Especificação dos Requisitos

do

Sistema de Controle de Estoque de Loja de Cosméticos

Versão <3.0>

João Otávio Biondo	**<81746>**
Marco Aurélio Deoldoto Paulino	**<82473 >**
Pedro Barbieiro	**<68086 >**
Raphael Augusto Granado Vieira	**<78972.>**

Professor(a): **Donizete Carlos Bruzarosco**

Disciplina: **Implementação de Sistemas de Software**

Sumário

Revisões

Versão	Autores	Descrição da Versão	Data
1.0	Marco Aurélio Raphael Ganado Vinicis Medeiros Missima Jecohti	Versão inicial com objetivo de auxilar uma Loja de Cosméticos a gerenciar o estoque, através do cadastramento e movimentos dos produtos e cadastro de Nota Fiscais	23/05/2014
2.0	Marco Aurélio Raphael Vieira Pedro Barbieiro	Versão atualizada	02/05/2015
3.0	João Otávio Marco Aurélio Raphael Vieira Pedro Barbieiro	Versão corrigida e modificada para o uso do Scrum	20/10/2015

1 Introdução

O produto detalhado neste documento consiste em um software gerenciador de estoque de lojas.

1.1 Objetivo do Documento

Este presente documento tem como objetivo descrever o conjunto de requisitos do sistemas de modo a contribuir com o andamento do desenvolvimento.

1.2 Escopo do Produto

O produto visa em contribuir no gerenciamento do estoque de uma loja, contribuindo na área de compra e vendas de produtos, além de pagamentos. Os benefícios na utilização deste software estão na automatização do trabalho e assim diminuir o custo de realizar tais operações, além de tornar a realização das operações mais confiáveis, devido a menor probabilidade de um erro humano.

1.3 Público-Alvo

Proprietários de estabelecimentos comerciais, principalmente comércio de pequeno ou médio porte.

2 Visão Geral

2.1 Perspectiva do Produto

O desenvolvimento do software, bem como já dito, visa no auxílio no gerenciamento do estoque em lojas, a importância em ter um software para esta área está na questão de diminuir o tempo e a complexidades nas tarefas de compras e vendas, baixa e retorno de produtos, e pagamento. Com a queda na complexidade das tarefas, o perigo de ocorrer erros durante as operações também diminuem e assim a confiabilidade aumenta.

2.2 Funcionalidade do Produto

A seguir serão apresentadas as principais funcionalidades do produto com uma breve de seu objetivo.

Gerenciar Estoque: contempla as operações necessárias para o funcionamento do estoque, como por exemplo, criar, excluir, inserir, alterar e buscar um produto do estoque.

Gerenciar Acesso: controle no acesso as demais funcionalidade do sistema.

Gerenciar Compras: funcionalidade na qual visa o controle dos produtos comprados pela própria loja.

Retorno Compras: tem como objetivo cuidar da parte relacionada a devolução de produtos da loja à fornecedores em casos como defeito ou envio de produtos errados.

Gerenciar Vendas: visa em gerenciar a parte de vendas efetuadas pela loja.

Devolução de Produtos: visa em controlar a parte de produtos devolvidos pelos clientes após uma compra.

Pagamento: cuida da parte do pagamento escolhido pelo cliente, seja a data do pagamento (à vista ou a prazo) ou a forma de pagamento (em dinheiro ou cartão de crédito).

O sistema permitirá o cadastro de clientes para a funcionalidade de Vendas e de Fornecedores para a funcionalidade de Compras, além de ter

2.3 Usuários

Os usuários do sistema serão o gerente e funcionários da loja.

2.4 Ambiente Operacional

Computador com acesso a internet, sem restrições a sistema operacional e hardware. Servidor para armazenar o software com também acesso a internet com um armazenamento interno de no mínimo, 2 Terabytes.

2.5 Restrições de Projeto e Implementação

A única limitação será no espaço de armazenamento, devido a necessidade de armazenamento dos dados por um período de tempo razoável, por isso será necessário backups dos dados e compras de disco rígidos de armazenamento quando necessário. A linguagem utilizada na implementação será o Java, e será utilizado o protocolo TCP/IP para a comunicação dos computadores e servidor.

2.6 Documentação do Usuário

Será necessário apenas um guia de utilização do software.

3 Especificação das Interfaces Externas

3.1 Requisitos de Interface Externa

3.1.1 Interfaces do Usuário

Tela de autenticação, tela de vendas e compras de produtos, devolução de produtos e retorno de compras, além do pagamento.

3.1.2 Interfaces de Hardware

Os computadores serão conectados ao servidor via Internet.

3.1.3 Interfaces de Software

Será utilizado o protocolo TCP/IP para a comunicação entre os computadores.

3.1.4 Interfaces de Comunicação

Será utilizada a arquitetura cliente-servidor, sendo utilizado o cliente “gordo”.

4 Requisitos Funcionais

4.1 RF001 <Gerenciar Estoque >

A funcionalidade de Gerenciar Estoque tem como objetivo controlar os produtos cadastrados no sistema mantendo o sistema consistente com o estoque real da loja.

Os dados de entradas serão o produto, via o código identificador e em casos de alteração, também será passado o novo valor para o campo.

Operações básicas: cadastro, remoção, busca e alteração de produtos, verificar a quantidade de um produto, atualizar específicos de um produto, tal como, quantidade, preço e nome.

Os dados de saída será o produto em casos de busca e em casos de inserção, deleção ou edição será um valor booleano informando o sucesso ou não da operação.

4.2 RF002 <Gerenciar Compra >

A funcionalidade de Gerenciar Compra tem como objetivo cuidar dos aspectos relacionados a compras efetuadas pela loja, tal como o cadastro dos produtos comprados no sistema, além do fornecedor de quem foi comprado.

Os dados de entrada desta funcionalidade será o fornecedor e o pedido da compra. O pedido, nada mais é, uma lista contendo os produtos comprados, além da data da compra e o valor total.

Operações básicas: cadastro, remoção, busca e alteração do Fornecedor e do Pedido.

Os dados de saída serão o Fornecedor ou Pedido em casos de buscas, para operações de inserção, deleção ou edição será retornado um valor booleano informação o sucesso da operação.

4.3 RF003 < Retorno Compras >

A funcionalidade de Gerenciar Retorno Compra tem como objetivo de cuidar da parte relacionada a retorno de compras efetuadas pela loja ao fornecedor.

Os dados de entrada será o pedido na qual será retornado.

Operações básicas: busca, cancela e altera Pedido

Dados de saída: O próprio pedido em caso de busca por ele, nas demais operações será retornado o valor booleano informando o sucesso da operação

4.4 RF004 <Gerenciar Acesso >

A funcionalidade de Gerenciar Acesso tem como objetivo de cuidar da parte relacionada ao acesso do sistema e assim não permitir a entrada de invasores ao sistema garantindo segurança aos dados armazenados.

Dados de entrada: Usuário, bem como cpf, nome, login e senha. O usuário pode ser do tipo Gerente ou Funcionário

Operações básicas: verifica, cria, altera, exclui e busca usuários.

Dados de saída: o próprio usuário ao ser realizado uma busca e nas demais operações valor booleano informando o sucesso na operação.

4.5 RF005 <Gerenciar Venda >

A funcionalidade de Venda tem como objetivo controlar as vendas efetuadas pela loja, garantindo que compatibilidade entre os valores do sistema com os valores reais.

Dados de entrada: A venda, bem como, o código da venda, o cliente para quem vendeu, lista de produtos vendidos, valor total de venda.

Operações: recupera Venda, fecha Venda, cancela Venda, altera Venda e altera Venda.

Dados de saída: valor booleano informando o sucesso na operação, exceto no recupera Venda na qual retorna a própria venda.

4.6 RF006 <Gerenciar Pagamento>

A funcionalidade de Gerenciar Pagamento tem como objetivo controlar a parte do pagamento realizado pelo cliente após a venda. Para esta funcionalidade utilizaremos o padrão de projeto State para auxiliar na situação de pagamento.

Dados de entrada: o pagamento, bem como, seu código de identificação, data de vencimento, cliente relacionado e tipo do pagamento.

Operações básicas: gera pagamento, fecha pagamento, busca, altera, cancela e efetua pagamento.

Dados de saída: o próprio pagamento em caso de busca e valor booleano para as demais operações.

4.7 RF007 <Gerenciar Devolução de Produtos>

A funcionalidade de Gerenciar Devolução de Produtos tem como objetivo controlar a parte de devolução de produtos pelo cliente após uma compra.

Dados de entrada: A venda, via o seu código de identificação.

Operações: recupera Venda, cancela Venda e altera Venda.

Dados de saída: a própria Venda para recupera Venda e valor booleano informando o sucesso da operação para as demais operações.

5 Relação dos Requisistos priorizados

Gerenciar Vendas;

Gerenciar Compras;

Gerenciar Estoque;

Pagamento;

Devolução de Produtos;

Retorno Compra;

Gerenciar Usuário;

Gerenciar Clientes;

Autenticar Usuário;

6 Planejamento das Sprints

Sprint	João Otávio	Marco Aurélio	Pedro Barbiero	Raphael Augusto
1ª	Gerenciar e Autenticar Usuários	Gerenciar Vendas	Gerenciar Estoque	Gerenciar Clientes
2ª	Gerenciar Pagamento	Devolução de Produtos	Retorno de Compras	Compra de Produtos

7 Protótipo de Interfaces

Para a criação das interfaces com o usuário foi utilizada a ferramenta Pencil, disponível em http://pencil.evolus.vn/.

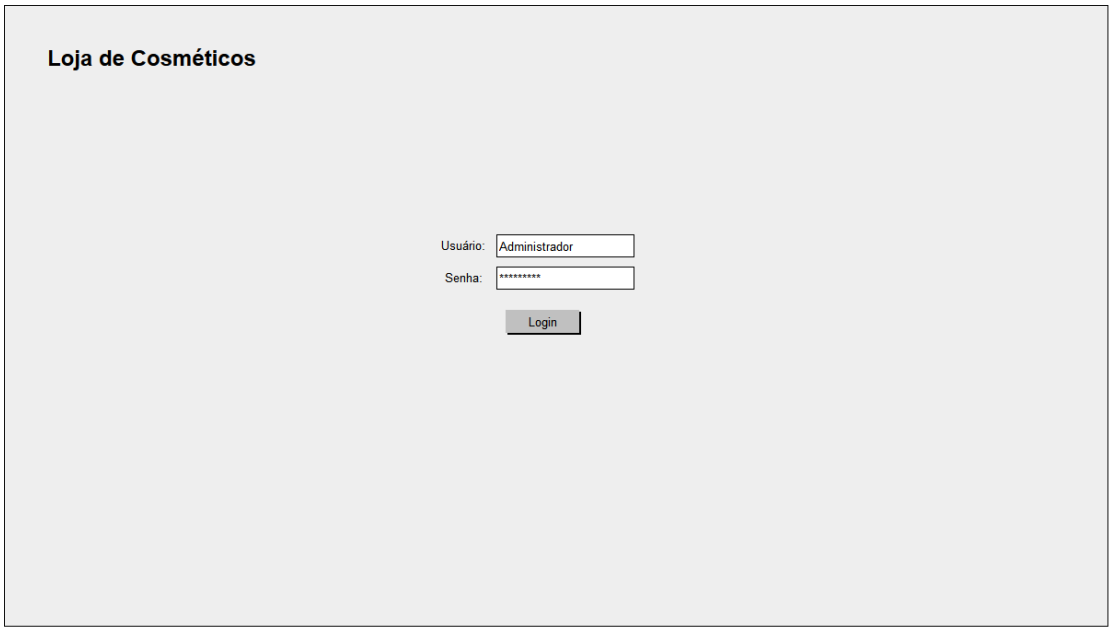

Figura 1. Tela de Login

Figura 2. Tela de Menu Inicial

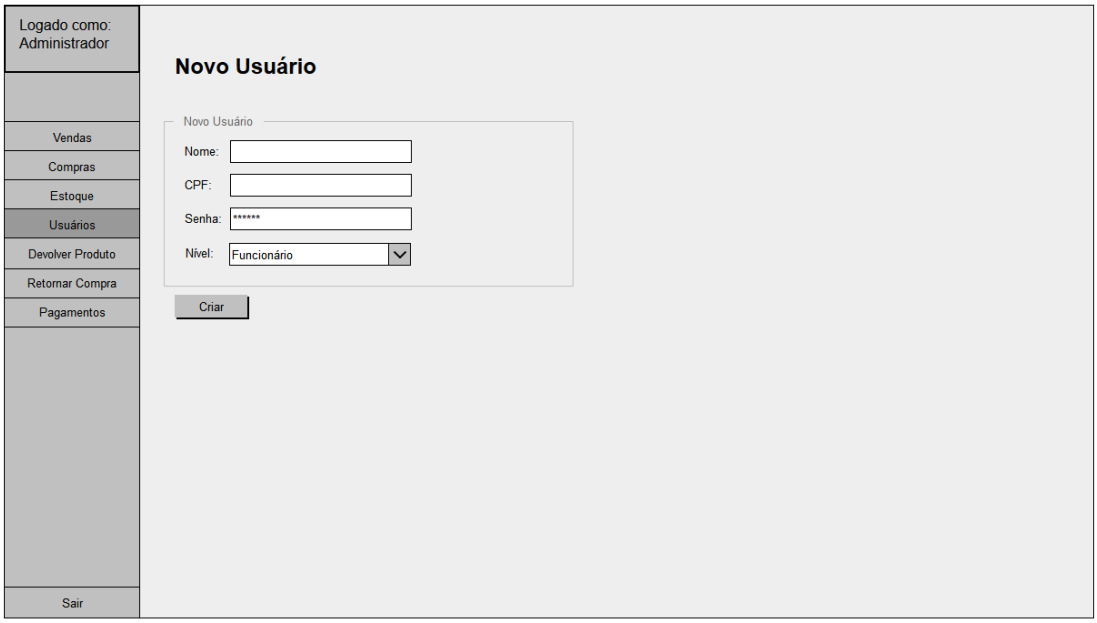

Figura 3. Tela de Criar Usuário

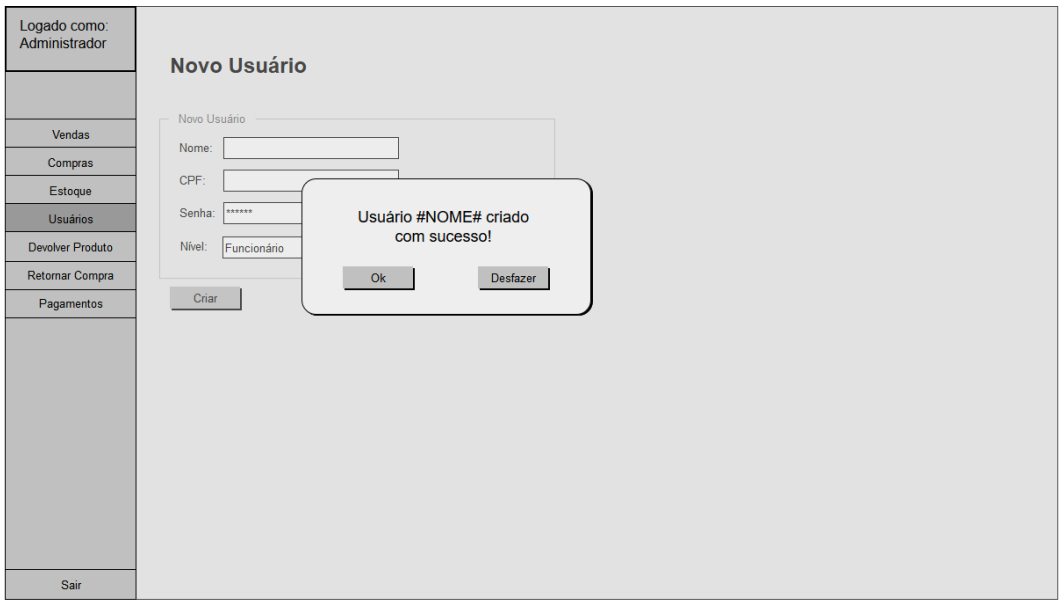

Figura 4. Tela de Confirmar Novo Usuário

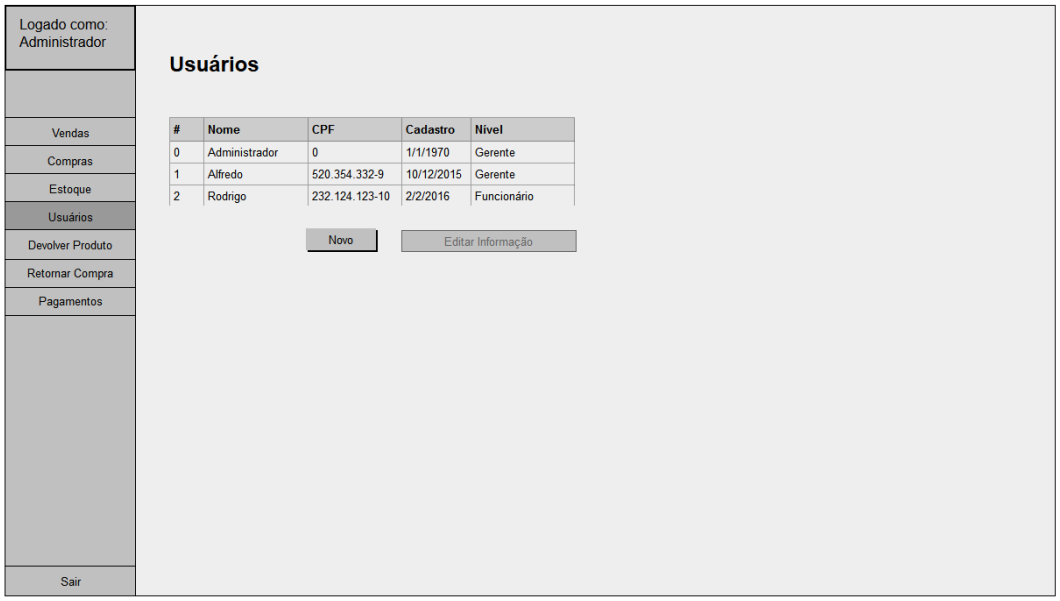

Figura 5. Tela Lista Usuários

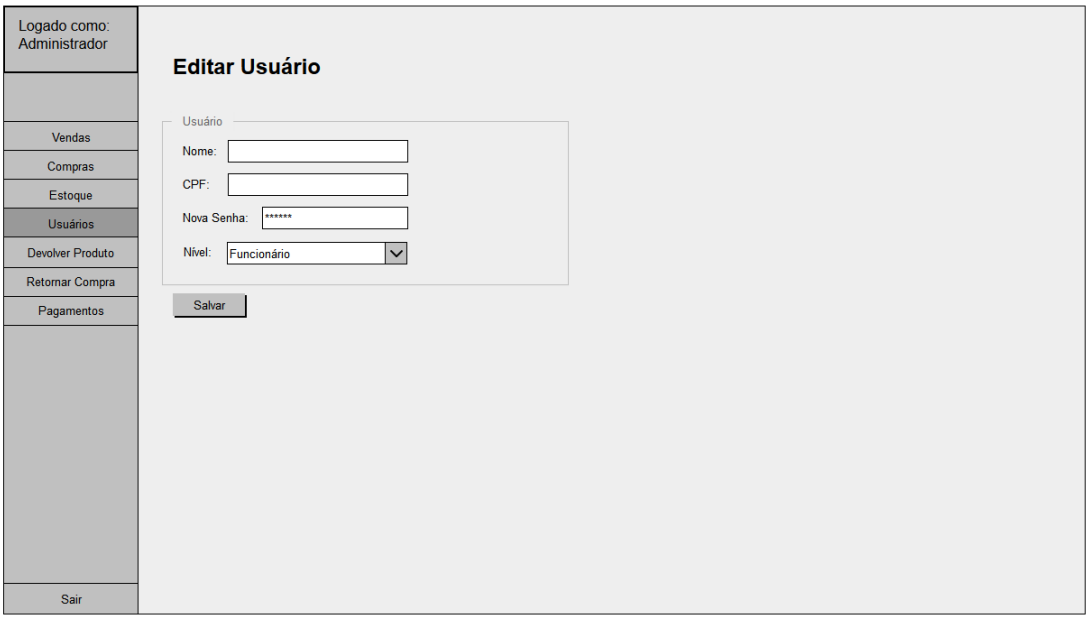

Figura 6. Tela de Editar Usuário

Figura 7. Tela Vendas

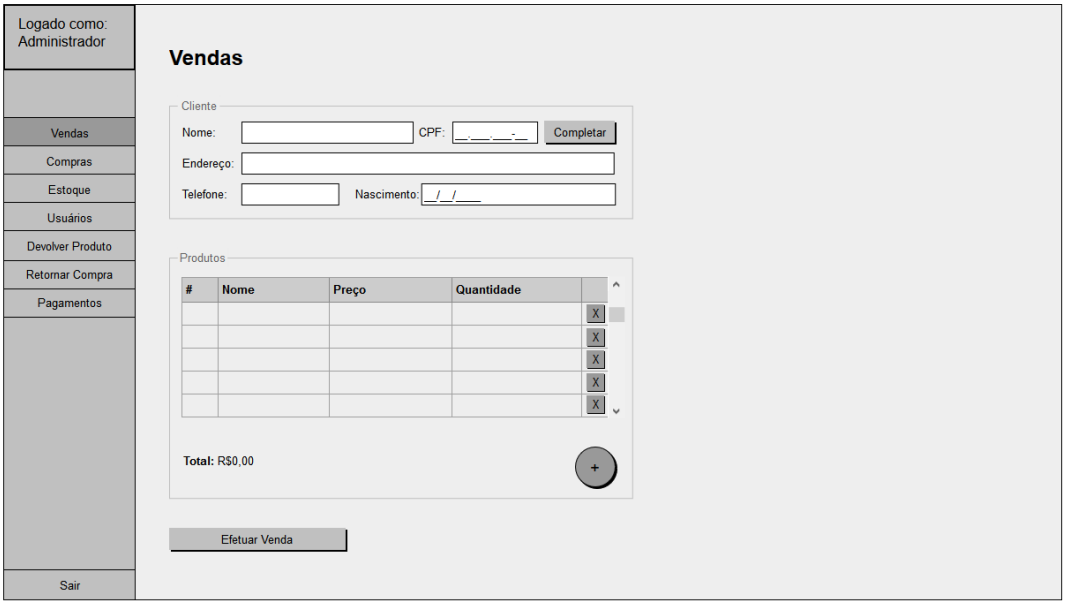

Figura 8. Tela de Cadastro de Vendas

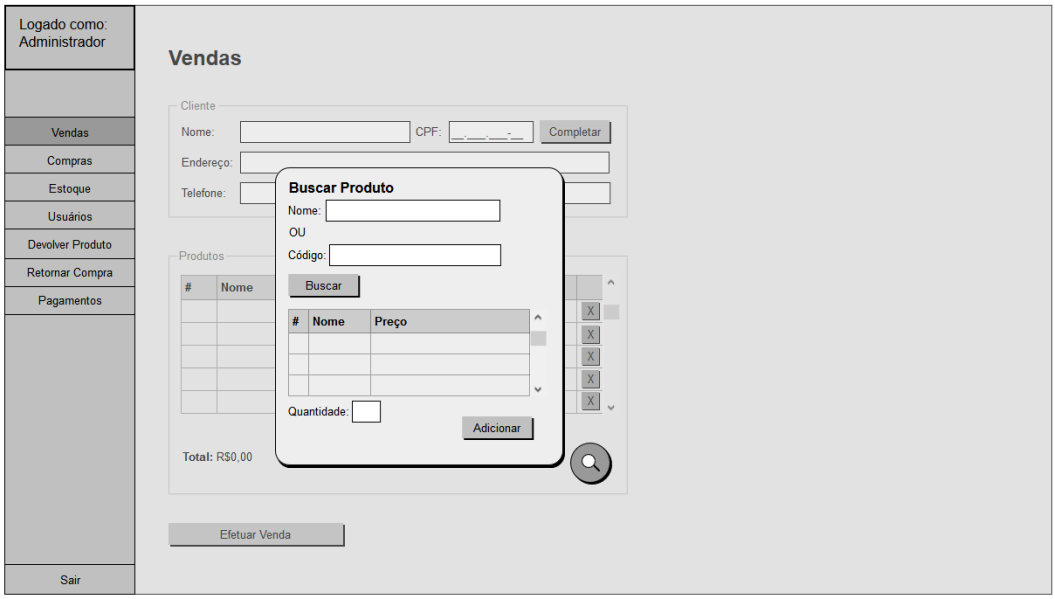

Figura 9. Tela de Buscar Produto

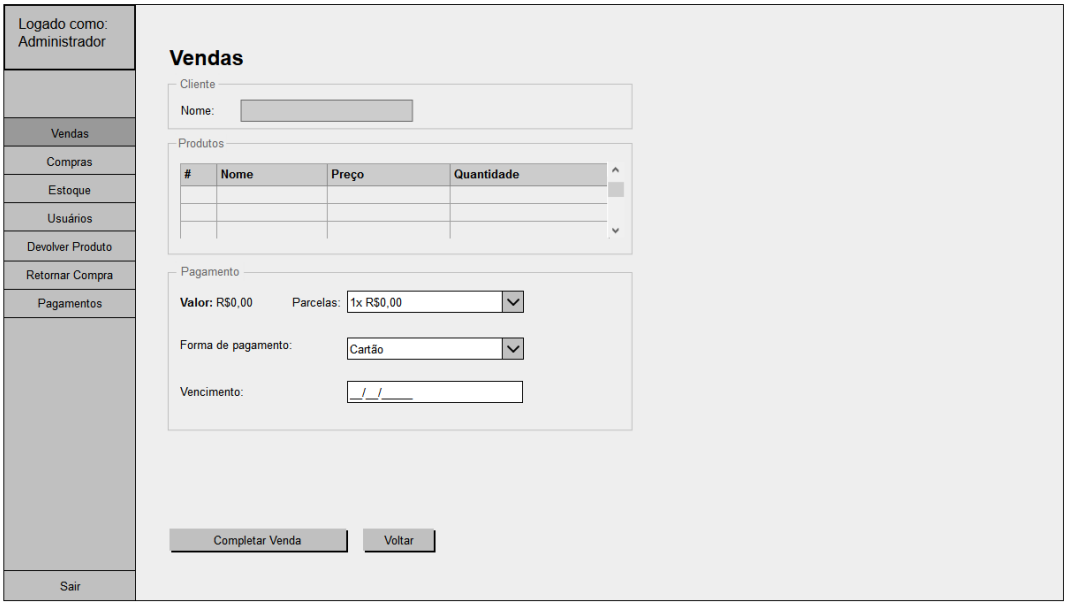

Figura 10. Tela de Escolha de Pagamento

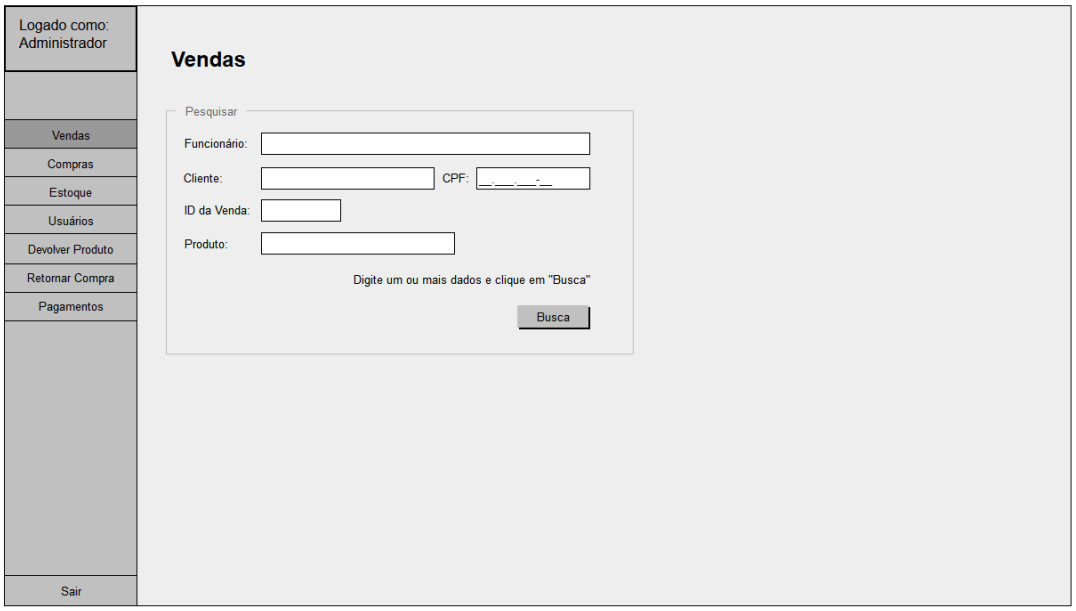

Figura 11. Tela Consulta Vendas

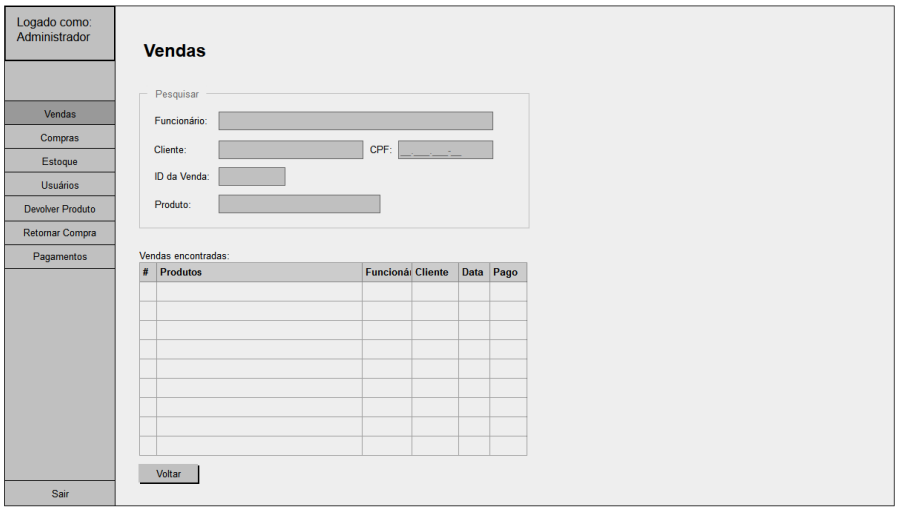

Figura 12. Tela Consulta Vendas com resultados

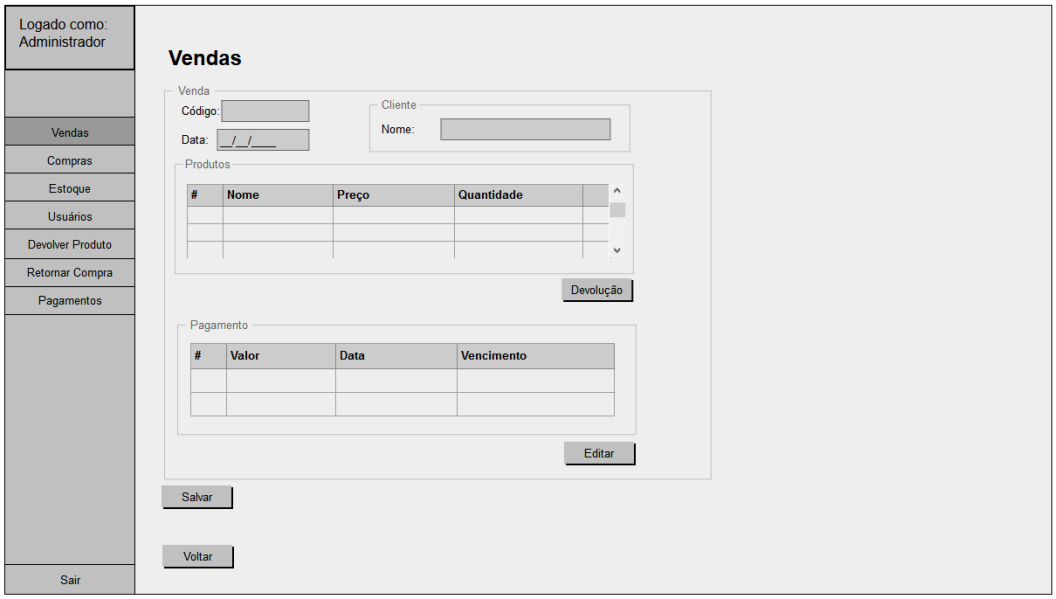

Figura 13. Tela Venda detalhada

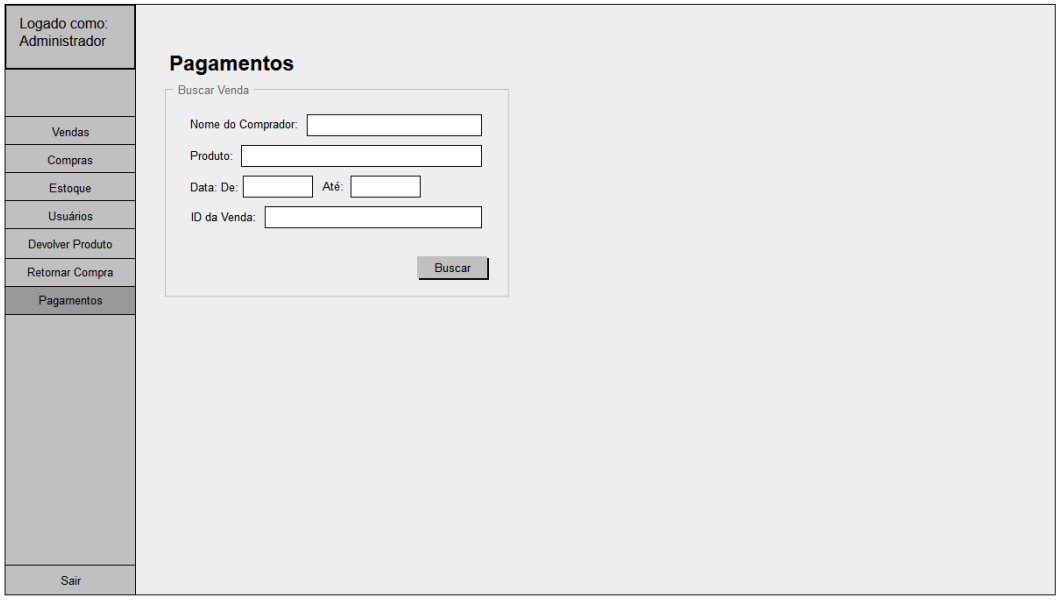

Figura 14. Tela de Busca Pagamento

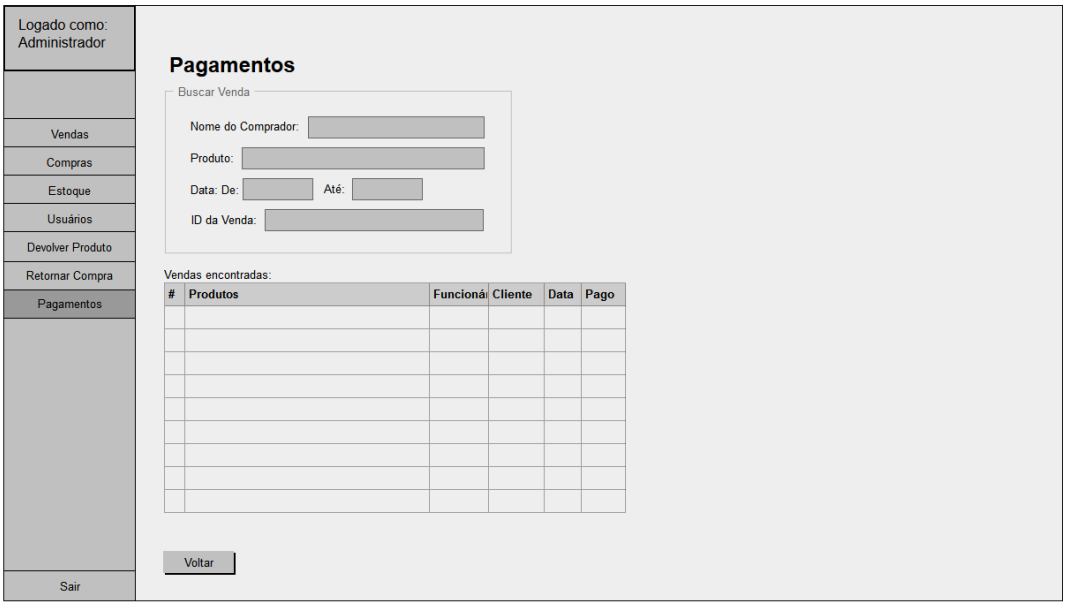

Figura 15. Tela de Vendas encontradas a partir do Pagamento

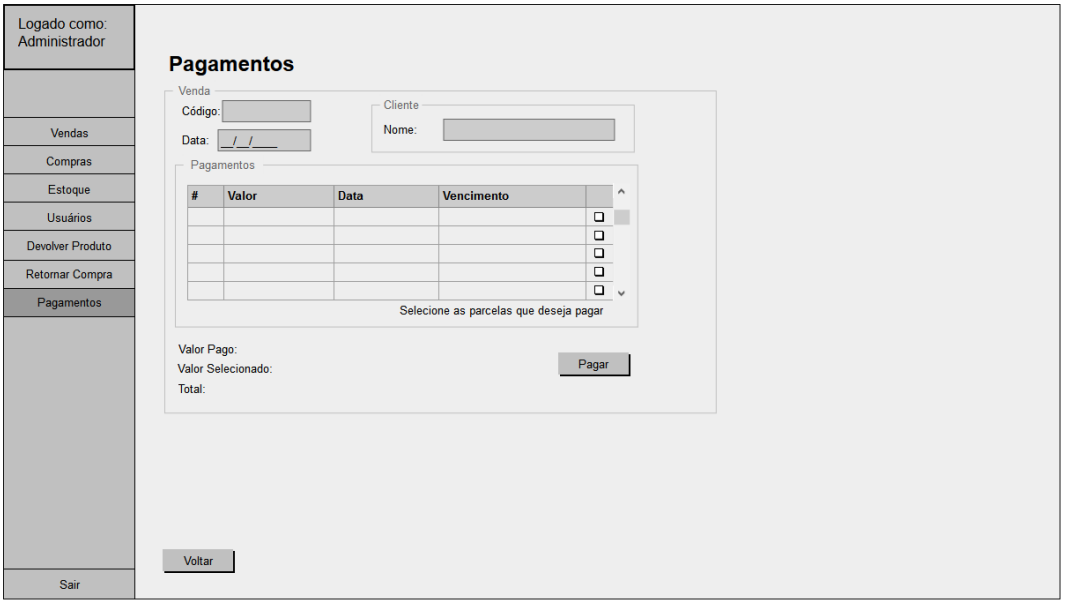

Figura 16. Detalhes das Vendas relacionadas ao Pagamento

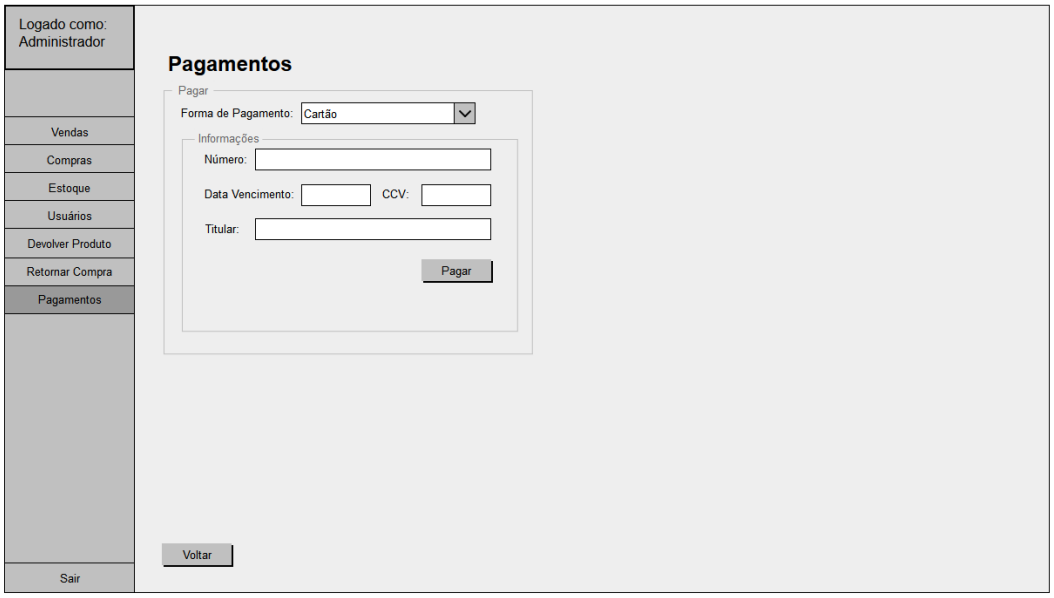

Figura 17. Tela de Opções de Pagamento

Figura 18. Tela de Compras

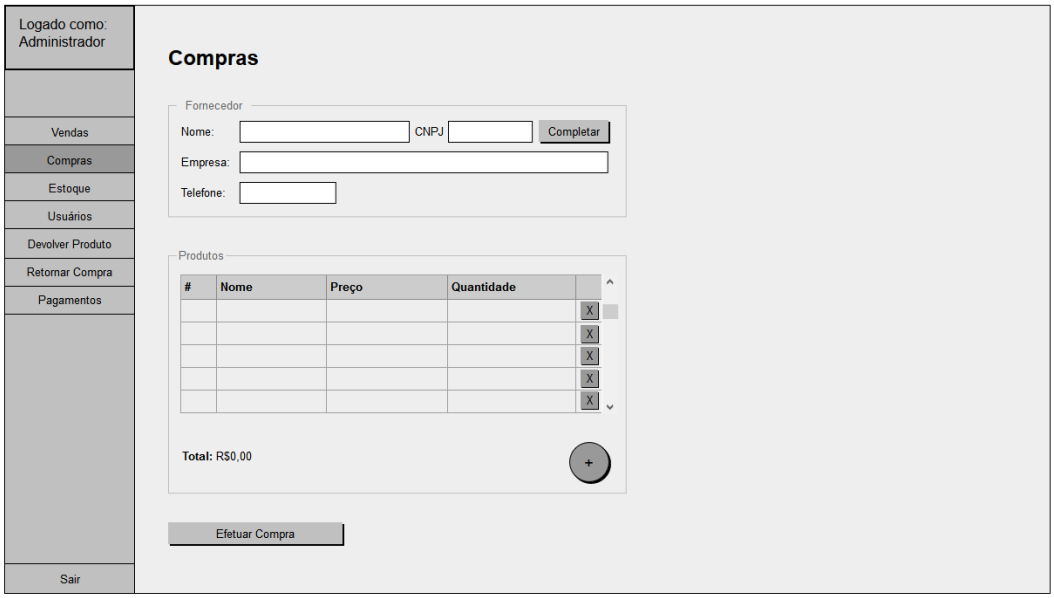

Figura 19. Tela de Cadastro de Compras

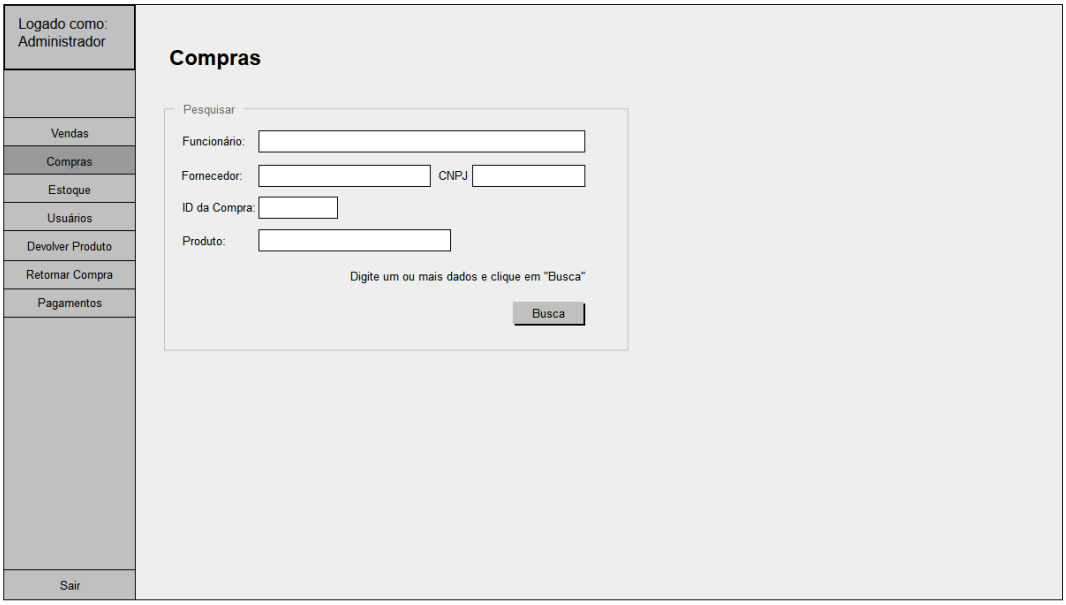

Figura 20. Tela de Consulta de Compras

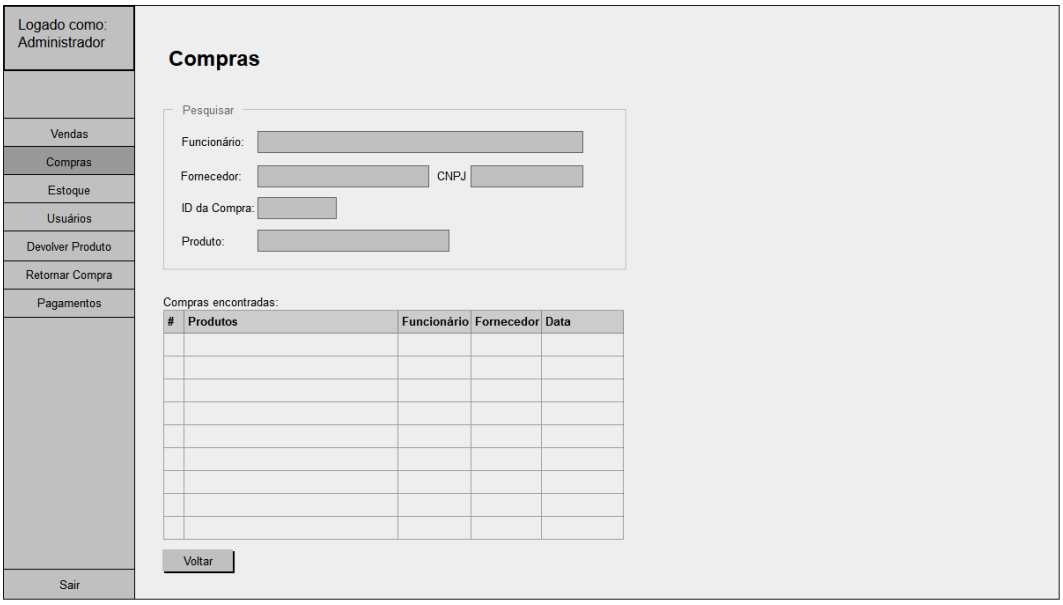

Figura 21. Tela de Resultados em Compras

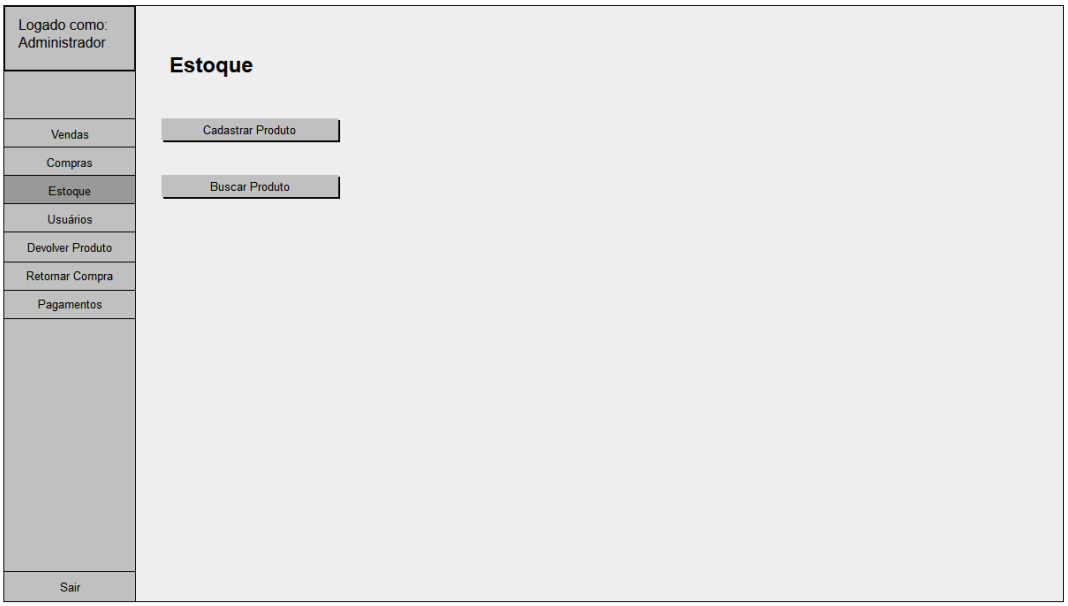

Figura 22. Tela de Estoque

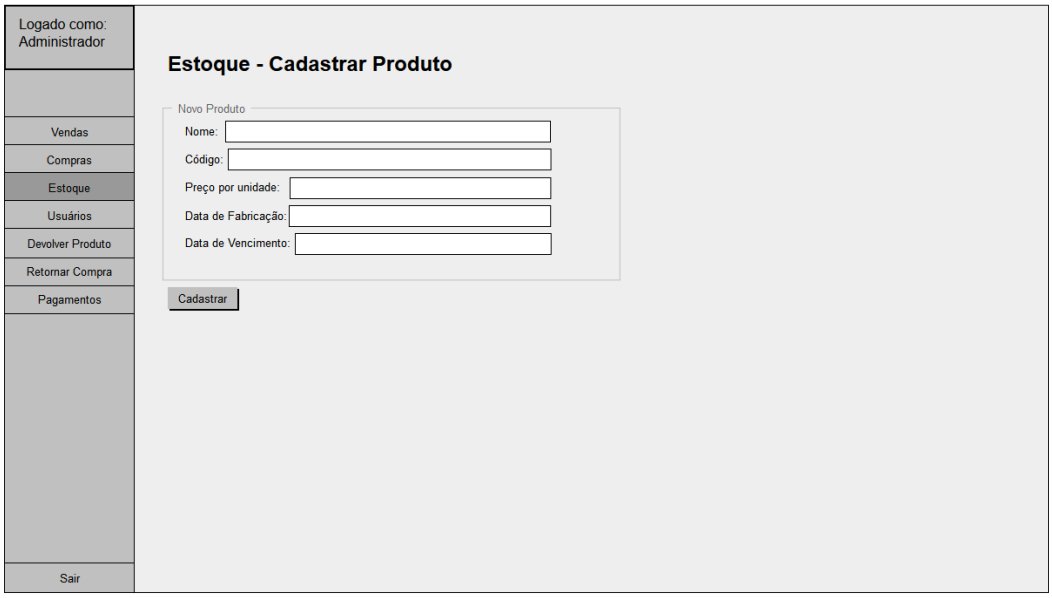

Figura 23. Tela de Cadastro de Produtos

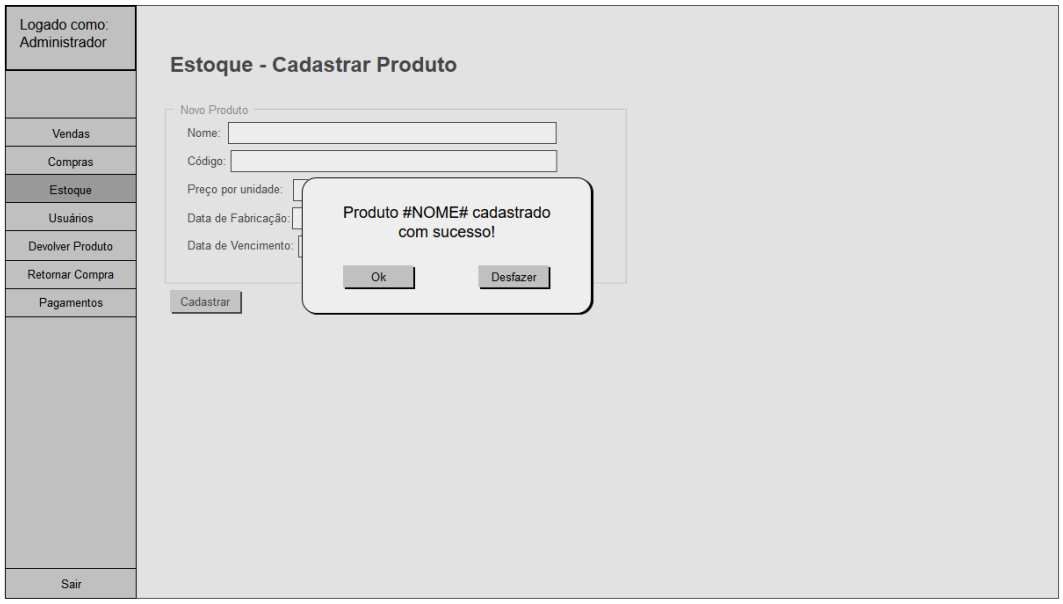

Figura 24. Tela de Confirmação de Novo Produto

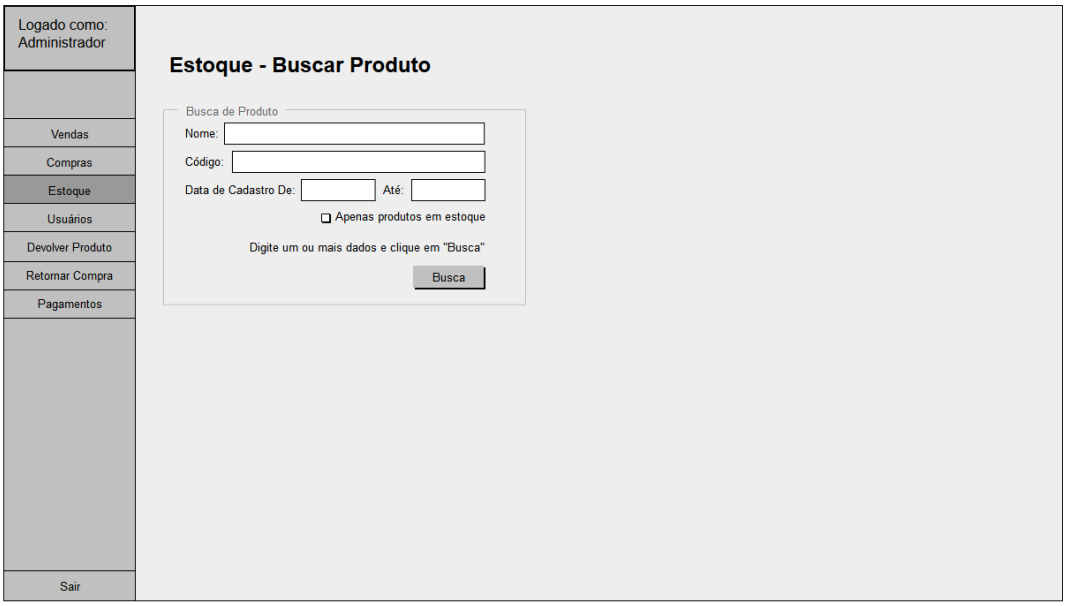

Figura 25. Tela de Busca de Produto

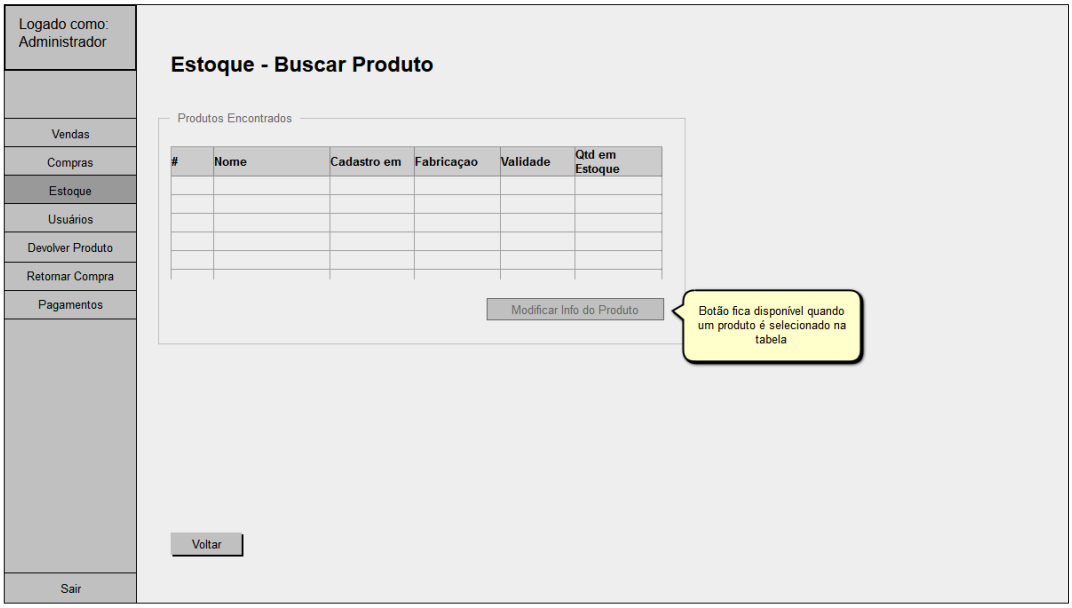

Figura 26. Tela de Resultados do Busca Produto

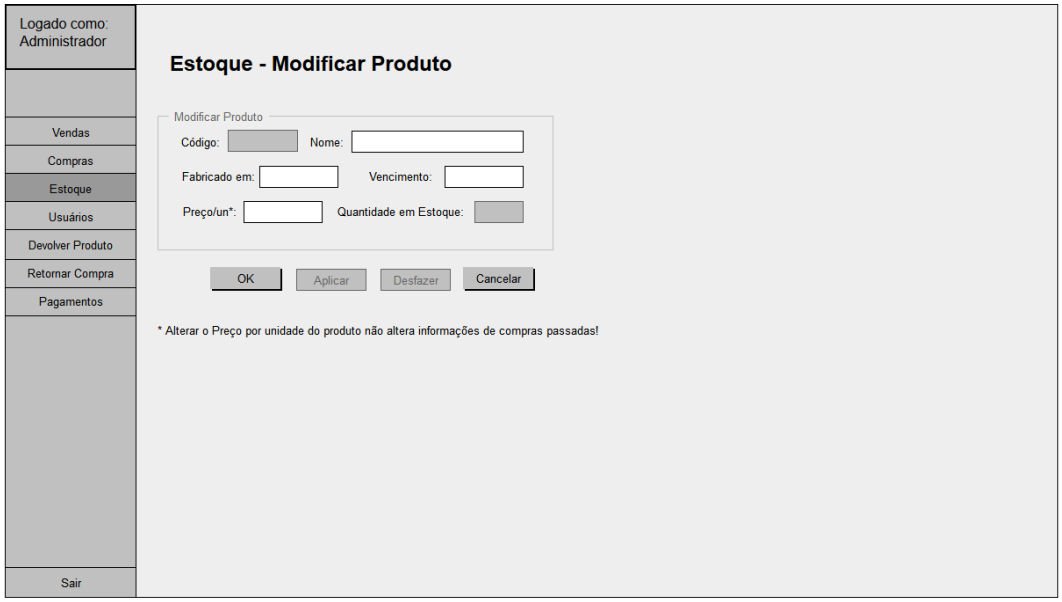

Figura 27. Tela de Alterar Produto

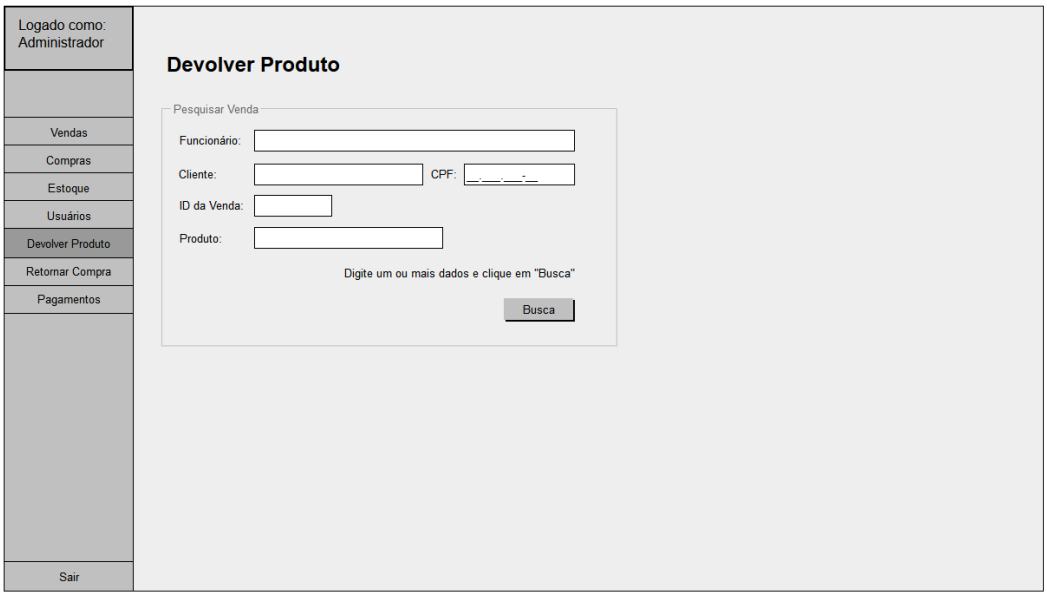

Figura 28. Tela de Devolver Produto

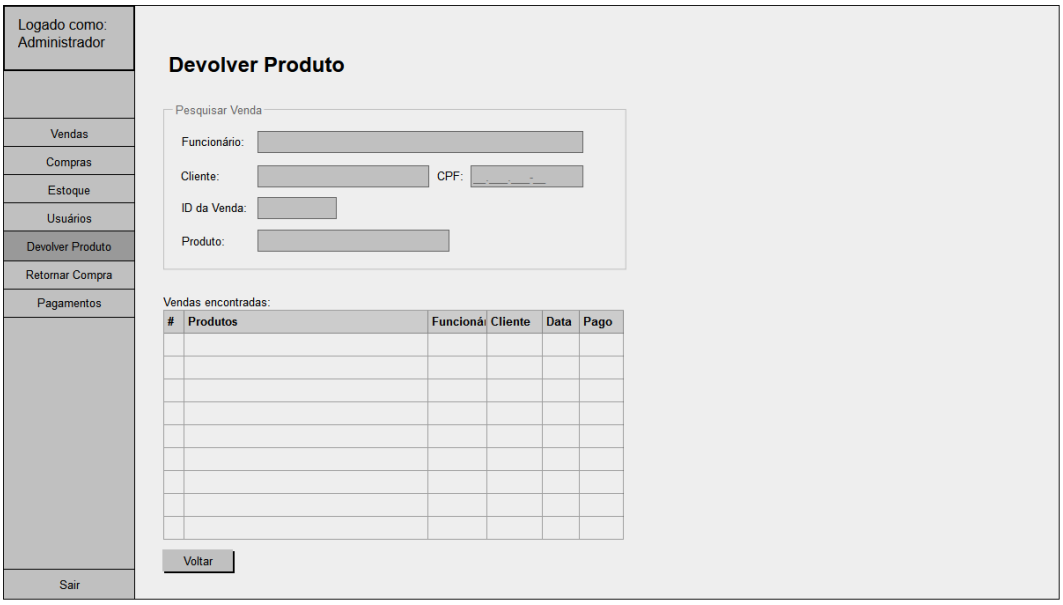

Figura 29. Tela de Vendas encontradas para haver a devolução de produtos.

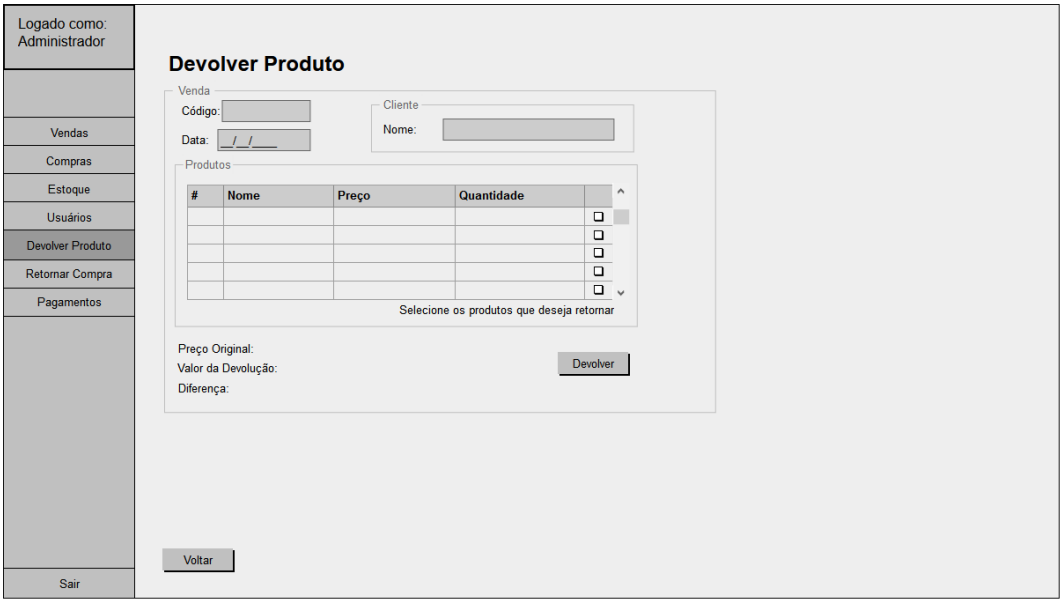

Figura 30. Tela de escolha de produtos devolvidos

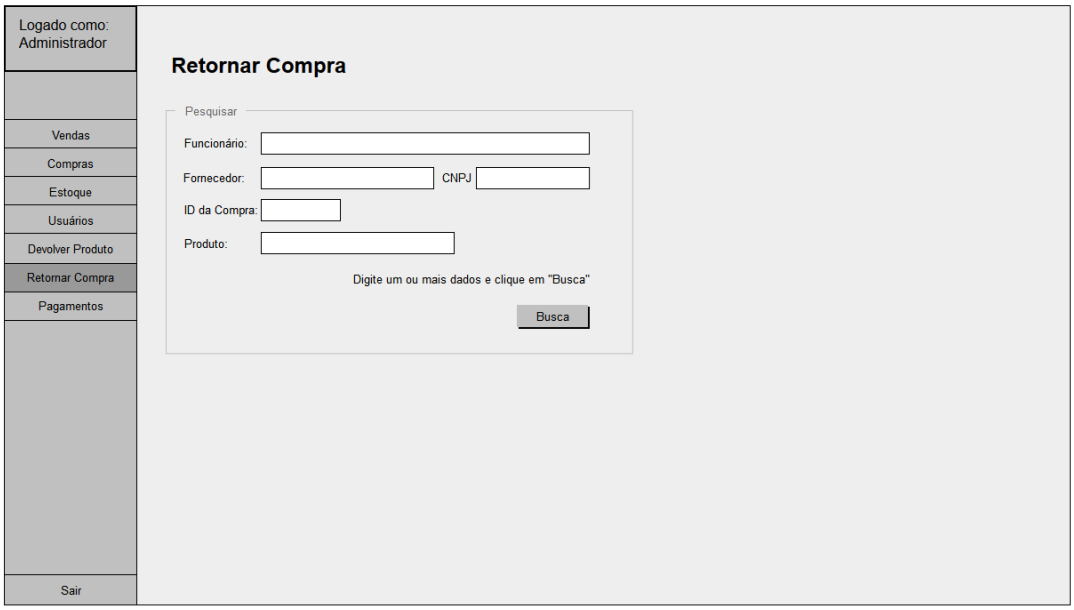

Figura 31. Tela de Retorno de Compras

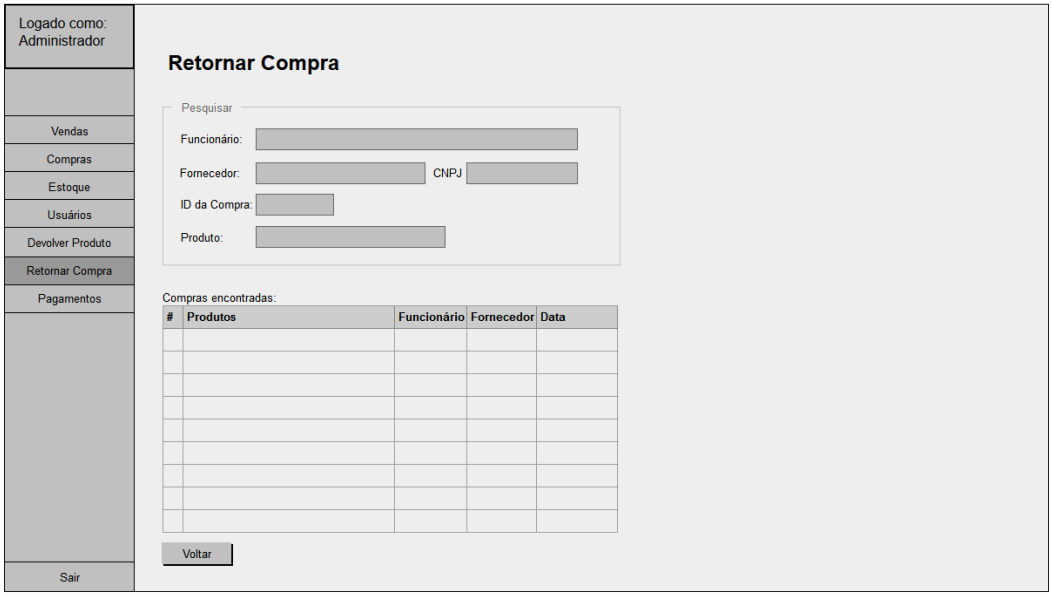

Figura 32. Tela de Resultados para Retorno de Compras

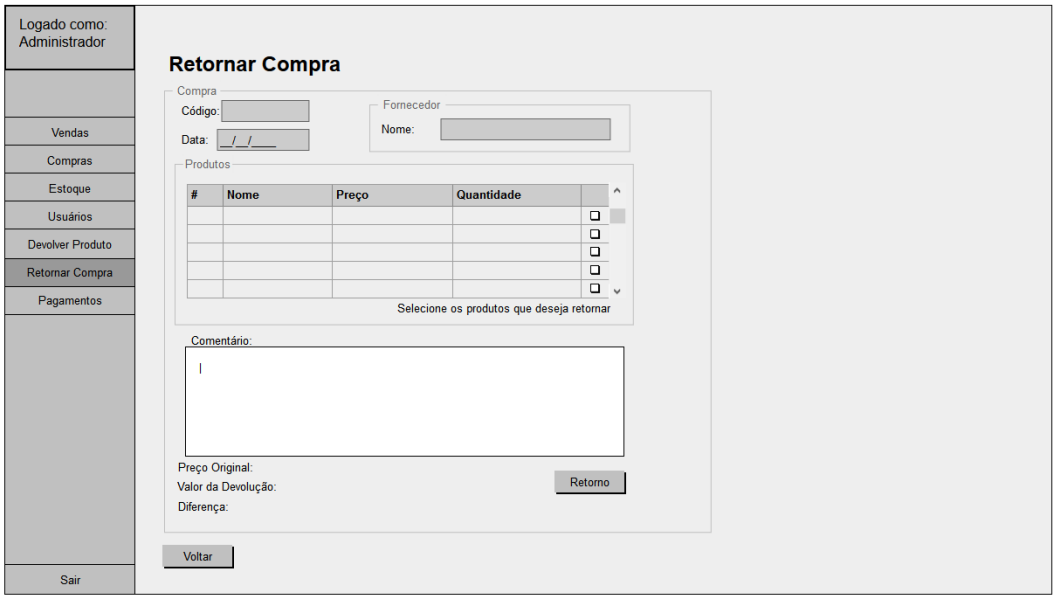

Figura 33. Tela de Compra a Retornar

8 Diagramas de Classe MVC

Os diagramas das classes MVC serão apresentados divididos para melhor visualização do mesmo.

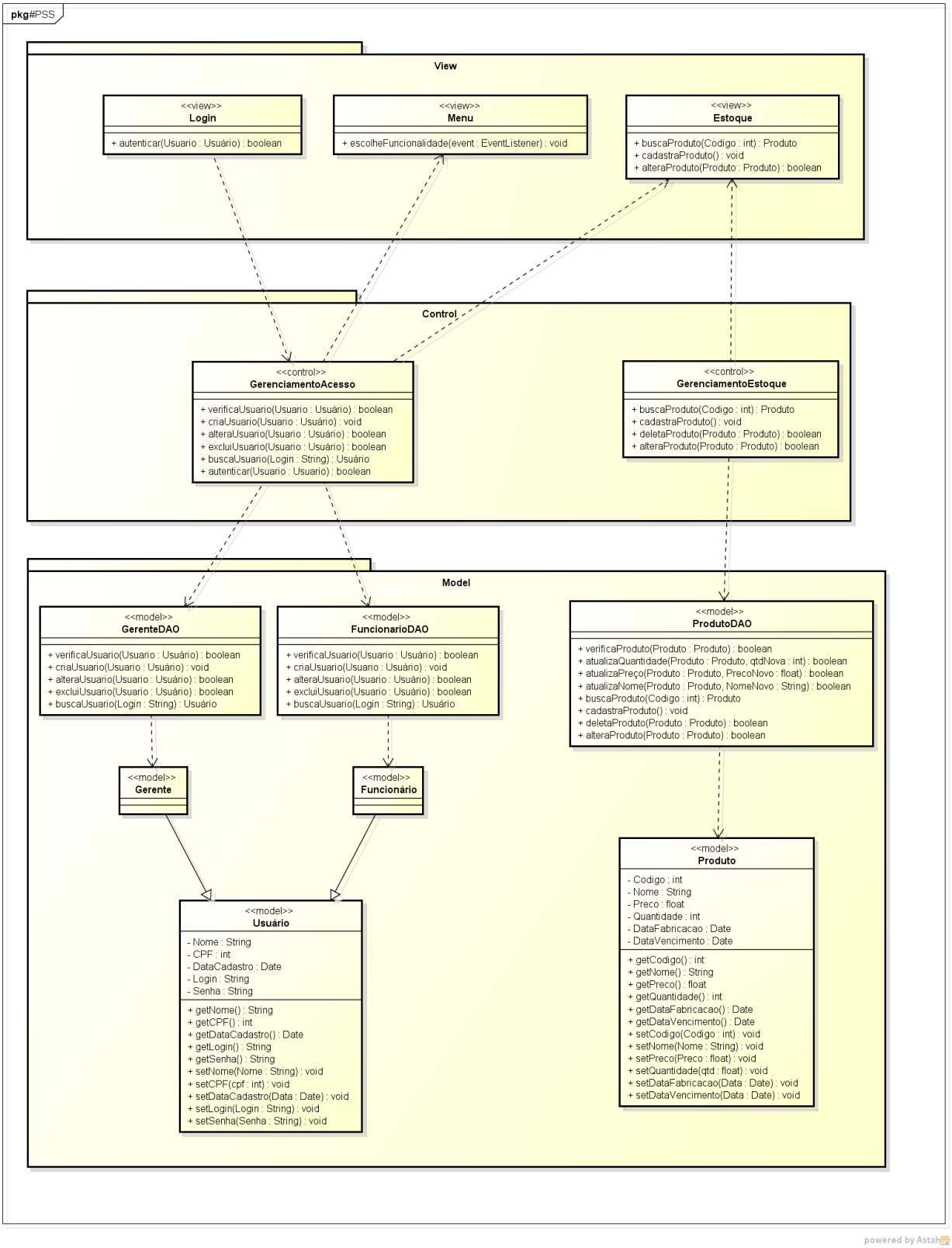

Figura 34. Classes MVC Usuário e Estoque

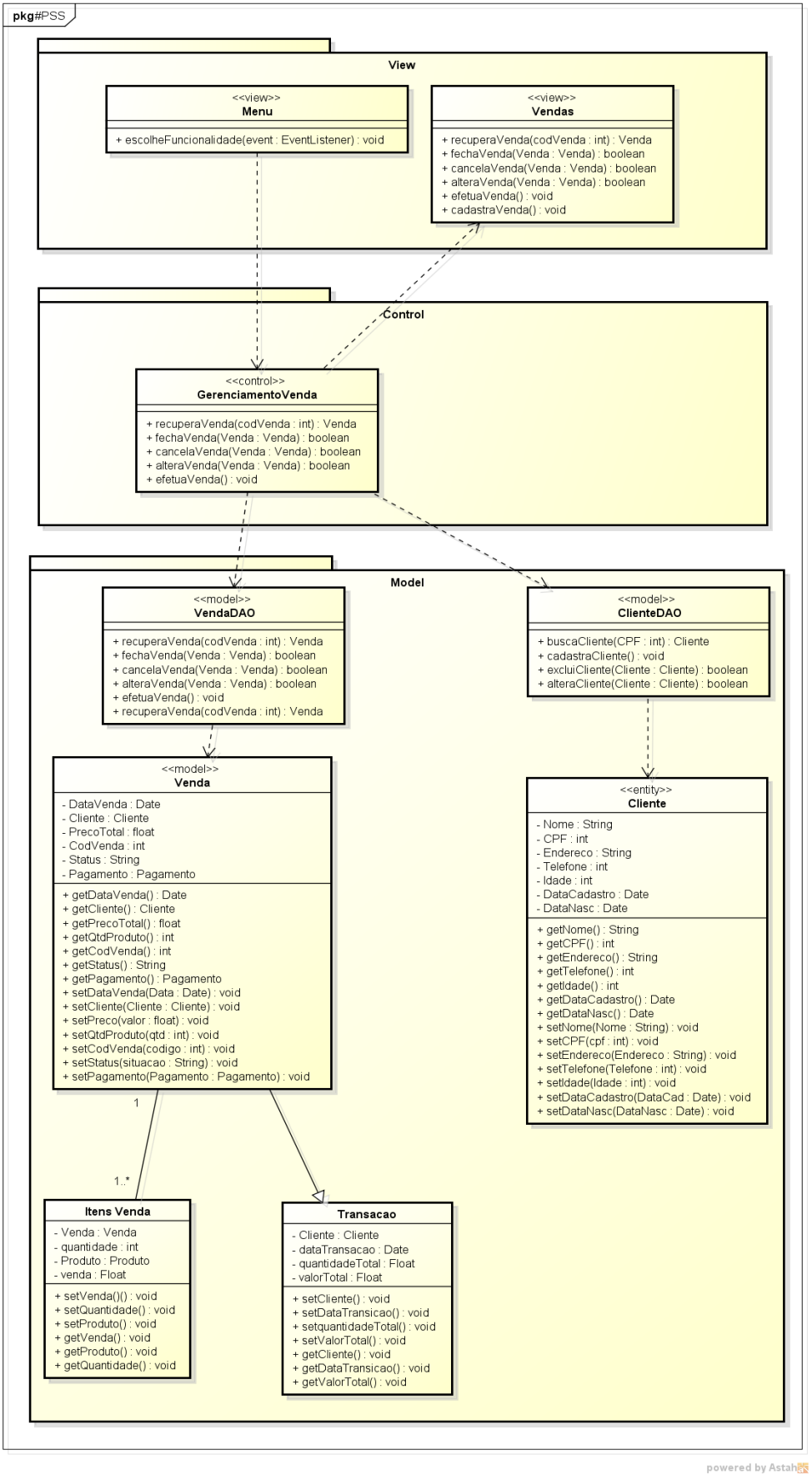

Figura 35. Classe MVC para Venda e Cliente

9 Frameworks

Hibernate: É um framework para mapeamento objeto-relacionado escrito na linguagem Java, sendo disponível também em .Net com o nome NHibernate.

O objetivo do Hibernate é facilitar o mapeamento dos atributos entre uma base de dados relacionais e o modelo objeto de uma aplicação, seja com o uso de arquivos XML, seja por anotações Java. Assim diminuir a complexidade entre os programas Java, baseado no modelo orientado a objeto, que precisam trabalhar com um banco de dados do modelo relacionado.

Foi criado por diversos desenvolvedores Java espalhados ao redor do mundo, sendo liderados por Gavin King. Posteriormente os principais desenvolvedores do programa foi contratado pela empresa JBoss Inc. para realizar o suporte do produto. O Hibernate é um software livre de código aberto distribuído com a licença LGPL.

A atual versão do Hibernate ORM (Mapeamento Objeto Relacional) é a 5.0, sendo lançada no dia 19 de agosto de 2015, ou seja, trazendo como novidades um novo bootstrap API, suporte para o Java 8, e outras novidades que podem ser encontradas no site http://hibernate.org/.

O framework utiliza o dialeto HQL (*Hibernate Query Language*), que nada mais é uma linguagem de consulta parecida com a SQL, com o diferencial de ser totalmente orientada a objeto, incluindo os paradigmas de herança, polimorfismo e encapsulamento. Caso seja de sua preferência, também utilizar a linguagem SQL.

Utilizar a linguagem HQL tem seus benefícios, tais como, executar pedidos SQL sobre as classes de persistência do Java ao invés de tabelas no banco de dados. Também, ao utilizar o HQL temos a vantagem de portabilidade de banco, assim se houver mudança no banco de dados, o HQL automaticamente criará comandos referentes ao novo banco.

A principal característica deste framework é a transformação das classes em Java para tabelas de dados. O Hibernate gera chamadas SQL e assim libera o desenvolvedor do trabalho manual da conversão dos dados resultante, e ainda mantém o programa portável para quaisquer bancos de dados SQL, apenas há um acréscimo no tempo de execução.

Para gerenciamento de transações e tecnologia de acesso à banco de dados a responsabilidade serão de outros elementos na infraestrutura do programa. Mesmo que existam API no Hibernate que possuem operações de controle transacional, o Hibernate delegará funções para a infraestrutura na qual foi instalada.

Para aplicações construídas para serem executadas em servidores de aplicação, o gerenciamento de transações seguirá o padrão JTA, enquanto em aplicações standalone, o tratamento será de responsabilidade do driver JDBC.

Abaixo, uma lista de características do Hibernate:

Mapeamento Objeto Relacional (ORM): Com o Hibernate ORM permite aos desenvolvedores a escrita mais fácil de aplicações cujos dados sobrevivem ao processo de aplicação. Com um

framework ORM, o Hibernate está preocupado com a persistência de dados que se aplica aos banco de dados relacionais via JDBC.

JPA Provider: O Hibernate também é uma implementação da especificação Java Persistence API (JPA). Com isso, ele é utilizado de maneira fácil em qualquer ambiente de apoio JPA, incluindo aplicativos Java SE, servidores de aplicação Java EE, etc.

Persistência Idiomática: É permito o desenvolvimento de classes persistentes seguindo expressões idiomáticas orientadas a objetos naturais, como herança, polimorfismo, associação, composição e o framework das coleções do Java. O Hibernate não necessita de interfaces ou classes base para classes persistentes e permite que qualquer classe ou estrutura de dados seja persistente.
Alta Performance: O Hibernate suporta a inicialização lenta, inúmeras estratégias atraentes e bloqueio otimista com versionamento automático e tempo de estampagem. Hibernate não necessita de tabelas de database especiais ou campos e gera a maior parte do SQL em tempo de inicialização do sistema, em vez de em tempo de execução.

Hibernate consistentemente oferece desempenho superior sobre o código JDBC em linha reta, tanto em termos de produtividade do desenvolvedor e desempenho de tempo de execução.

Escalabilidade: Hibernate foi projetado para trabalhar em um cluster de servidor de aplicação e oferece uma arquitetura altamente escalável.

Confiável: O Hibernate é utilizado por dezenas de milhares de desenvolvedores Java devido a sua excelente estabilidade e qualidade.

Extensibilidade: O Hibernate é altamente configurável e extensível.

Spring: É um framework open source para plataforma Java desenvolvido por Rod Johnson é descrito em seu livro "*Expert One-on-One: JEE Design e Development*". É baseado nos padrões de projeto inversão de controle (IoC) e injeção de dependência.

A inversão de dependência é o diferencial entre um framework e uma simples biblioteca. Uma biblioteca consiste em um conjunto de classes que um usuário instancia e utiliza seus métodos. Após a chamada ao método, o controle do fluxo da aplicação retorna para o usuário. Enquanto em um framework o fluxo é diferente, pois para utilizar algum framewok, o código próprio da aplicação deve ser criado e mantido acessível ao framework, pode ser via classes que estendem classes do framework, este então, realizada a chamada deste código da aplicação. Ao fim da utilização do código da aplicação, o fluxo retorna para ele.

A injeção de dependência trata-se de uma especialização da inversão de controle, que visa garantir um baixo acoplamento em uma cadeia de classes. A injeção de dependência é baseado em outro padrão de projeto, o *Builder*, que será responsável por construir os objetos e armazená-los. Ao passamos a responsabilidade para o *builder*, precisamos indicar qual a interface que desejamos, e ele criará o objeto que implemental tal interface e injeta esta dependência na classe Cliente.

No Spring, o *container* se encarrega de instanciar classes de uma aplicação Java e definir dependências entre elas através de arquivo de configuração XML, inferências do framework, e também anotações de classes, métodos e propriedades. De tal forma, o Spring permite o baixo acoplamento entre classes de uma aplicação orientada a objetos.

O Spring possui uma arquitetura baseada em interfaces e POJOs, que são objetos Java que seguem um desenho simplificado em contraposição aos EJBS. A esse último, são oferecidos mecanismos de segurança e controle de transações. Também facilita testes unitários e é uma alternativa à complexidade no uso de EJBs.

Atualmente, o Spring possui diversos módulos, tal como, o Spring Data, que trata da persistência de objetos e Spring Security, trata da segurança do aplicativo. Com as funcionalidades de injeção de dependência e programação orientada a objetos, cabe ao desenvolvedor escolher quais ferramentas que deseja utilizar do Spring, facilitando aplicações simples devido a sua não necessidade de arrastar todas as ferramentas do framework.

O Spring possui sua própria configuração XML, e assim pode ser utilizado em ambientes "não Web".

O módulo Spring Core representa as principais funcionalidades do Spring, no qual o principal elemento é o BeanFactory. Trata-se de uma implementação do padrão de projeto Factory, responsável em remover a programação de Singletons e permitindo o baixo acoplamento entre a configuração e a especificação de dependência, de sua lógica de programação.

O módulo Spring DAO provê uma camada de abstração para JDBC, eliminando grande parte da codificação necessária para interagir com o banco de dados. O módulo ORM provê integração do Spring com outros frameworks para persistência de objetos, como o Hibernate, que será utilizado neste projeto.

No Spring, qualquer objeto que forma a sua aplicação e que está sob seu controle é considerado um bean. O bean nada mais é um objeto da sua aplicação e é gerenciado pelo *Container IoC*. As dependências entre beans são definidas através de metadados.

Grails: Framework para construção de aplicação para web através da linguagem de programação Groovy. Desenvolvido para ser um framework de alta produtividade graças ao uso do paradigma de programação por convenção que tem como objetivo preservar o desenvolvedor dos detalhes de configuração. É utilizado o modelo de desenvolvimento de software MVC, adiante será explicado como é implementado o MVC pelo Grails, e conta com importantes APIs Java em sua arquitetura como o Hibernate, Spring (descritos acima e utilizados em nosso projeto), JSF e Maven. Também pode ser integrado com qualquer biblioteca Java através de plug-ins ou acesso direto.

Um grande facilitador do Grails é a utilização da programação por convenção para definir o papel das várias entidades de uma aplicação ao invés de exigir a utilização de uma série de arquivos XML. Exemplificando, classe terminadas com "Controller" é considerado um controller, assim também sendo para "Model" e "View".

A linguagem Groovy é uma linguagem orientada a objetos com tipagem dinâmica que roda na *Java Virtual Machine*. Sua sintaxe é parecida com a linguagem Java e, além disso, é possível "integrar" aplicações Java e Groovy de forma transparente.

A princípio por ser um projeto simples, o servidor será a própria máquina utilizada para executar o aplicativo, sendo assim, será acessado localmente sem a necessidade de estar conectada a Internet.

Neste projeto o Grails será implementado juntamente com os frameworks Hibernate, Spring e JasperReports, este último será descrito logo abaixo.

Framework MVC:

Nesta seção será apresentado como o framework Grails implementa o padrão de arquitetura de software, Modelo-Visão-Controlador (MVC). A figura logo abaixo apresenta como é o fluxo de solicitação no Grails MVC.

A seguir será explicado a implementação de cada componente do MVC e um simples código exemplificando como seria o cadastro do Produto

Modelo: Objeto Java que armazena os dados que pode ser utilizados pelos controladores e visões. Tecnicamente, os controladores podem criar um modelo ou apenas processá-los em operações. No Grails é fornecido um mecanismo vinculado que ajuda a referenciar seu modelo a partir dos componentes de interface do usuário, tal como, g:each.

```
1. package grails.mvc.exemplo
2.
3. class Produto {
4. int codigo;
5. int nome;
6. float preco;
7. int quantidade;
8. Date dataFabricacao;
9. Date dataVencimento;
10. }
```

Controlador: Servlet que manipula todo o pedido do front-end. Em geral, Grails servlets estendem DispatcherServlet do Spring para iniciar o ambiente Grails (SimpleGrailsController) para lidar com os pedidos. A
SimpleGrailsController delega para a classe chamada
SimpleGrailsControllerHelper que realmente manipula a solicitação.

```
1. class GerenciamentoEstoqueController {
2. ArrayList<Produto> = listaProduto = new ArrayList<Produto>();
3. //Escopo EstÃ¡tico variavel usado para especificar o escopo do controler
4. static scope = "session"
```

```
5. //EspecificaÃ§Ã£o do aÃ§Ã£o default do Controlador
6. static defaultAction = "initialize"
7.
8. def initialize(){
9. render vier:"/gerenciamentoEstoque/cadastaProduto";
10. }
11.
12. def cadastrarNovamente(){
13. render view:"/gerenciamentoEstoque/cadastaProduto";
14. }
15.
16. def cadastrar(){
17. Produto produto = new Produto()
18. produto.properties = params
19. try{
20. listaProduto.add(Produto)
21. } catch (Exception e) {
22. log.debug("Exception Occured ::", e)
23. }
24. render view:"/viewProdutos/lista", model:[produtos:listaProduto]
25. }
26. }
26
```

Visão: Páginas do servidor Groovy são responsáveis por renderizar os modelos. O Grails utilizada GroovyPagesTemplateEngine para a renderização das visões GSP como visões que suportam o conceito de bibliotecas de tags personalizadas. Pode ser acessada as as tags g diretamento de seu GSP, sem necessidade de importação. Várias tags ready-made são fornecidas para uso. A biblioteca Grails g fornece a capacidade de acessar dados vinculados e associações de classe de domínio.

```
1. <%@ page contentType="text/html;charset=UTF-8" %>
2. <html>
3. <head>
4. <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-
1"/>
5. <meta name="layout" content="main"/>
6. <title>Cadastro de Produtos</title>
7. </head>
8. <body>
9. <div class="body">
10. <g:form action="register">
11. <table>
12. <tr><td>Codigo:</td><td><input id="codigo" name="codigo" type=
"int"/></td></tr>
13. <tr><td>Nome:</td><td><input id="nome" name="nome" type="text"
/></td></tr>
14. <tr><td>Preco Unitario:</td><td><input id="preco" name="preco"
type="float"/></td></tr>
15. <tr><td>Quantidade:</td><td><input id="quantidade" name="quant
idade" type="int"/></td></tr>
16. <tr><td>Data Fabricacao:</td><td><input id="dataFabricacao" na
me="dataFabricacao" type="text"/></td></tr>
```

```
17. <tr><td>Data Vencimento:</td><td><input id="dataVencimento" na
me="dataVencimento" type="text"/></td></tr>
18. <tr><td colspan="2"><input value="Register" type="submit"/></t
d></tr>
19. </table>
20.
21. </g:form>
22. </div>
23. </body>
24. </html>
```

Para o desenvolvimento deste software será utilizado alguns padrões de projeto de forma implícita, já que alguns padrões são encapsulados por frameworks ou por linguagens de programação. Tal como, o padrão *Adapter*, também conhecido por *Translator*, que pode ser visto no framework de persistência de dados Hibernate, que converte objetos persistentes em um formato compreensível para banco de dados. Outro padrão de projeto é o *Iterator* que é parte da própria linguagem de programação para iterar sobre listas de objetos.

Todavia, o padrão de projeto *State* será utilizado de forma explícita neste software, este projeto atua em objetos com estado maleável em que determina seu comportamento. Um exemplo é em um sistema em o usuário logado é um administrador tem opção habilitada de excluir usuários, enquanto outros tipos de usuário não tem permissão para tal operação e com isso a opção de exclusão está habilitada.

Neste trabalho, o *State* atuará na classe de Pagamento, visto que tal classe pode ter os estados Pago, Vencido, a Vencer, Parcela Paga, Parcela a vencer, Parcela vencida. Dependendo do estado que o pagamento estiver, poderá ser enviada uma mensagem ao usuário, como em estados de A Vencer e Vencimento, enquanto em estados de Pago, o sistema habilitará a opção de impressão de um comprovante de pagamento. Também será utilizado o padrão *Singleton.*

Nome: *State*
Contexto: Objetos com estado maleável que operam de forma diferente devido a seu estado em questão
Justificativa: Seria complicado manejar a mudança de comportamentos e os estados do objeto Pagamento em apenas um bloco de código e por isso o uso do padrão *State* vem a calhar ao propor a solução de criar um objeto para cada estado do Pagamento.
Classe e métodos envolvidos: A classe envolvida será a classe
Pagamento e o métodos envolvidos serão **mostraAlerta(),** que será ativado quando o estado do Pagamento estiver em a Vencer e Vencido, e
habilitaImpressao(), quando o estado for Pago, o botão para impressão do Comprovante de Pagamento estará habilitado e em outros estados estará desabilitado.

Nome: *Singleton*
Contexto: Elementos que precisam se manter acessíveis em todos os momentos no sistema.
Justificativa: O uso do *Singleton* será primordial visto que certos elementos são permanentes enquanto o sistema é acessível e devem-se manter persistentes. Construir e manter um onjeto

destes elementos na inicialização do sistema e destruí-los apenas quando o sistema for fechado.
Uso no projeto: Classes de gerenciamento (*control*) também seguem o padrão *Singleton*.

O Padrão *Singleton* indica uma situação em que se necessita de uma instância única e global de uma classe. Em Java, funções estáticas podem agir como um *Singleton*, sob a limitação de não poderem utilizar membros nãoestáticos. Outra forma de implementar o padrão *Singleton* seria com instâncias preguiçosas: Inicializa-se uma instância apenas quando necessário, e a mantém persistente até que o programa seja fechado.