UNIVERSIDADE FEDERAL FLUMINENSE
PROJETO DE SOFTWARE

Autores(as) :
Daniel Xin Xin Ye, 116031012
Flávia Elias Rocha, 115031025
Lucas Alexandre Siqueira Dos Santos, 116031055
Victor Souza de Menezes, 217031142

Links GitHub:
https://github.com/DanielYe1/comercy-back
https://github.com/lucasalexandre-cc/comercy-web

DOCUMENTO DE ARQUITETURA DE SOFTWARE

Sistema PDV

Niterói
2020

SUMÁRIO

1 Introdução

Este documento demonstra uma visão geral de alto nível para representar e explicar a arquitetura do software do Sistema de PDV (ponto de venda).

1.1 Objetivo

O Documento de Arquitetura de Software visa demonstrar de forma precisa e clara, como e com quais estratégias a arquitetura do sistema consegue cumprir os requisitos funcionais e não funcionais definidos no Documento de Requisitos.

Para descrever isso de maneira clara este documento usa a estratégia *"4+1" View Model of Software Architecture* proposta por Philippe Kruchten em 1995, tentando assim, demonstrar para os diferentes stakeholders não somente as características da arquitetura mas também a lógica e pensamento por trás das decisões de arquitetura que foram tomadas na elaboração do sistema.

1.2 Escopo

Este Documento de Arquitetura de Software foi desenvolvido a partir da análise dos requisitos funcionais e não-funcionais para o Sistema de PDV proposto como trabalho final da disciplina TCC 00338 - Projeto De Software. Ele contém todos os aspectos mais relevantes e que serviram de base para a construção do modelo de arquitetura.

1.3 Referências

[Trygve Reenskaug]: Thing-Model-View-Editor – an Example from a planningsystem, 1979
http://heim.ifi.uio.no/~trygver/1979/mvc-1/1979-05-MVC.pdf

[Kruchten]: The "4+1" view model of software architecture, Philippe Kruchten, November 1995,
http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/2003/Pbk4p1.pdf

[Peter Eeles]: Layering Strategies, 1999

[UFPE-RUPVC]: Exemplo da Web do Projeto de Registro em Curso Versão 2001.03 https://www.cin.ufpe.br/~gta/rup-vc/extend.formal_resources/guidances/examples/resources/sadoc_v1.htm

[Eric Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra]: Head First Design Patterns. O'Reilly Media, 2004.

[Marco Tulio Valente]: Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade, 2020.

1.4 Visão Geral

Seção 2 : Vai apresentar como a arquitetura vai ser descrita no documento.

Seção 3 : Apresenta os objetivos a serem alcançados pela arquitetura, e alguns processos de decisões envolvendo o projeto.

Seção 4 : Breve descrição dos casos de uso, assim como diagrama dos casos de uso.

Seção 5 : Diagrama de classes de domínio, e diagrama de classes mais completo para casos de uso.

Seção 6 : Diagrama de atividades para caso de uso mais importante.

Seção 7 : Diagrama de pacotes

Seção 8 : Diagrama de *Deployment*

2 Arquitetura

Como já mencionado na seção 1.1 este documento vai fazer o uso de algumas visões para representar a arquitetura; Visão de Negócio, Visão Conceitual, Visão de Desenvolvimento e Visão de *Deployment.* Essas visões serão compostas por uma série de diagramas utilizando como modelo a UML (linguagem de modelagem unificada. Para representar essas visões serão utilizados diagramas como:

1. Diagrama de Casos de Uso;
2. Diagrama de Classes de Domínio;
3. Diagrama de Classes Participantes para pasos de uso;
4. Diagrama de Pacotes;
5. Diagrama de *Deployment;*

3 Restrições e Decisões de Arquitetura

As principais restrições do sistema que impactaram diretamente nas decisões arquiteturas foram :

1. RF1 O sistema deve ser acessado via Web
2. RF2 O sistema deve ser implementado em Java
3. O sistema deve ter uma interface gráfica que não possua acoplamento com o domínio do negócio.
4. O sistema deve ter a sua persistência implementada usando JDBC, ou algum framework
5. Procure desenvolver seu sistema prevendo possíveis extensões e mudanças.

Para atender essas restrições foi escolhido trabalhar com o padrão arquitetural MVC (model-view-controller), uma vez que usando este padrão é completamente possível e até simples de se construir um sistema onde a interface gráfica não possui acoplamento com o domínio de negócio, esse desacoplamento também ajuda a tornar a aplicação mais escalável. Além disso o MVC também ajuda a cumprir com outra das restrições, que diz que o sistema precisa ser acessado via Web (RF1), uma vez que o MVC já é um padrão consolidado e que funciona muito bem para projetos web, possui uma grande variedade de frameworks que auxiliam na criação do sistema.

Mais detalhes sobre como o MVC foi aplicado neste projeto estarão nas próximas sessões.

4 Visão de Casos de Uso

Esta seção tenta descrever exatamente como funciona o negócio, selecionando e representando os principais cenários, atores, e os processos do negócio centrais e mais significativos e que possuem uma ampla cobertura arquitetural.

Uma lista de casos de uso selecionados para esse sistema de PDV encontra-se abaixo, além de uma descrição breve sobre os casos de uso ainda nesta seção.

1. Comprar itens
2. Fazer reclamação

3. Trocar produto
4. Trocar pontos
5. Fazer login
6. Inicializar sistema
7. Alterar produto
8. Cancelar compra

Estes casos de uso são iniciados pelo cliente, caixa ou gerente e ocorre interação com um agente externo (sistema de pagamento).

O abaixo está o diagrama de casos de uso, que representa os casos de uso descritos acima.

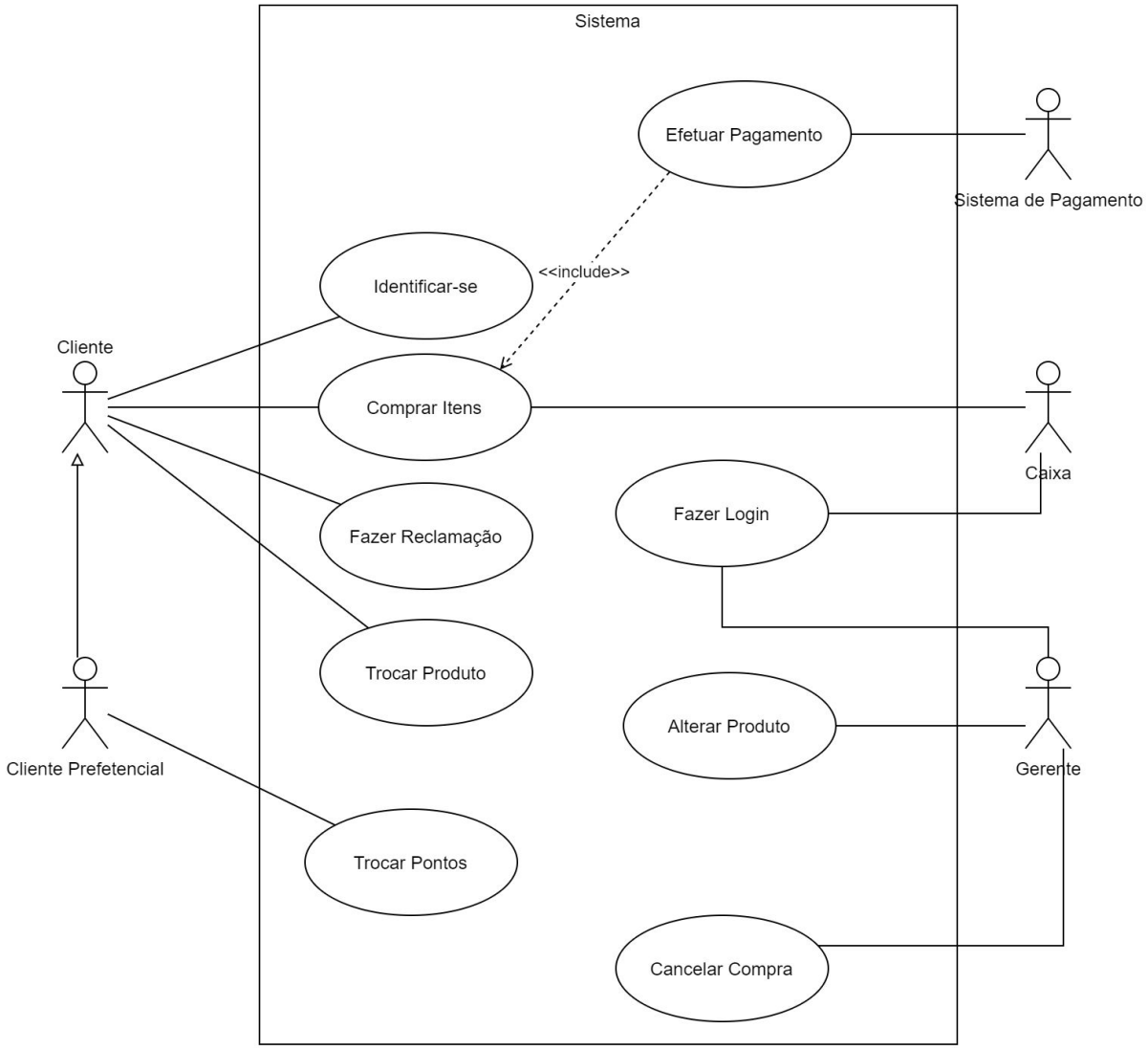


Figura 1 (Diagrama de casos de uso completo do sistema)

4.1 Comprar itens

O caso de uso de **Comprar itens** acabou sendo um dos mais críticos e mais completos do sistema, portanto este documento reservou um diagrama separado e mais completo para representar este caso de uso mais detalhadamente.

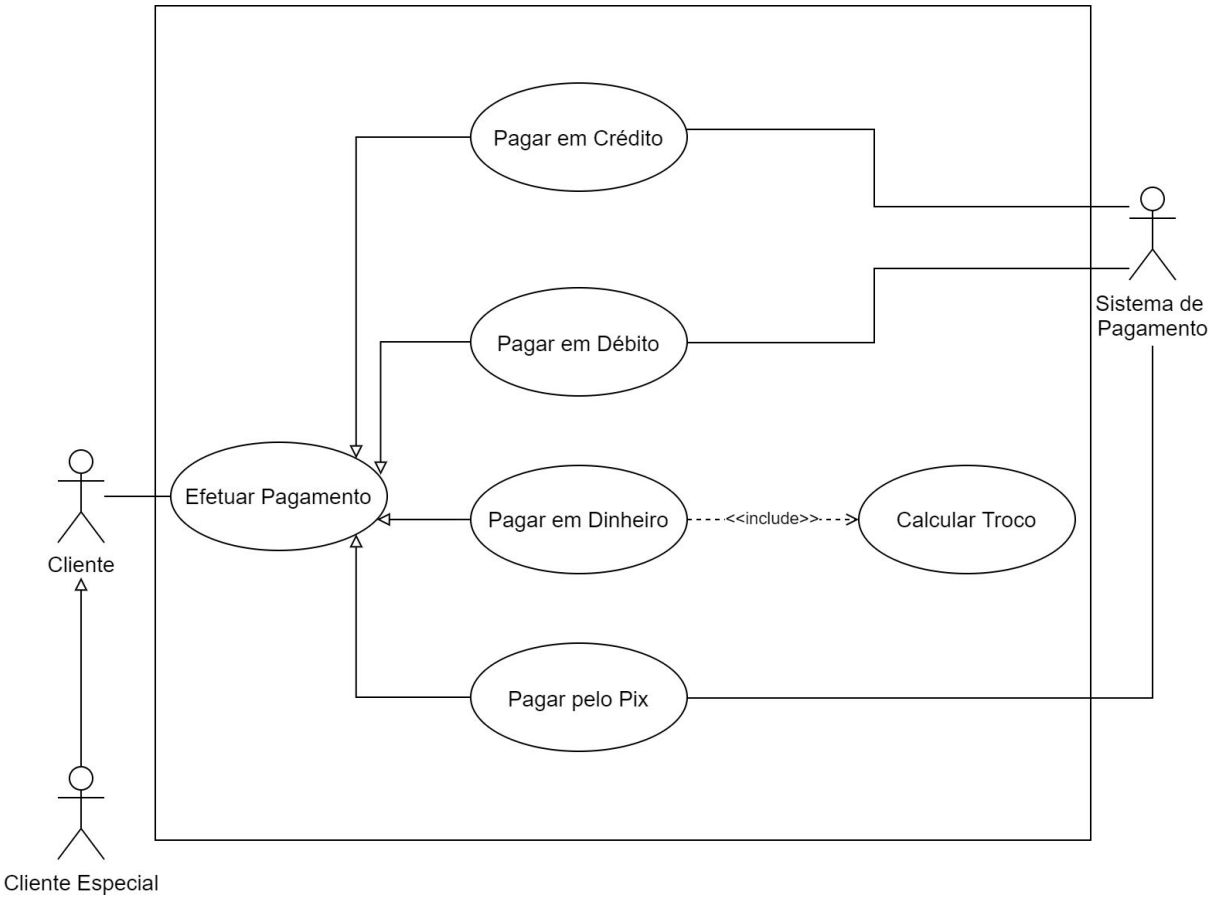


Figura 2 (Representação mais detalhada do caso de uso de Venda)

4.2 Descrição dos casos de uso

1. Comprar itens

   O Cliente vai até a loja, escolhe na loja produtos que deseja comprar e entrega para o Vendedor, além de fornecer o cpf para o vendedor. Caso o cpf não esteja no sistema, o cliente passa seu rg, seu endereço de e-mail para o vendedor fazer o seu cadastro. Caso o cliente seja preferencial, o vendedor pede que o cliente digite novamente o cpf. Caso o cliente seja preferencial e tenha pontos sobrando, ele pode utilizar esses pontos para obter um desconto na compra, que é calculado pelo sistema. O cliente faz o pagamento e o pedido da compra é registrado pelo

vendedor, que entrega uma nota fiscal gerada para o cliente, que sai da loja com os itens comprados. Caso o cliente seja preferencial, ele recebe um aviso com o seu total de pontos atualizado.

2. Fazer reclamação

   Este caso de uso ocorre quando o cliente não ficou satisfeito com o serviço. Ele deixa a reclamação na loja e de dois em dois meses os vendedores tiram um dia para cadastrar essa reclamação no sistema para o gerente ver.

3. Trocar produto

   Este caso de uso ocorre quando o cliente não ficou satisfeito com o produto e deseja trocar por outro. O cliente devolve o produto para o vendedor e entrega o novo produto que ele quer levar. Se a troca não for por defeito, o vendedor insere o código do produto devolvido e o do novo. Caso esteja com defeito ele cadastra que o produto estava com defeito e o código do novo produto. O sistema valida a troca e gera uma nova nota fiscal.

4. Trocar pontos

   Este caso de uso ocorre quando o cliente preferencial tem pontos acumulados e deseja trocá-lo numa compra por desconto. O cliente diz para o vendedor o seu cpf, diz quantos pontos deseja trocar e digita novamente o seu cpf para confirmar. Se os pontos forem suficientes para algum desconto, o sistema retorna o desconto e o novo saldo. Caso não, ele cancela a troca de pontos.

5. Fazer login

   Os usuários do sistema autorizados colocam seu login e sua senha para visualizar o que podem do sistema e assim realizar o que desejam.

6. Alterar produto

   Apenas um gerente logado pode ter acesso a parte de do sistema que gerencia os produtos, então o gerente tem a opção de alterar os dados de um produto no sistema, como nome e descrição.

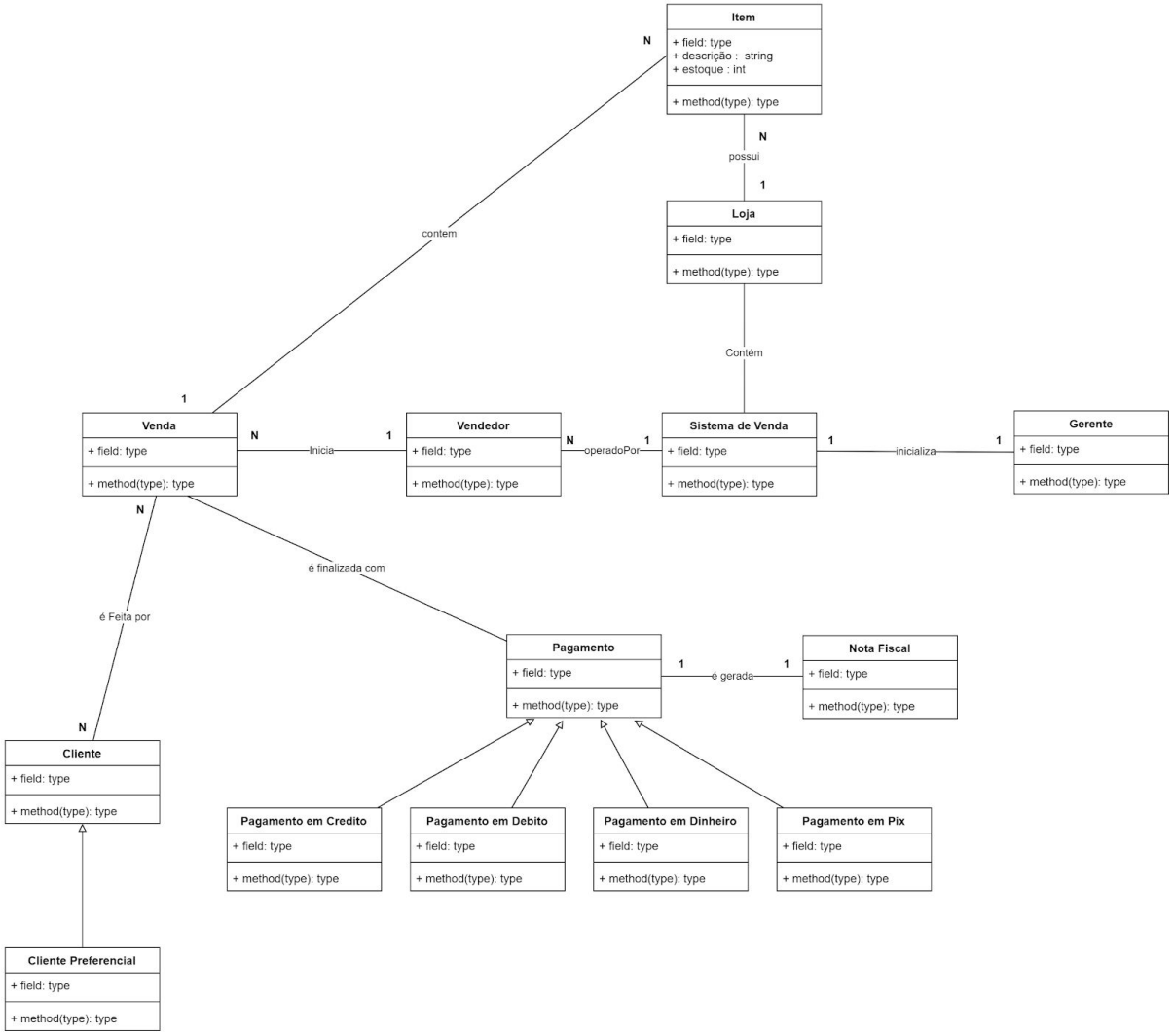
7. Cancelar compra

   O cliente diz que desistiu de comprar, então o vendedor clica no botão de cancelar compra e ela não é nem registrada.

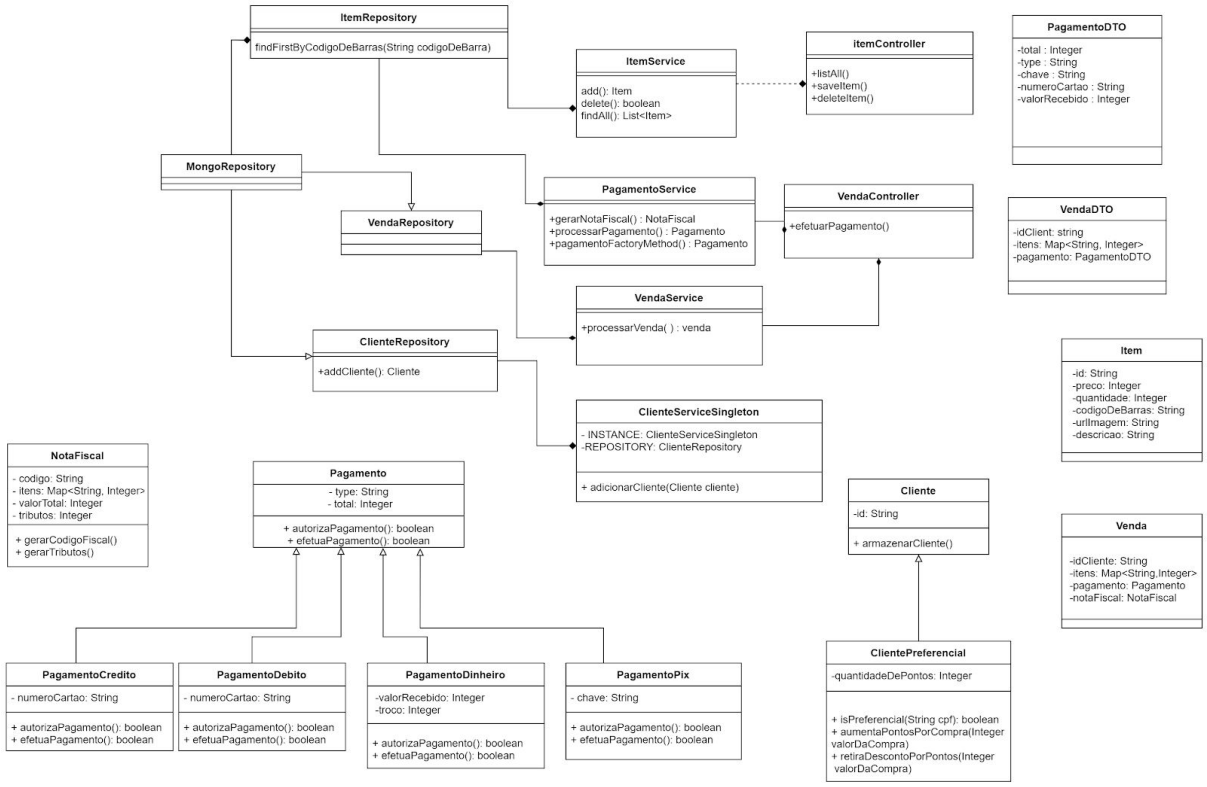
5 Visão Lógica

Para representar a visão lógica foi representado um diagrama de classes com a representação apenas das classes de domínio de negócio.

Com esse diagrama já conseguimos perceber a aplicação de alguns padrões de projeto, como por exemplo, o padrão *polymorphism* que foi aplicado nas classes de pagamento em Crédito, pagamento em Débito, pagamento em Dinheiro que herdam da superclasse pagamento.



5.1 Diagrama de Classes para Comprar Itens



Durante o processo de arquitetura do sistema foi identificado um problema em que foi necessário aplicar o padrão *Polymorphism.* Este caso ocorreu com a classe pagamento, que poderia ter mais de uma forma de lidar com pagamento (em crédito, em débito, em dinheiro e pix); Para evitar ter que usar switch-case para selecionar o comportamento da função foi usado *Polymorphism* criando assim classes separadas para cada um dos métodos de pagamento para lidar com essa diferenciação; Cada uma dessas classes vai possuir uma relação de herança com a superclasse de pagamento, que vai conter os métodos e atributos em comum entre elas.

Para a criação das diferentes instâncias de pagamento, foi utilizado o padrão *Factory Method*, onde é utilizado um método de fabricação para a criação dos objetos. Nesse método, são criados os pagamentos de acordo com os atributos do tipo, fazendo a diferenciação entre os meios de pagamento devido a suas diferentes necessidades.

Os componentes de serviço são utilizados como *Pure fabrication*, elas possuem lógicas que são importantes e que poderiam diminuir a coesão,

aumentar o acoplamento e diminuir o reuso se fossem implementadas nas aplicações do modelo.

Para a transferência de dados entre camadas de negócio, foi estabelecido o padrão de *Data Transfer Object (DTO)*. Nele, foi agrupado um conjunto de atributos em uma classe simples, como forma de facilitar a transferência de dados dentro da aplicação.

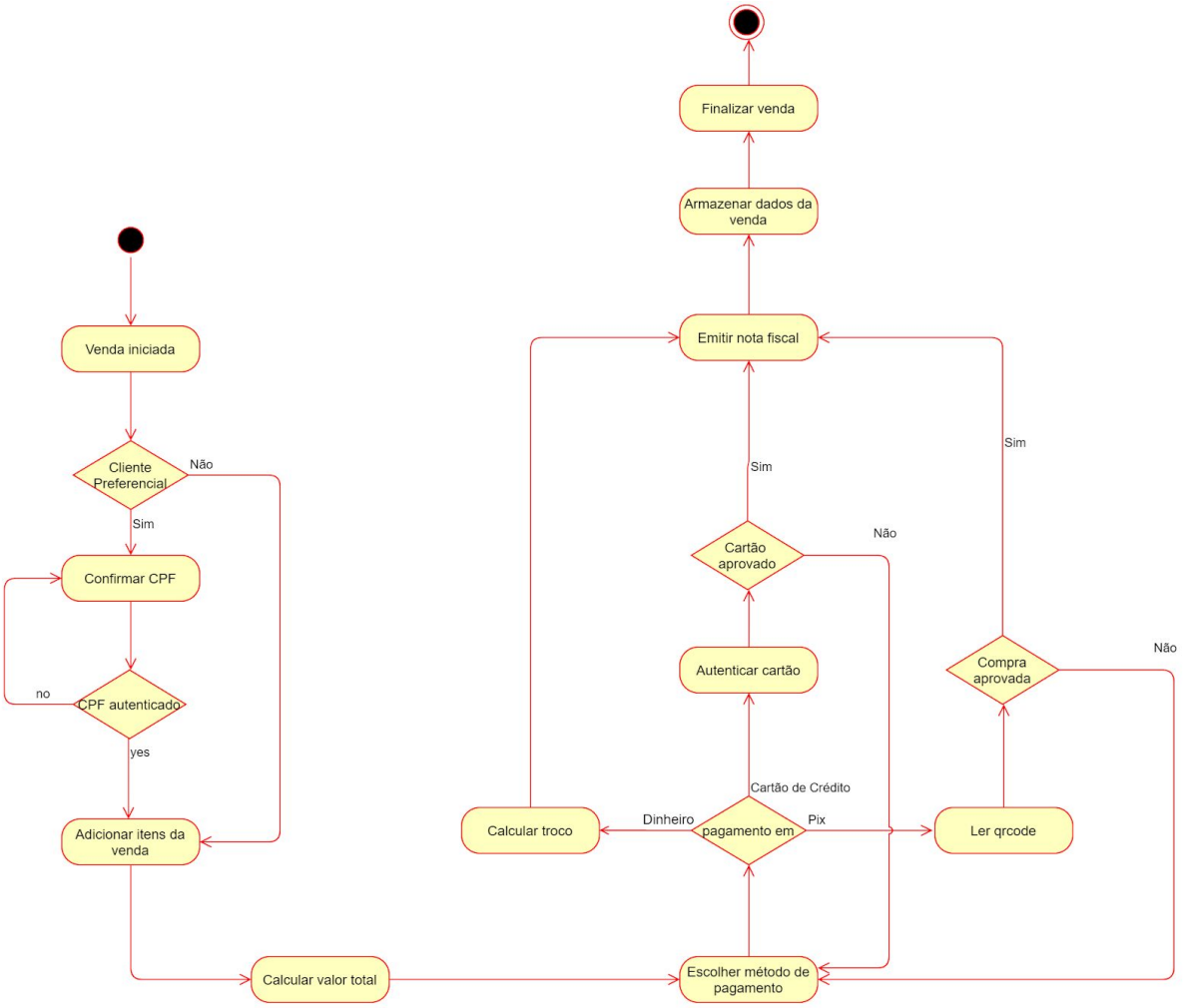
Um exemplo bem comum para a criação de *DTO's* em *API's REST* é quando existe a necessidade de mapear os dados que podem vir em um *body* do request, para transferir esses dados para classes lógicas da aplicação. Para isso, é possível criar uma classe cujos atributos representem o que se espera que venha nesse body do request. O *DTO* foi utilizado exatamente para esse caso na classe de VendaController, onde foi criado um VendaDTO para representar o que um request para "/venda" envia como informações.

O padrão Singleton foi utilizado na classe de *ClienteService*, as classes de serviço e os *controllers*, eles também são Singletons mas a implementação do padrão se dá por conta do framework Spring. No caso da classe ClienteService, foi implementada utilizando o padrão, de forma que apenas uma única instância dessa classe é utilizada em todo o projeto.

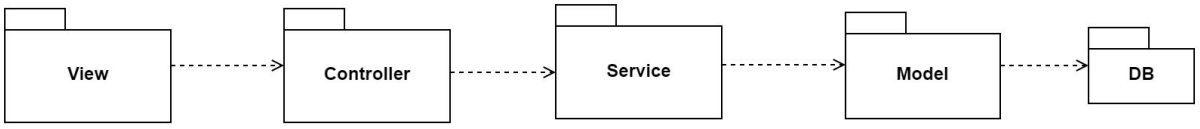
O padrão MVC está explicado na seção 7.

6 Visão de Processos

Para essa visão foi utilizada um diagrama de atividades para representar o processo de uma Venda, especificado no caso de uso [1] Comprar item



7 Visão de Implementação



View:

- Camada visual da aplicação, onde é renderizada a interface.
- Quando a View precisar de alguma informação dinâmica, irá se comunicar com a camada de Controller.

Controller:

- Ela fica responsável por formatar a requisição da View, e delegar a tarefa para a camada de Service.
- Após receber uma resposta da camada de Service, ela também é responsável por formatar o resultado e entregar para a View.

Service:

- Camada responsável pela lógica de negócio do sistema.
- Quando, para lógica de negócio, for necessário utilizar algum modelo do nosso banco de dados, ele irá chamar a camada de Model.

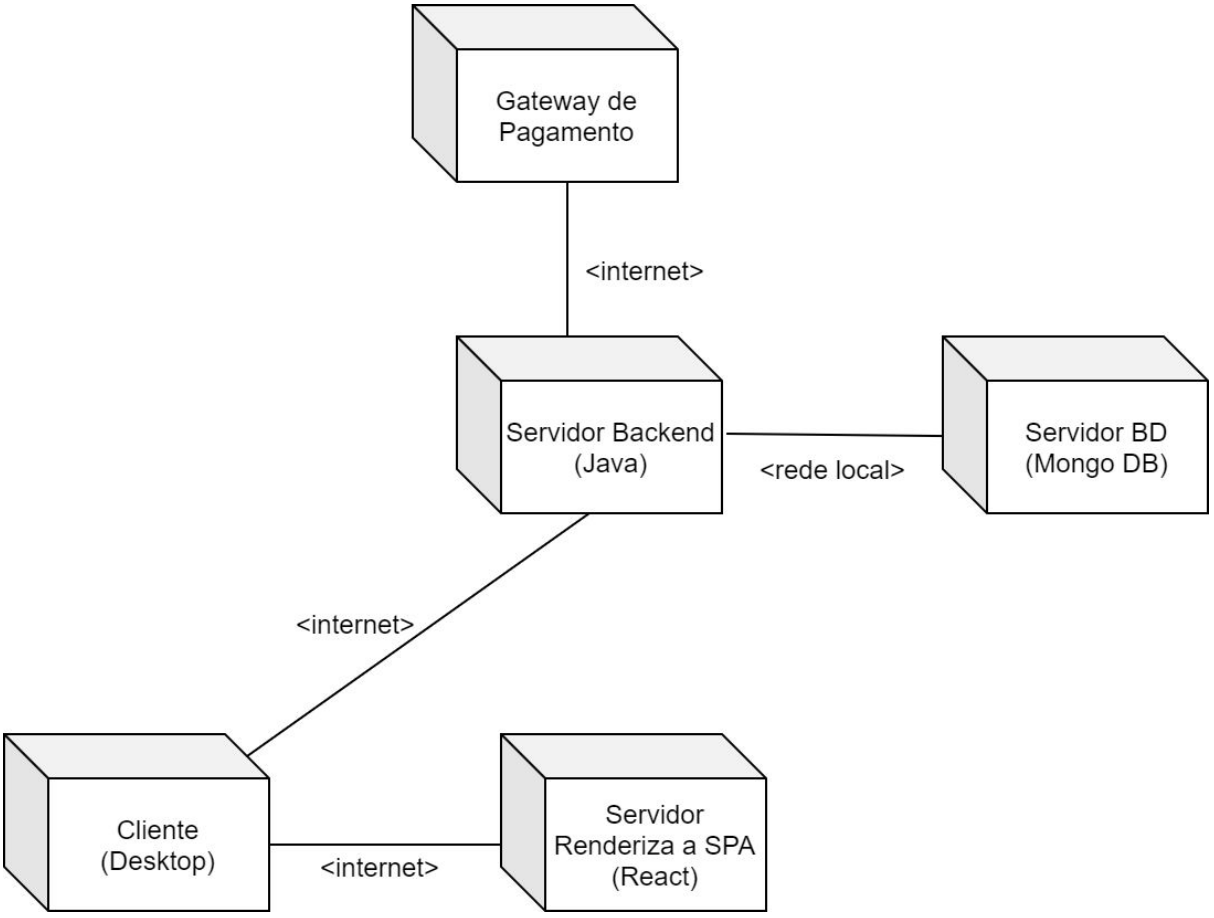
Model:

- Camada responsável pela interface com o banco de dados. Ela é responsável por representar um modelo, e oferecer abstrações da comunicação desse modelo com o banco de dados.

DB:

- Camada responsável pela persistência de dados do sistema.

8 Visão de *Deployment*



Cliente:

- Navegador do usuário.

Servidor SPA:

- Um servidor que renderiza a aplicação front-end, que no caso é uma Single Page Application (SPA) desenvolvida utilizando a linguagem javascript e o framework ReactJS.

Servidor Backend:

- Uma API REST que contém a lógica de negócio da aplicação, desenvolvida utilizando Java e o framework Spring.

Servidor BD:

- Um servidor que contém a instância do banco de dados. Para o banco, utilizamos um banco NoSql MongoDB.

Gateway de Pagamento:

- Um servidor que contém a lógica relacionada aos pagamentos por cartões e PIX.