Ponto de Venda

Documento de Arquitetura de Software

Versão <1.4>

Histórico da Revisão

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versão** | **Descrição** | **Autor** |
| 15/06/22 | 1.4 | Atualizadas decisões, restrições e justificativas | Gabriel Figueiredo e Marcio Bedran |
| 15/06/22 | 1.3 | Atualizada sessão 8.1 | Victor Verdan |
| 14/06/22 | 1.2 | Atualizadas sessões 3,4 e 6. Alterações gerais na formatação e espaçamento. | Gabriel Figueiredo |
| 12/06/22 | 1.1 | Atualizado diagrama de casos de uso | Gabriel Figueiredo |
| 18/05/22 | 1.0 | Elaboração do Documento | Jair de Lima, Thiago R. da Motta, Victor Verdan, Gabriel Figueiredo, Rodrigo Carvalho, Winne Domingues, Marcio Bedran. |

Índice Analítico

[1. Introdução 4](#_Toc106224485)

[1.1 Finalidade 4](#_Toc106224486)

[1.2 Escopo 4](#_Toc106224487)

[1.3 Definições, acrônimos e abreviações 4](#_Toc106224488)

[1.4 Visão Geral 4](#_Toc106224489)

[2. Metas e Restrições da Arquitetura 4](#_Toc106224490)

[3. Suposições e Dependências 4](#_Toc106224491)

[4. Requisitos Arquiteturalmente Significantes 5](#_Toc106224492)

[5. Decisões, restrições e justificativas 5](#_Toc106224493)

[6. Camadas da Arquitetura 5](#_Toc106224494)

[7. Visões da Arquitetura 7](#_Toc106224495)

[7.1 Visão Estrutural 7](#_Toc106224496)

[7.1.1 Visão Geral 7](#_Toc106224497)

[7.1.2 Estrutura de Pacotes Significativos 8](#_Toc106224498)

[7.1.3 Estrutura de Classes 9](#_Toc106224499)

[7.2 Casos de Uso 10](#_Toc106224500)

[7.3 Visão de Classes Participantes (VCP) 11](#_Toc106224501)

[8. Qualidade 12](#_Toc106224502)

**Documento de Arquitetura de Software**

# Introdução

Este documento tem como objetivo descrever o documento de arquitetura do projeto Ponto de Venda. Esse projeto tem como propósito a implementação de um sistema de caixa para mercado que seja de fácil adoção e customização para poder ser adotado a diferentes tipos de negócio.

## Finalidade

Este documento oferece uma visão geral arquitetural abrangente do sistema, usando diversas visões arquiteturais para representar diferentes aspectos do sistema. O objetivo deste documento é capturar e comunicar as decisões arquiteturais significativas que foram tomadas em relação ao sistema.

## Escopo

Este documento é voltado para o auxílio dos envolvidos no desenvolvimento do projeto Ponto de Venda, captando aspectos arquiteturais do sistema.

## Definições, acrônimos e abreviações

* MVC – Padrão de arquitetura de software que consiste em M – Modelo, V – Visualização e C – Controlador. O modelo é responsável pelas regras de negócio, visualização é responsável pela parte gráfica da aplicação e controladores pelo controle de dados e eventos.
* DAO – Padrão de acesso a dados Data Access Object que tem como objetivo separar a lógica de negócios da lógica de persistência de dados.
* CRUD – Conjunto de operações básicas de um banco de dados relacional. C – Create, R – Read, U – Update e D – Delete, ou traduzindo, criar, ler, atualizar e deletar, respectivamente.
* JPA – Java Persistence API
* SGBD – Sistema Gerenciador de Banco de Dados

## Visão Geral

Serão apresentadas neste documento diferentes visões arquiteturais de como o sistema deverá se comportar em diferentes processos, como deverá ser implementado, justificações pelas escolhas feitas nesse projeto junto a como elas contribuem para todos os recursos.

# Metas e Restrições da Arquitetura

Algumas das restrições de requisito e de sistema terão uma relação fundamental com a arquitetura do projeto, sendo elas:

* Sistema deverá ser multiplataforma
* Utilização do paradigma de desenvolvimento Orientado a Objetos;
* Padrão de estrutura MVC;
* Linguagem de programação *Java*;
* Framework *SpringMVC* e *Hibernate*;
* Padrão DAO para operações CRUD;
* Banco de dados *PostgreSQL*.

# Suposições e Dependências

* Sistema de pagamentos externo para uso de cartões de crédito e débito
* Banco de dados com usuário administrador inicial pré-cadastrado

# Requisitos Arquiteturalmente Significantes

**Disponibilidade:**

O percentual de tempo em que o sistema deve estar disponível para utilização, incluindo interrupções planejadas como manutenção do sistema.

**Persistência:**

Serviços para lidar com a leitura e gravação de dados armazenados.

**Segurança:**

Fornecimento de serviços para proteger o acesso a determinadas partes do sistema e restrições em operações CRUD.

**Impressão:**

Fornece facilidades para a interface com impressoras.

# Decisões, restrições e justificativas

* Utilização da linguagem Java devido à facilidade da portabilidade e implantação do sistema em diferentes ambientes graças a JVM.
* Escolha do SpringMVC pois devido ao seu grande a infraestrutura direto na aplicação, permite que os desenvolvedores se concentrem mais na parte lógica na aplicação. Além de possuir uma grande integração com frameworks de persistência de dados, como o *Hibernate*. É um dos frameworks Web mais utilizados na comunidade de Java e atuando juntamente com o SpringBoot pode permitir um início rápido do desenvolvimento, com poucas linhas de código.
* Escolha do *Hibernate* devido a sua implementação de fácil uso da JPA, reduzindo drasticamente a quantidade de linhas de código, além de sua integração simplificada com diversos SGBD’s.
* Utilização do padrão DAO para permitir a separação das regras de negócio das regras de acesso a banco de dados na camada Model. O motivo disso é para tornar as classes mais legíveis e permitir futuras alterações no banco de dados sem interferir nas regras de negócio.
* Escolha do banco de dados *PostgreSQL* definida com base na facilidade de subir uma instância do mesmo e gerenciá-la utilizando o *pgAdmin*, bem como experiências passadas da equipe de desenvolvimento envolvendo o uso do *PostgreSQL*. Além de ser um projeto open-source e com fácil integração com o *Hibernate*.

# Camadas da Arquitetura

**MVC**:   
A escolha da arquitetura MVC foi definida pela facilidade de entendimento do código quando dividido corretamente entre as três camadas, além de permitir mais facilmente a segregação do trabalho entre os membros do grupo, no que diz respeito a códigos relacionados a classes de visualização e classes de regras de negócio.

**Arquitetura Orientada a Mensagem**:   
Comunicação feita de forma assíncrona entre cliente e servidor, onde o cliente (terminais dos caixas) insere uma mensagem na fila de mensagem do servidor (terminal central) e pode continuar com o seu processamento.

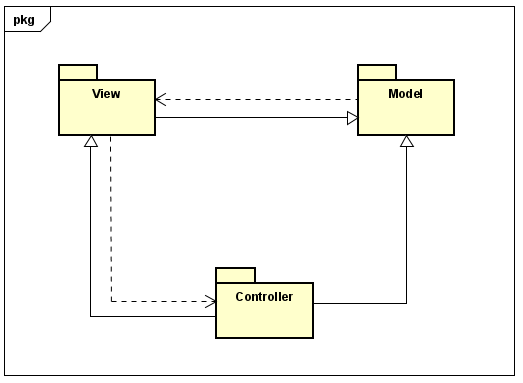
**Arquitetura Cliente/Servidor:**Centralizar o processamento das transações num servidor. O servidor consistiria de três módulos (autenticação/criação de usuários, registro de produtos/vendas, auditoria/relatórios) e o cliente seria a interface do usuário nos terminais dos caixas.

# Visões da Arquitetura

## Visão Estrutural

### Visão Geral

A visão lógica define a estrutura da arquitetura. Abaixo será especificado o padrão MVC que foi selecionado para o desenvolvimento do sistema com seus pacotes principais.



* **View:** Componente responsável pela apresentação da interface gráfica do sistema
* **Controller:** Componente responsável por receber as requisições da *View*, o que pode acarretar em uma alteração no estado do *Model* e/ou da *View*
* **Model:** Componentes que armazenam os dados manipulados pela aplicação e que têm a ver com o domínio do sistema em construção

### Diagrama, Gráfico de caixa estreita Descrição gerada automaticamenteEstrutura de Pacotes Significativos

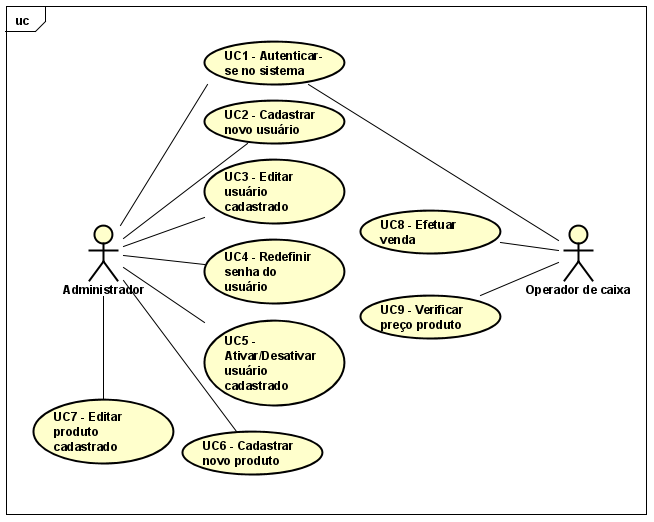
Diagrama de Pacotes do sistema

### Diagrama Descrição gerada automaticamenteEstrutura de Classes

Diagrama de Classe geral do sistema

## Casos de Uso

O seguinte diagrama descreve os casos de uso do sistema.



## Visão de Classes Participantes (VCP)

* + 1. UC1 - Autenticar-se no sistema
    2. UC2 - Cadastrar novo usuário
    3. UC3 - Editar usuário cadastrado
    4. UC4 – Redefinir senha do usuário
    5. UC5 – Ativar/Desativar usuário cadastrado
    6. UC6 – Cadastrar novo produto
    7. UC7 – Editar produto cadastrado
    8. UC8 – Efetuar venda
    9. UC9 – Verificar preço do produto

# Qualidade

O padrão de arquitetura adotado para esse projeto tem como principal objetivo garantir uma boa organização do código fonte, manutebilidade e possibilidade de extensão das funcionalidades ou modificação de elementos como a interface gráfica ou banco de dados sem drasticamente afetar o funcionamento do sistema.