações sendo validadas antes da persistência dos dados. A atualização de registros foi validada para garantir que as alterações feitas fossem refletidas corretamente, enquanto a exclusão foi testada para garantir que registros errados ou desnecessários fossem removidos de forma eficaz.

Além disso, os testes garantiram que as consultas retornassem os resultados esperados, com os dados sendo recuperados corretamente de acordo com os identificadores fornecidos, e que a ausência de registros gerasse respostas apropriadas, como retornos nulos ou mensagens de erro claras. Esses testes ajudam a validar a integridade e a consistência dos dados no sistema, além de garantir que as operações de manipulação de informações sejam feitas com precisão e segurança.

Com esses testes, foi possível garantir que a aplicação esteja robusta, eficiente e livre de falhas, proporcionando uma experiência mais confiável e segura para os usuários.

Sprint 2 - Registro de funcionalidades de desenvolvimento da View e do controle de acesso

1. CRUD de Lojas

Cadastro de lojas

A classe LojaView contribui para a funcionalidade de cadastro de lojas principalmente por meio do método cadastrarLoja(), que interage com o usuário através do console. Nesse método, o usuário é solicitado a fornecer as informações necessárias para cadastrar uma nova loja. A classe realiza algumas validações simples nos campos, como verificar se o nome não está vazio, se o e-mail contém um "@" e se a senha tem pelo menos 6 caracteres. O CNPJ também é validado com uma expressão regular para garantir que esteja no formato correto.

Após validar os dados, o método chama o LojaController.createLoja(), passando as informações coletadas do usuário. Caso ocorra algum erro, como dados inválidos ou CNPJ já cadastrado, uma exceção do tipo IllegalArgumentException é lançada e a mensagem de erro é exibida ao usuário. Se o cadastro for bem-sucedido, uma mensagem de sucesso é mostrada, indicando que a loja foi cadastrada com sucesso.

Cadastrando uma loja:

```
=== Cadastro de Loja ===
Nome: Leticia modas
E-mail: leticiamodas@gmail.com
Senha (mínimo 6 caracteres): leticiamodas
CPF/CNPJ: 109.876.543-21
Endereço: Av. Mal. Floriano Peixoto - 55 - Centro
Cadastro realizado com sucesso!
```

Visualização de lojas

A classe LojaView contribui para a funcionalidade de visualização e exibição de lojas cadastradas no sistema por meio do método listarLojas(). Esse método recupera a lista de todas as lojas cadastradas, chamando o método LojaController.getTodasLojas(), que retorna as lojas armazenadas.

Após obter as lojas, o método exibe uma mensagem no console com o título "=== Lista de Lojas Cadastradas ===". Em seguida, ele verifica se a lista de lojas está vazia. Se estiver, uma mensagem indicando que não há lojas cadastradas é exibida. Caso contrário, o método percorre a lista de lojas, exibindo o nome e o CPF/CNPJ de cada loja de forma ordenada. A numeração de cada loja na listagem é gerada com base na posição da loja na lista, começando de 1.

Exibição das lojas cadastradas no sistema:

```
=== Lista de Lojas Cadastradas ===
1. Leticia modas - 109.876.543-21
2. Sandrine modas - 123.456.789-10
```

Exclusão de lojas

A classe LojaView contribui para a funcionalidade de exclusão de lojas cadastradas no sistema por meio do método deletarLoja(). Esse método permite que o usuário remova uma loja do sistema, interagindo diretamente com a interface de linha de comando.

Primeiramente, o método solicita ao usuário que insira o CPF/CNPJ da loja a ser deletada. Ele valida a entrada do CPF/CNPJ, garantindo que o formato seja válido, com o uso de uma

expressão regular. Caso o CPF/CNPJ fornecido seja inválido, o sistema pede ao usuário para inserir um valor válido.

Após a validação do CPF/CNPJ, o método tenta excluir a loja correspondente chamando o método LojaController.deleteLojaPorCpfCnpj(cpfCnpj). Caso o CPF/CNPJ fornecido não corresponda a uma loja existente, uma exceção será gerada e tratada. Se a loja for removida com sucesso, uma mensagem de confirmação será exibida, informando que a loja foi deletada com sucesso.

Excluindo uma loja do sistema:

```
=== Deletar Loja ===
Digite o CPF/CNPJ da loja a ser deletada: 109.876.543-21
Loja deletada com sucesso!
```

Atualização de Lojas

A classe LojaView contribui para a funcionalidade de atualização dos dados de lojas cadastradas no sistema por meio do método atualizarLoja(). Esse método permite ao usuário atualizar as informações de uma loja já cadastrada, oferecendo uma interface interativa para a edição dos dados.

O processo começa com a solicitação ao usuário para fornecer o CPF/CNPJ da loja a ser atualizada. O CPF/CNPJ é validado com uma expressão regular para garantir que esteja no formato correto. Caso o CPF/CNPJ seja inválido, o usuário é solicitado a inserir um valor válido.

Em seguida, o método tenta localizar a loja correspondente ao CPF/CNPJ informado, utilizando o método LojaController.getLojaPorCpfCnpj(cpfCnpj). Se a loja não for encontrada, uma mensagem de erro é exibida, e o processo de atualização é interrompido. Caso a loja seja localizada, o sistema exibe os dados atuais da loja e permite ao usuário editar os campos desejados, como nome, e-mail, senha e endereço. O usuário pode deixar os campos em branco se não desejar alterar determinadas informações.

Após a entrada dos dados de atualização, o método chama o LojaController.atualizarLoja() para aplicar as alterações no sistema. A loja é então atualizada com as novas informações, e uma mensagem de sucesso é exibida.

Atualizando o e-mail de uma loja cadastrada no sistema:

```
=== Atualizar Loja ===
Digite o CPF/CNPJ da loja a ser atualizada: 123.456.789-10
Deixe em branco os campos que não deseja alterar.
Novo Nome [Sandrine modas]:
Novo E-mail [sandrine@gmail.com]: sandrine@outlook.com
Nova Senha (mínimo 6 caracteres) [*****]:
Novo Endereço [rua maria angelical - cruzeiro - campina grande - pb]:
Loja atualizada com sucesso!
```

2. CRUD de Compradores

Cadastro de compradores

A classe CompradorView desempenha um papel essencial na interação com o usuário para o cadastro de compradores no sistema. Ela coleta as informações fornecidas pelo usuário, como nome, e-mail, senha, CPF e endereço, e valida esses dados antes de prosseguir. Caso algum dado esteja incorreto ou inválido, como um CPF com formato errado ou uma senha com menos de 6 caracteres, a classe solicita que o usuário insira as informações novamente, garantindo que todos os campos atendam aos critérios estabelecidos.

Após a validação dos dados, a CompradorView cria um objeto Comprador com as informações fornecidas pelo usuário e chama o método create do CompradorController. Esse método é responsável por adicionar o comprador ao sistema, garantindo que os dados sejam corretamente persistidos. O CompradorController cuida da lógica de verificação, como garantir que o CPF não esteja duplicado, e realiza a escrita do novo comprador no arquivo JSON. Além disso, a classe fornece feedback ao usuário, informando se o cadastro foi realizado com sucesso ou se ocorreu algum erro.

Cadastrando um comprador no sistema:

```
=== Cadastro de Comprador ===
Nome: Leticia
E-mail: leticia@gmail.com
Senha (mínimo 6 caracteres): leticia
CPF (XXX.XXX.XXX-XX): 123.456.789-10
Endereço: Av. Marechal Floriano Peixoto - 55 - Centro
Comprador adicionado com sucesso.
Cadastro realizado com sucesso!
```

Visualização de compradores

A classe CompradorView contribui significativamente para a funcionalidade de visualização e exibição de compradores ao fornecer uma interface de usuário que permite listar todos os compradores cadastrados e exibir detalhes sobre cada um deles. O processo começa quando o método listarCompradores é chamado. Ele solicita ao CompradorController a lista de todos os compradores registrados no sistema, utilizando o método getTodosCompradores.

Após obter essa lista, a classe verifica se há compradores cadastrados. Se a lista estiver vazia, exibe uma mensagem informando que não há compradores registrados. Caso contrário, ela imprime o nome e o CPF de cada comprador de forma ordenada, facilitando a visualização para o usuário.

Além disso, a classe permite que o usuário selecione um comprador específico para visualizar seus detalhes completos. Através de um número associado a cada comprador na lista, o usuário pode escolher qual comprador deseja ver em detalhes. Ao fazer essa escolha, a CompradorView exibe informações como nome, e-mail, CPF e endereço do comprador selecionado.

Esse processo garante que o usuário possa não só visualizar todos os compradores cadastrados, mas também acessar informações detalhadas de cada um.

Exibindo compradores cadastrados no sistema:

```
=== Lista de Compradores Cadastrados ===
1. Leticia - 123.456.789-10
Digite o número do comprador para ver detalhes ou 0 para sair: 1
Detalhes do Comprador:
Nome: Leticia
E-mail: leticia@gmail.com
CPF: 123.456.789-10
Endereço: Av. Marechal Floriano Peixoto - 55 - Centro
```

Exclusão de compradores

A classe CompradorView contribui para a funcionalidade de exclusão de compradores ao fornecer uma interface interativa que permite ao usuário selecionar um comprador a ser removido do sistema. O processo começa com o método deletarComprador, que exibe uma lista de todos os compradores cadastrados, utilizando o método getTodosCompradores do CompradorController.

Após listar todos os compradores, o método solicita ao usuário que informe o CPF do comprador que deseja excluir. Para garantir que o CPF inserido seja válido, a classe realiza uma validação do formato do CPF, utilizando uma expressão regular para verificar se ele segue o padrão XXX.XXX.XXX-XX. Se o CPF não for válido, o usuário é solicitado a inserir um CPF correto.

Uma vez que o CPF é validado, o método chama o método deleteCompradorPorCpf do CompradorController para realizar a exclusão. Esse método verifica se o comprador com o CPF informado existe e, caso contrário, retorna uma mensagem informando que o comprador não foi encontrado. Se a exclusão for bem-sucedida, o método retorna uma confirmação de que o comprador foi removido com sucesso.

Excluindo um comprador do sistema:

```
=== Deletar Comprador ===
Lista de Compradores:
Nome: Leticia | CPF: 123.456.789-10
Digite o CPF do comprador a ser deletado (XXX.XXX.XXX-XX): 123.456.789-10
Comprador removido com sucesso.
```

Atualização de compradores

A classe CompradorView contribui para a funcionalidade de alteração ou atualização dos dados dos compradores ao fornecer uma interface interativa que permite ao usuário modificar as informações de um comprador já cadastrado. O processo de atualização de dados é gerido pelo método atualizarComprador.

O método começa solicitando ao usuário que informe o CPF do comprador cujo cadastro deseja ser alterado. Para garantir que o CPF seja válido, a classe realiza uma validação do formato utilizando uma expressão regular, exigindo que o CPF siga o padrão XXX.XXX.XXX-XX. Caso o CPF seja inválido, o usuário é solicitado a inseri-lo novamente.

Após validar o CPF, a classe utiliza o método getCompradorPorCpf do CompradorController para recuperar os dados do comprador existente. Se o comprador com o CPF fornecido não for encontrado, o sistema informa o usuário e interrompe o processo de atualização.

Se o comprador for encontrado, a classe exibe os dados atuais e permite ao usuário atualizar as informações. O método solicita novos valores para os campos de nome, e-mail, senha e endereço, informando que, caso o usuário queira manter um dado atual, basta deixar o campo vazio. Para garantir que a senha tenha pelo menos 6 caracteres, uma validação adicional é realizada.

Depois que o usuário fornece os novos dados ou decide manter os antigos, um novo objeto Comprador é criado com as informações atualizadas. Em seguida, o método chama o método atualizarComprador do CompradorController, que realiza a atualização no arquivo JSON onde os dados dos compradores estão armazenados. Caso a atualização seja bem-sucedida, o sistema informa ao usuário que a alteração foi realizada com sucesso.

Atualizando o nome de um comprador cadastrado no sistema:

```
=== Atualizar Comprador ===
Digite o CPF do comprador a ser atualizado (XXX.XXX.XXX-XX): 123.456.789-10
OBS: Deixe os campos vazios para manter os dados atuais.
Novo Nome (atual: Leticia): Leticia Medeiros
Novo E-mail (atual: leticia@gmail.com):
Nova Senha (mínimo 6 caracteres, deixar vazio para manter):
Novo Endereço (atual: Av. Brasilia - 55 - Catol?):
Comprador atualizado com sucesso!
```

3. CRUD de Produtos

Cadastro de produtos

A classe ProdutoView contribui para a funcionalidade de cadastro de produtos ao fornecer uma interface que permite ao usuário inserir os dados necessários para registrar um novo produto no sistema. O método cadastrarProduto(String idLoja) é o responsável por isso. Ele começa solicitando ao usuário que forneça informações como nome, valor, tipo, quantidade, marca, descrição, ID do produto e o ID da loja, utilizando o Scanner para capturar essas entradas. Durante esse processo, a classe realiza validações para garantir que os dados sejam válidos, como verificar se o valor do produto é maior que zero, se a quantidade não é negativa e se campos como nome, tipo e descrição não estão vazios.

Após a validação, a classe chama o método ProdutoController.create(), passando as informações coletadas para o controlador, que é encarregado de persistir os dados no sistema, garantindo que o produto seja registrado corretamente no arquivo JSON. Caso ocorra algum erro, como um valor inválido ou uma exceção inesperada, o método captura esses erros e exibe mensagens apropriadas para informar o usuário sobre o problema.

Por fim, caso o cadastro seja bem-sucedido, o sistema fornece um feedback positivo ao usuário, informando que o produto foi cadastrado com sucesso. Se algum erro ocorrer durante o processo, o usuário será notificado para corrigir os dados fornecidos.

Cadastrando um produto no sistema:

```
=== Cadastro de Produto ===
Nome: Vestido bolinha
Valor: 250
Tipo: vestuario
Quantidade: 20
Marca: gucci
Descrição: Vestido bolinha pp
ID do Produto:
ID da Loja (CNPJ/CPF): 123.456.789-10
-----------------------------------
Produto adicionado.
Produto cadastrado com sucesso!
```

Visualização de produtos

A classe ProdutoView contribui para a funcionalidade de visualização e exibição de dados dos produtos ao fornecer métodos que permitem ao usuário visualizar as informações dos produtos cadastrados de maneira organizada e clara.

O método listarProdutosPorLoja(String cnpjCpfLoja) é responsável por listar todos os produtos pertencentes a uma loja específica. Ele solicita o ID da loja (CNPJ ou CPF), caso o usuário seja um administrador, e então utiliza o controlador ProdutoController para recuperar a lista de produtos associados à loja. Caso não haja produtos para a loja fornecida, uma mensagem informando a ausência de produtos é exibida. Se houver produtos, a classe formata e exibe os detalhes de cada um, utilizando o método ProdutoController.formatarProduto(), que organiza as informações de forma legível.

Já o método listarTodosProdutos() permite ao usuário visualizar todos os produtos cadastrados no sistema. Ele consulta o controlador para obter a lista completa de produtos e, caso existam, exibe os detalhes de cada um. Assim como no método anterior, cada produto é exibido de forma formatada, tornando a visualização dos dados mais amigável e compreensível. Se não houver produtos registrados, o sistema informa o usuário sobre a ausência de produtos cadastrados.

Além disso, o método buscarProdutoPorID(String id) possibilita a busca de um produto específico por seu ID. O usuário é solicitado a inserir o ID do produto que deseja buscar. Se o usuário não for administrador, o sistema limita a busca aos produtos da loja associada ao ID informado. Se o produto for encontrado, seus detalhes são exibidos. Caso contrário, o sistema informa que o produto não foi encontrado.

Exibindo os produtos cadastrados no sistema:

```
=== Listar Todos os Produtos ===
ID: CP01
Nome: Calca jeans
Tipo: vestuario
Marca: prada
Descrição: calca prada azul pp
Valor: 250.0
Quantidade: 20
ID Loja: 123.456.789-10
------------------------------------
------------------------------------
ID: CP012
Nome: Calca jeans
Tipo: vestuario
Marca: prada
Descrição: calca prada azul pp
Valor: 250.0
Quantidade: 20
ID Loja: 123.456.789-11
------------------------------------
```

Buscando produto por ID:

```
=== Buscar Produto por ID ===
Digite o Id do produto a ser buscado: CP012
ID: CP012
Nome: Calca moletom
Tipo: vestuario
Marca: prada
Descrição: calca prada azul pp
Valor: 250.0
Quantidade: 20
ID Loja: 123.456.789-11
-----------------------------------
```

Buscando produtos associados a uma loja específica, cujo ID é “123.456.789-11”:

```
=== Produtos da Loja ===
123.456.789-11
ID: CP012
Nome: Calca moletom
Tipo: vestuario
Marca: prada
Descrição: calca prada azul pp
Valor: 250.0
Quantidade: 20
ID Loja: 123.456.789-11
-----------------------------------
```

Exclusão de produtos

A classe ProdutoView contribui para a funcionalidade de exclusão dos produtos por meio do método deletarProduto(String idLoja), que permite ao usuário remover um produto do sistema.

Esse método começa solicitando o ID do produto a ser excluído. Após o usuário fornecer o ID, a classe verifica se o produto existe chamando o método ProdutoController.getProdutoPorID(id). Se o produto não for encontrado, uma mensagem é exibida informando que o produto não existe.

Em seguida, a classe verifica se o usuário tem permissão para excluir o produto. Para isso, se o usuário não for um administrador, o sistema compara o ID da loja do produto com o ID da loja fornecido como parâmetro (idLoja). Caso o produto pertença a uma loja diferente daquela associada ao usuário, a exclusão não é permitida, e uma mensagem informando que o produto não pode ser deletado é exibida.

Se o produto for encontrado e o usuário tiver permissão para excluí-lo, a classe chama o método ProdutoController.deleteProdutoPorID(id), que é responsável por excluir o produto do sistema. Caso a exclusão seja bem-sucedida, uma mensagem de confirmação é exibida, informando que o produto foi removido com sucesso.

Excluindo um produto do sistema:

```
=== Deletar Produto ===
ID do Produto: CP01
-----------------------------------
Produto removido com sucesso.
Produto removido com sucesso!
```

Atualização de produtos

A classe ProdutoView contribui para a funcionalidade de atualização dos dados do produto por meio do método atualizarProduto(String idLoja), que permite ao usuário modificar as informações de um produto já cadastrado no sistema.

O processo começa com a solicitação do ID do produto que o usuário deseja atualizar. Após o fornecimento do ID, a classe verifica se o produto existe, chamando o método ProdutoController.getProdutoPorID(id). Se o produto não for encontrado, uma mensagem é exibida informando que o produto não existe e o processo é encerrado.

Em seguida, a classe verifica se o usuário tem permissão para atualizar o produto. Caso o usuário não seja um administrador, a classe compara o ID da loja associado ao produto com o ID da loja fornecido como parâmetro (idLoja). Se os IDs não coincidirem, a atualização é negada, e uma mensagem é exibida, informando que o produto não pode ser atualizado porque pertence a outra loja.

Se o produto existir e o usuário tiver permissão para editá-lo, a classe exibe os dados atuais do produto para o usuário e permite que ele modifique as informações de forma opcional. Para cada campo (como nome, valor, tipo, quantidade, marca e descrição), o usuário tem a opção de deixar o campo em branco para manter o valor atual ou fornecer um novo valor. A classe valida as entradas fornecidas pelo usuário, garantindo que o valor do produto seja maior que zero e que a quantidade não seja negativa.

Após a coleta das novas informações, a classe chama o método ProdutoController.atualizarProduto() para atualizar os dados do produto no sistema. Se a atualização for bem-sucedida, uma mensagem de sucesso é exibida, confirmando que o produto foi atualizado com sucesso.

Atualizando o nome de um produto cadastrado no sistema:

```
=== Atualizar Produto ===
ID do Produto: CP012
Produto Atual:
ID: CP012
Nome: Calca jeans
Tipo: vestuario
Marca: prada
Descrição: calca prada azul pp
Valor: 250.0
Quantidade: 20
ID Loja: 123.456.789-11
-----------------------------------
Deixe em branco para manter o mesmo valor.
Novo Nome (atual: Calca jeans): Calca moleton
Novo Valor (atual: 250.0):
Novo Tipo (atual: vestuario):
Nova Quantidade (atual: 20):
Nova Marca (atual: prada):
Nova Descrição (atual: calca prada azul pp):
```

4. Testes

Os testes de integração implementados nas classes LojaViewTest, CompradorViewTest e ProdutoViewTest foram projetados para validar o correto funcionamento das interações entre as diferentes camadas do sistema, garantindo que a comunicação entre a interface do usuário e o back-end ocorra de maneira eficiente e sem falhas.

Esses testes verificam se as operações de cadastro, listagem, busca, atualização e remoção de lojas, compradores e produtos são executadas corretamente, assegurando que as informações sejam registradas, recuperadas e manipuladas de forma adequada. Além disso, os testes garantem que os dados inseridos pelos usuários sejam validados corretamente, impedindo o armazenamento de informações inconsistentes ou inválidas, como CPFs, CNPJs e e-mails

mal formatados, além de valores numéricos negativos para atributos como preço e quantidade de produtos.

Os testes também validam a persistência de dados, conferindo se as operações realizadas na interface refletem corretamente nos arquivos de armazenamento, garantindo que as informações sejam salvas e recuperadas conforme esperado. Da mesma forma, verificam se as mensagens de erro adequadas são exibidas sempre que um usuário tenta realizar operações inválidas, como buscar registros inexistentes ou excluir um item que já foi removido.

Com esses testes de integração, é possível garantir a consistência e confiabilidade da aplicação, assegurando que todas as funcionalidades oferecidas ao usuário operem sem erros e que os dados sejam manipulados de forma segura e precisa.

5. Controle de acesso

Fachada para Login

A classe FachadaLogin tem como objetivo centralizar e gerenciar a autenticação de usuários no sistema, permitindo que diferentes tipos de usuários (Admin, Loja e Comprador) façam login de forma adequada. A classe exibe um menu para o usuário escolher o tipo de conta que deseja acessar (Comprador, Loja ou Admin) e, em seguida, solicita o identificador e a senha. Após a coleta das credenciais, o sistema valida essas informações com base no tipo de usuário selecionado.

O método principal da classe, realizarLogin, entra em um loop onde o usuário escolhe seu tipo de conta. Caso o tipo de usuário seja válido (1 para Comprador, 2 para Loja ou 3 para Admin), o sistema verifica as credenciais inseridas. Para o login de Admin, a autenticação é feita comparando o identificador com o valor "admin" e a senha com "123", valores predefinidos na implementação. Para a Loja e o Comprador, o sistema valida as credenciais através de uma consulta ao banco de dados ou sistema de armazenamento, utilizando o BCrypt para verificar se a senha fornecida corresponde ao hash armazenado.

Se o login for bem-sucedido, a classe chama o método MenuSelecionador da classe FachadaMenus, que apresenta o menu apropriado ao tipo de usuário. Caso as credenciais estejam incorretas, o sistema exibe uma mensagem de erro e o loop reinicia, permitindo que o usuário tente novamente.

Fazendo login como loja:

```
=== Bem-vindo ===
1. Login
2. Registrar
0. Sair
Escolha uma opção: 1

=== Escolha o Tipo de Usuário ===
1. Comprador
2. Loja
3. Admin
0. Sair
Escolha uma opção: 2
Digite seu identificador: 123.456.789-10
Digite sua senha: leticia
Login como Loja realizado com sucesso!
```

Validador de login

A classe ValidadorLogin é responsável por validar as credenciais de login dos usuários no sistema, garantindo que apenas usuários autenticados possam acessar as funcionalidades específicas de acordo com seu tipo. Ela contém três métodos principais, um para cada tipo de usuário no sistema: Loja, Comprador e Administrador.

O método loginLoja(String id, String senha) valida as credenciais de uma loja, verificando se o CPF ou CNPJ fornecido corresponde a uma loja existente e se a senha informada bate com o hash armazenado no banco de dados. O método loginComprador(String id, String senha) realiza uma validação similar para o comprador, conferindo se o CPF fornecido é válido e se a senha coincide com o hash armazenado. Já o método loginADM(String id, String senha) valida as credenciais do administrador, onde o ID é comparado com o valor "admin" e a senha é verificada com uma senha fixa ("123").

Se a validação for bem-sucedida, os métodos retornam true, permitindo que o usuário seja autenticado e tenha acesso às funcionalidades correspondentes. Caso contrário, retornam false, indicando que as credenciais fornecidas estão incorretas. Essa classe desempenha um

papel crucial na segurança do sistema, garantindo que apenas usuários com credenciais válidas possam acessar suas áreas específicas e interagir com as funcionalidades do sistema.

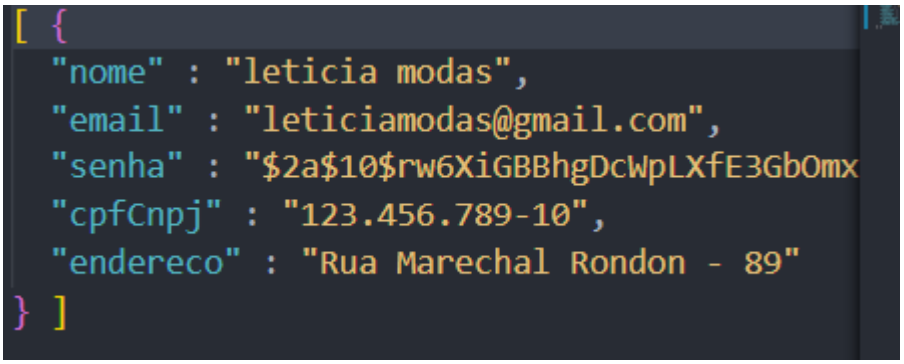
Exemplo de login com as credenciais inválidas:

```
=== Escolha o Tipo de Usuário ===
1. Comprador
2. Loja
3. Admin
0. Sair
Escolha uma opção: 1
Digite seu identificador: 123.456.789-10
Digite sua senha: leticia
Credenciais de Comprador inválidas!
```

Criptografia de dados

A classe SecurityHash tem como objetivo fornecer funcionalidades de segurança para o gerenciamento de senhas no sistema, utilizando o algoritmo BCrypt para garantir o armazenamento seguro das senhas dos usuários. Ela possui dois métodos principais: o primeiro, hashPassword(String password), é responsável por gerar um hash da senha fornecida, utilizando um salt aleatório, e armazena o hash no lugar da senha em texto claro, protegendo os dados sensíveis. O segundo método, checkPassword(String password, String hashed), serve para verificar se a senha fornecida pelo usuário corresponde ao hash armazenado. Ele faz essa comparação utilizando o algoritmo BCrypt, garantindo que a senha digitada durante o login seja validada de forma segura. Dessa forma, a classe SecurityHash assegura que as senhas dos usuários sejam protegidas contra acessos indevidos, armazenando apenas hashes irreversíveis das senhas e evitando o risco de vazamento de informações sensíveis.

Exemplo de uma loja cadastrada no sistema com a senha criptografada:



Menu de registro

A classe MenuRegistro tem como principal função exibir o menu de registro de novos usuários (Comprador ou Loja) e permitir que o usuário escolha entre essas opções para criar uma conta no sistema. Ela oferece uma interface simples de interação com o usuário, onde o mesmo pode optar por se registrar como Comprador, Loja ou voltar para o menu anterior.

A classe contém o método exibirMenuRegistro, que exibe as opções de registro disponíveis. Quando o usuário escolhe uma das opções (1 para se registrar como Comprador, 2 para se registrar como Loja, ou 0 para voltar), o sistema direciona para a funcionalidade correspondente. Por exemplo, ao escolher a opção 1, o sistema chama o método cadastrarComprador da classe CompradorView para registrar um novo Comprador. Da mesma forma, ao escolher a opção 2, o método cadastrarLoja da classe LojaView é chamado para registrar uma nova Loja. Se a opção 0 for selecionada, o método simplesmente retorna, voltando para o menu anterior.

Além disso, a classe também contém o método menuInicial, que exibe um menu inicial com opções de Login e Registro. Esse menu permite que o usuário escolha entre realizar um login no sistema ou se registrar como Comprador ou Loja. Caso o usuário escolha o registro, a classe MenuRegistro é chamada novamente para que o usuário faça sua escolha de registro.

Em resumo, a classe MenuRegistro serve para facilitar o processo de registro de novos usuários no sistema, oferecendo uma interface simples para registrar Compradores e Lojas e direcionando o fluxo de navegação do usuário conforme suas escolhas.

Exemplo registro de comprador:

```
=== Bem-vindo ===
1. Login
2. Registrar
0. Sair
Escolha uma opção: 2

=== Registro ===
1. Registrar como Comprador
2. Registrar como Loja
0. Voltar
Escolha uma opção: 1
=== Cadastro de Comprador ===
Nome: leticia
E-mail: leticia@gmail.com
Senha (mínimo 6 caracteres): leticia
CPF (XXX.XXX.XXX-XX): 123.456.789-10
Endereço: Rua dos Ipes - 55
Comprador adicionado com sucesso.
Cadastro realizado com sucesso!
```

Fachada para os menus

A classe FachadaMenus tem como objetivo centralizar e gerenciar os menus de navegação do sistema, direcionando os usuários para as opções apropriadas conforme seu tipo de login, seja Administrador, Comprador ou Loja. Ela funciona como uma fachada que organiza a interação entre o usuário e o sistema, oferecendo menus específicos para cada perfil e garantindo que cada um tenha acesso apenas às opções que correspondem ao seu papel.

Essa classe contém diversos métodos de menu, cada um projetado para um contexto específico. O menuComprador, por exemplo, é exibido quando o usuário está logado como Comprador, oferecendo opções para cadastrar, listar, atualizar ou deletar compradores. O menuLojaCompleto é destinado aos usuários logados como Loja, permitindo o gerenciamento de lojas com opções para cadastrar, deletar, listar, buscar e atualizar lojas. Já o menuLojaParaCompradores é acessado por Compradores, que podem listar todas as lojas ou buscar uma loja específica usando o CPF/CNPJ.

O menuProdutoParaAdmin é voltado para os Administradores e oferece opções para gerenciar produtos em todas as lojas, enquanto o menuProdutoParaLojas é acessado pelas Lojas, que podem cadastrar, listar, buscar, atualizar ou deletar produtos dentro de sua própria loja. O menuProdutoParaComprador, por sua vez, é voltado para Compradores, permitindo que listem produtos por loja, vejam todos os produtos ou busquem produtos específicos.

Além disso, a classe possui o método MenuSelecionador, que serve como um ponto central para redirecionar os usuários para os menus corretos de acordo com seu tipo de login. Esse método garante que, ao fazer o login, o usuário seja direcionado para um menu específico com base em seu perfil. A classe também permite que o usuário faça logout, retornando ao estado inicial do sistema.

Em resumo, a classe FachadaMenus organiza e exibe menus distintos para diferentes tipos de usuários no sistema, proporcionando uma navegação estruturada e garantindo que cada usuário tenha acesso às opções adequadas ao seu papel.

Diagramas

1. Casos de uso

Usuário geral

useCaseUsuario.drawio - draw.io

Usuário comprador

useCaseComprador.drawio - draw.io

Usuário lojista

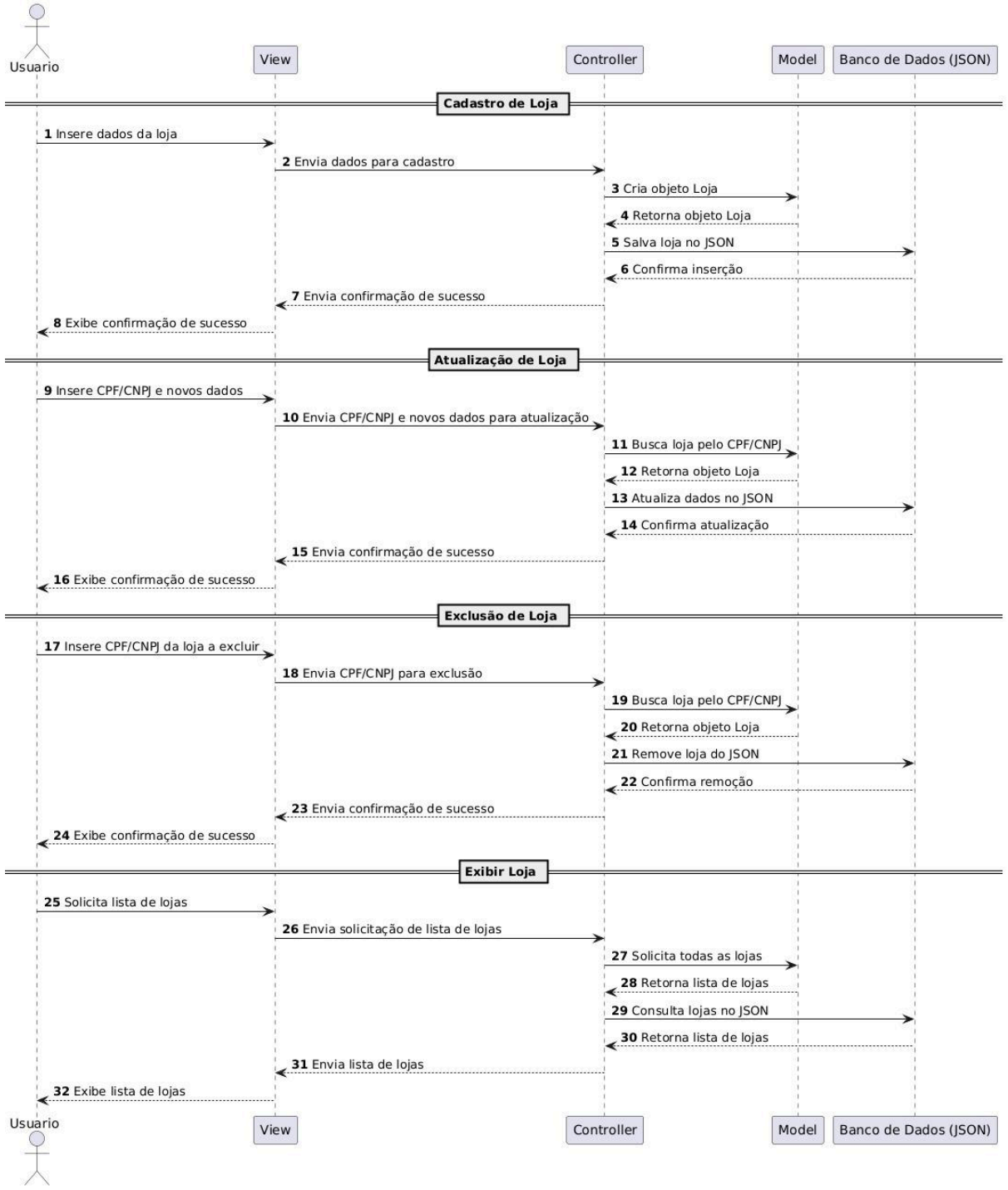
useCaseLojista.drawio - draw.io

Usuário administrador

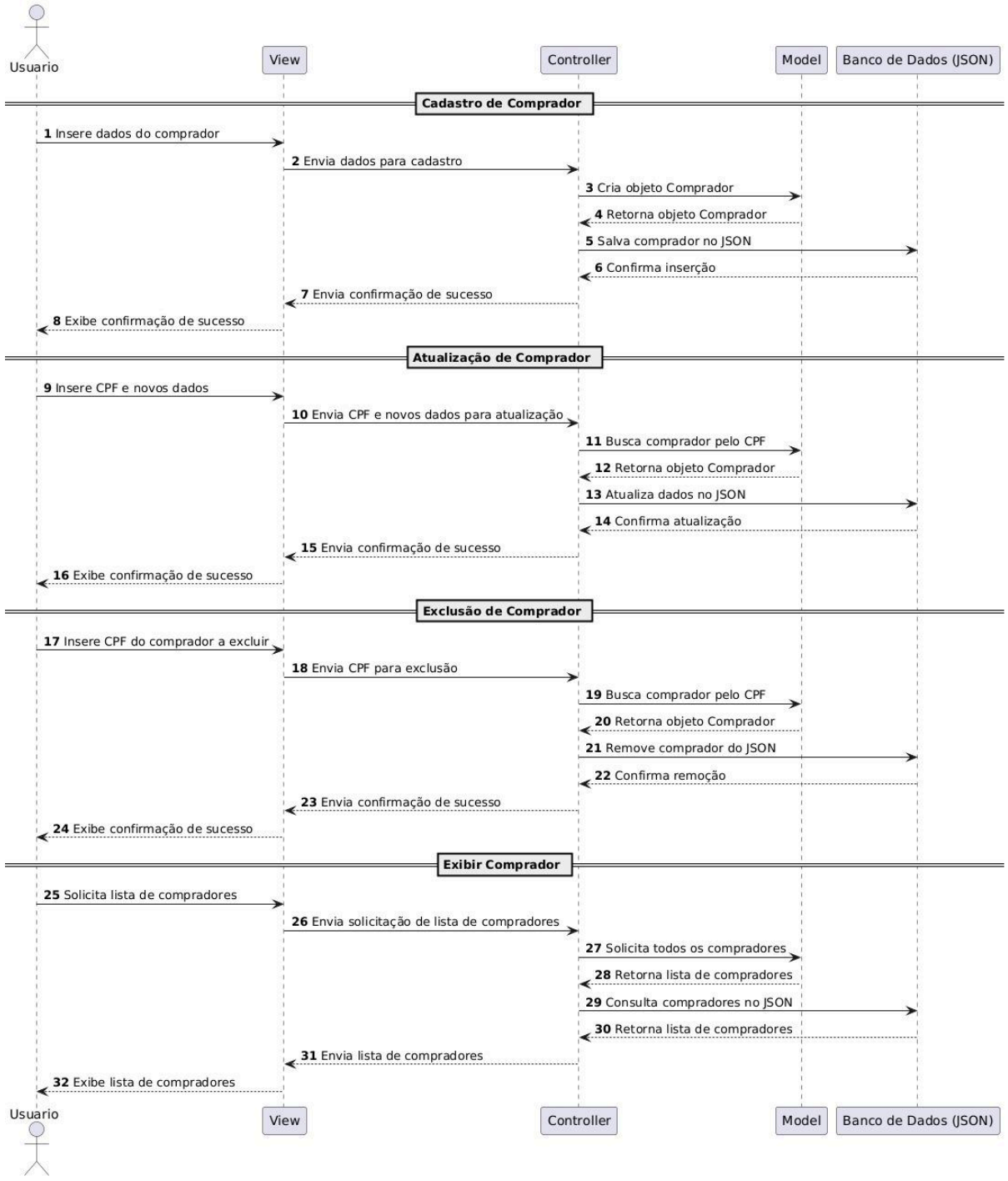
useCaseAdmin.drawio - draw.io

2. Sequência

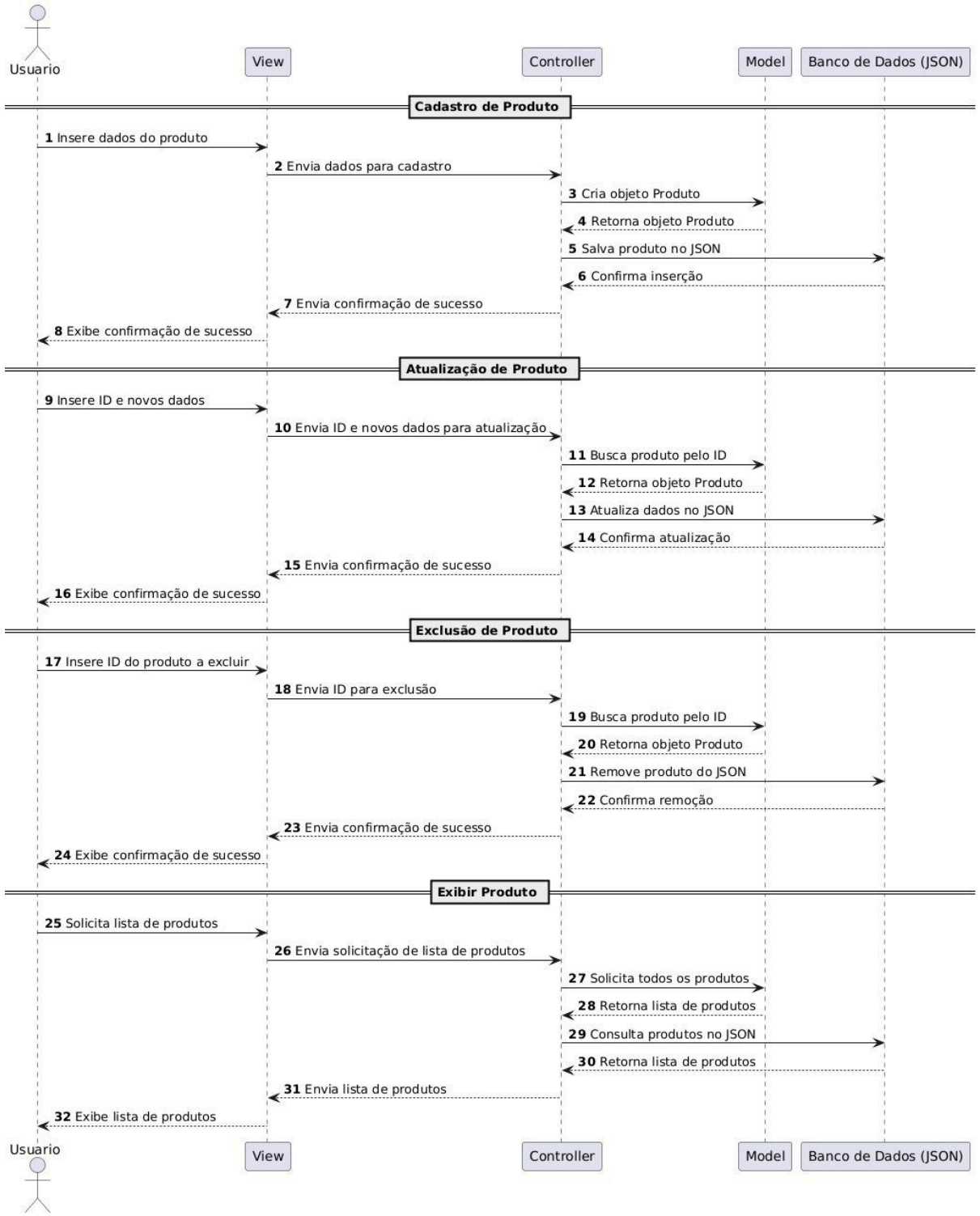
Usuário lojista



Usuário comprador



Produtos


Usuario
View
Controller
Model
Banco de Dados (JSON)
Cadastro de Produto
1 Insere dados do produto
2 Envia dados para cadastro
3 Cria objeto Produto
4 Retorna objeto Produto
5 Salva produto no JSON
6 Confirma inserção
7 Envia confirmação de sucesso
8 Exibe confirmação de sucesso
Atualização de Produto
9 Insere ID e novos dados
10 Envia ID e novos dados para atualização
11 Busca produto pelo ID
12 Retorna objeto Produto
13 Atualiza dados no JSON
14 Confirma atualização
15 Envia confirmação de sucesso
16 Exibe confirmação de sucesso
Exclusão de Produto
17 Insere ID do produto a excluir
18 Envia ID para exclusão
19 Busca produto pelo ID
20 Retorna objeto Produto
21 Remove produto do JSON
22 Confirma remoção
23 Envia confirmação de sucesso
24 Exibe confirmação de sucesso
Exibir Produto
25 Solicita lista de produtos
26 Envia solicitação de lista de produtos
27 Solicita todos os produtos
28 Retorna lista de produtos
29 Consulta produtos no JSON
30 Retorna lista de produtos
31 Envia lista de produtos
32 Exibe lista de produtos
Usuario
View
Controller
Model
Banco de Dados (JSON)

Login como Lojista

SequencyLoginLojista.drawio - draw.io

Login como Comprador

SequencyLoginComprador.drawio - draw.io

Login como Administrador

SequencyLoginAdmin.drawio - draw.io

3. Diagrama de classes

diagramaClasses.pdf