Documentação final (Projeto de Software)

—

Ana Carolina Vollu - 116083067

Pedro Gomes de Sousa - 116083025

Bruno Neves Belizario - 216083120

Jean Paulo Campos - 215083127

Link do projeto no GitHub:
Front: https://github.com/anavollu/sistema-compra-client | API: https://github.com/anavollu/sistema-compra-api
Link no Heroku: https://sistema-compra.herokuapp.com/#/

Universidade Federal Fluminense

Niterói, Rio de Janeiro

1. INTRODUÇÃO

Esta documentação compila os modelos e diagramas utilizados para o desenvolvimento de um sistema eletrônico de gerenciamento de vendas para um comércio genérico.

2. ARQUITETURA E TECNOLOGIA

A arquitetura definida para a implementação do projeto é a de **microsserviços**, pois dessa forma certos grupos de módulos são executados em processos independentes, sem compartilhamento de memória. Ou seja, o sistema é decomposto em módulos não apenas em tempo de desenvolvimento, mas também em tempo de execução. Com isso, as chances de que mudanças em um módulo causem problemas em outros módulos ficam bem menores.

Para o desenvolvimento do front-end do caso de uso “Cliente preferencial faz uma compra com descontos, pagando com Pix e acumulando pontos” foi utilizado o framework **Vue.js**. Para desenvolvimento do back-end foi utilizado o **Node.js**.

3. REQUISITOS

A documentação e desenvolvimento do sistema foram baseados nos seguintes requisitos:

Requisitos funcionais (RF):

RF1: O sistema deve registrar a venda de diferentes produtos em uma transação de compra.
RF2: O sistema deve calcular o total de produtos comprados em uma transação.
RF3: O sistema deve receber o pagamento que pode ser em espécie, usando pix, cartão de crédito ou débito.
RF4: O sistema deve calcular o troco para pagamentos feitos em espécie.
RF5: Após efetuada a compra, o sistema deve gerar a nota fiscal.

RF6: O sistema deve armazenar todos os dados das compras efetuadas: CPF, identidade, endereço e e-mail.
RF8: O sistema deve ser capaz de identificar clientes preferenciais, através de seu CPF, antes de efetuarem a compra.
RF9: O sistema deve permitir que os clientes preferenciais confirmem sua identificação antes da compra
RF10: O sistema deve armazenar o valor total comprado por um cliente preferencial e converter o total em pontos que podem ser utilizados em compras futuras.
RF11: O sistema deve permitir que um cliente preferencial troque seu total de pontos por um desconto, durante uma compra.
RF12: O sistema deve manter um controle de estoque dos produtos à venda.
RF13: Os produtos devem ser mantidos em um catálogo de produtos contendo o nome, o código de identificação, numeração do código de barras e fornecedor.
RF14: O sistema deve manter um cadastro de usuários ativos do sistema que incluem os gerentes e os vendedores (caixas).
RF15: Somente usuários autorizados devem acessar o sistema.
RF16: O sistema deve ser inicializado pelo gerente.
RF17: Alterações no cadastro de produtos deve ser feito pelo gerente.
RF18: O sistema deve permitir que o gerente cancele uma compra.
RF19: Antes de iniciar suas atividades, o vendedor deve se logar no sistema.
RF20: A cada 2 meses o sistema deve enviar informações de promoções aos clientes preferenciais.
RF21: O sistema deve registrar trocas de produtos feitas por clientes preferenciais ou não
RF22: O sistema deve registrar reclamações feitas pelos clientes quer sejam preferenciais ou não em uma data específica.

Requisitos não funcionais (RNF):

RNF1: O sistema deve ser acessado via Web
RNF2: O sistema deve ser implementado em Java

4. CASOS DE USO - UC (“Use case”)

Segue abaixo o detalhamentos dos casos de uso:

UC1 - Inicializar sistema

Objetivo:	Inicializar o sistema para uso dos demais usuários.
Requisitos:	RF15, RF16
Atores:	Gerente
Pré-condições:	Gerente deve ter um usuário ativo no sistema e com autorização de acesso.
Condição de entrada:	O gerente abre o sistema.
Fluxo principal:	1. Gerente abre o sistema para ser inicializado. 2. Sistema apresenta a tela de login para o gerente. 3. Gerente informa suas credenciais de acesso. 4. Sistema verifica se as credenciais informadas são referentes à um usuário do tipo “Gerente” ativo e com permissão de acesso ao sistema. 5. Gerente inicializa o sistema para uso dos demais usuários.

UC2 - Logar no sistema

Objetivo:	Validar o acesso de um usuário ao sistema.
Requisitos:	RF14, RF15, RF19
Atores:	Gerente e Vendedor
Pré-condições:	O sistema deve ter sido inicializado pelo gerente.
Condição de entrada:	O sistema foi inicializado.
Fluxo principal:	1. Sistema apresenta uma tela de login para o usuário. 2. Usuário informa suas credenciais de acesso. 3. Sistema verifica se as credenciais são referentes à um usuário ativo e com autorização de acesso. 4. Sistema habilita o acesso ao sistema ao usuário.

UC3 - Incluir produto no catálogo

Objetivo:	Incluir um novo produto no catálogo de produtos do sistema.
Requisitos:	RF13, RF17
Atores:	Gerente
Pré-condições:	O usuário logado deve ser do tipo “Gerente”.
Condição de entrada:	O gerente recebe os dados de um novo produto a ser cadastrado no catálogo do sistema.
Fluxo principal:	1. Gerente realiza o login no sistema. 2. Gerente seleciona a opção de cadastro de produtos no catálogo. 3. Sistema apresenta a tela de cadastro de produtos. 4. Gerente seleciona a opção de inclusão de um produto. 5. Sistema apresenta a tela para inclusão de um produto. 6. Gerente informa os dados do novo produto: Nome, código de identificação, numeração do código de barras e fornecedor. 7. Gerente clica no botão para salvar o produto. 8. Sistema verifica que o produto é novo e não possui cadastro no catálogo. 9. Sistema realiza a inclusão do registro do novo produto no catálogo.

UC4 - Alterar produto do catálogo

Objetivo:	Alterar os dados de um produto do catálogo de produtos do sistema.
Requisitos:	RF13, RF17
Atores:	Gerente
Pré-condições:	O usuário logado deve ser do tipo “Gerente”.
Condição de entrada:	O gerente recebe uma solicitação para alteração do cadastro de um produto do catálogo.

Fluxo principal:	1. Gerente realiza o login no sistema. 2. Gerente seleciona a opção de cadastro de produtos no catálogo. 3. Sistema apresenta a tela de cadastro de produtos. 4. Gerente realiza a pesquisa pelo produto no catálogo. 5. Sistema retorna os produtos filtrados. 6. Gerente seleciona o produto. 7. Sistema apresenta a tela de cadastro do produto. 8. Gerente seleciona a opção de alteração de cadastro do produto. 9. Sistema apresenta a tela de edição do cadastro do produto. 10. Gerente informa os novos dados do cadastro do produto. 11. Gerente clica na opção para alterar o produto. 12. Sistema realiza a atualização do registro do produto no catálogo.

UC5 - Remover produto do catálogo

Objetivo:	Excluir um produto do catálogo de produtos do sistema.
Requisitos:	RF13, RF17
Atores:	Gerente
Pré-condições:	O usuário logado deve ser do tipo “Gerente”.
Condição de entrada:	O gerente recebe uma solicitação para exclusão de um produto do catálogo.
Fluxo principal:	1. Gerente realiza o login no sistema. 2. Gerente seleciona a opção de cadastro de produtos no catálogo. 3. Sistema apresenta a tela de cadastro de produtos. 4. Gerente realiza a pesquisa pelo produto no catálogo. 5. Sistema retorna os produtos filtrados. 6. Gerente seleciona o produto. 7. Sistema apresenta a tela de cadastro do produto. 8. Gerente seleciona a opção de exclusão do

	produto. 9. Sistema apresenta a tela de confirmação da exclusão. 10. Gerente confirma a exclusão do produto. 11. Sistema realiza a exclusão do registro do produto no catálogo.

UC6 - Informar dados do cliente

Objetivo:	Inserir os dados do cliente que está realizando uma compra.
Requisitos:	RF6, RF8
Atores:	Vendedor
Pré-condições:	--
Condição de entrada:	O vendedor finaliza o registro de uma venda.
Fluxo principal:	1. Vendedor finaliza o registro de uma venda. 2. O sistema exibe uma tela para informar o CPF do cliente. 3. O vendedor informa o CPF do cliente. 4. O vendedor seleciona a opção para confirmar.

UC7 - Cadastrar novo cliente

Objetivo:	Cadastrar um novo cliente no sistema.
Requisitos:	RF6
Atores:	Vendedor
Pré-condições:	--
Condição de entrada:	O sistema verifica que o CPF informado pelo vendedor não consta no cadastro de clientes.
Fluxo principal:	1. Sistema exibe tela para inclusão de um novo cliente. 2. Vendedor informa os dados do novo cliente: CPF, identidade, endereço e e-mail.

	3. Vendedor seleciona a opção para salvar o novo cliente. 4. Sistema realiza a inclusão do novo cliente no cadastro.

UC8 - Informar CPF de cliente preferencial

Objetivo:	Informar o CPF de um cliente preferencial que está realizando uma compra.
Requisitos:	RF6, RF8
Atores:	Vendedor
Pré-condições:	--
Condição de entrada:	O vendedor finaliza o registro de uma venda.
Fluxo principal:	1. Vendedor informa no sistema o CPF do cliente. 2. Sistema verifica que o cliente é preferencial. 3. Sistema exibe confirmação da identidade do cliente. 4. Vendedor solicita a confirmação da identidade do cliente. 5. Vendedor confirma na tela a identidade do cliente.

UC9 - Registrar venda

Objetivo:	Realiza o registro de uma nova venda no carrinho.
Requisitos:	RF1, RF2
Atores:	Vendedor
Pré-condições:	O vendedor deve estar logado no sistema.
Condição de entrada:	Vendedor recebe um cliente para uma nova venda.
Fluxo principal:	1. Vendedor inicializa a venda. 2. Sistema apresenta tela para seleção dos produtos. 3. Vendedor seleciona os produtos.

	4. Vendedor seleciona a quantidade de itens por produto. 5. Sistema inclui os produtos no carrinho. 6. Sistema contabiliza o valor total da venda. 7. Vendedor clica para finalizar a venda.

UC10 - Registrar venda com desconto

Objetivo:	Realiza o registro de uma nova venda, para um cliente preferencial, com desconto.
Requisitos:	RF1, RF2, RF8, RF11
Atores:	Vendedor
Pré-condições:	O cliente deve ser do tipo preferencial e possuir pontos.
Condição de entrada:	O vendedor registra os produtos no carrinho referente a venda para um cliente preferencial.
Fluxo principal:	1. Vendedor clica para finalizar uma venda e informa o CPF do cliente. 2. Sistema verifica que o cliente é do tipo “Preferencial”. 3. Sistema verifica que o cliente possui pontos a serem trocados por desconto na compra. 4. Sistema aplica um desconto de um real, no valor da venda, para cada ponto do cliente.

UC11 - Selecionar método de pagamento

Objetivo:	Selecionar o método de pagamento optado pelo cliente.
Requisitos:	RF1, RF2, RF3
Atores:	Vendedor
Pré-condições:	A venda deve ter sido registrada pelo vendedor.
Condição de entrada:	O vendedor finaliza o registro da venda.
Fluxo principal:	1. Sistema exibe a tela de checkout da venda.

	2. Vendedor verifica com o cliente qual o método de pagamento. 3. Vendedor seleciona o método de pagamento no sistema: Dinheiro, Cartão ou PIX.

UC12 - Receber pagamento no dinheiro

Objetivo:	Realiza o recebimento do pagamento da venda em espécie.
Requisitos:	RF3, RF4
Atores:	Vendedor
Pré-condições:	O vendedor deve ter selecionado o método de pagamento “Dinheiro”
Condição de entrada:	Vendedor seleciona o método de pagamento “Dinheiro”
Fluxo principal:	1. Vendedor informa ao cliente o valor total da compra. 2. Vendedor informa no sistema o valor pago pelo cliente. 3. Sistema calcula o troco que será retornado ao cliente. 4. Vendedor entrega o troco ao cliente.

UC13 - Receber pagamento no cartão

Objetivo:	Realiza o recebimento do pagamento da venda no cartão.
Requisitos:	RF3
Atores:	Vendedor
Pré-condições:	O vendedor deve ter selecionado o método de pagamento “Cartão”
Condição de entrada:	Vendedor seleciona o método de pagamento “Cartão”
Fluxo principal:	1. Vendedor informa ao cliente o valor total da

	compra. 2. Vendedor solicita ao cliente que insira o cartão na máquina. 3. Vendedor solicita que o cliente insira a senha. 4. Sistema verifica o processamento do pagamento junto ao banco do cliente. 5. Sistema valida o processamento do pagamento.

UC14 - Receber pagamento no Pix

Objetivo:	Realiza o recebimento do pagamento da venda via PIX.
Requisitos:	RF3
Atores:	Vendedor
Pré-condições:	O vendedor deve ter selecionado o método de pagamento “Pix”
Condição de entrada:	Vendedor seleciona o método de pagamento “Pix”
Fluxo principal:	1. Vendedor informa ao cliente o valor total da compra. 2. Sistema gera o QR code para pagamento via Pix. 3. Vendedor exibe o QR para o cliente. 4. Sistema verifica o processamento do pagamento junto ao banco do cliente. 5. Sistema valida o processamento do pagamento. 6. Sistema envia uma notificação para o cliente.

UC15 - Trocar produto

Objetivo:	Realizar a troca de um produto solicitada por um cliente.
Requisitos:	RF19, RF21
Atores:	Vendedor
Pré-condições:	O vendedor deve estar logado no sistema.
Condição de entrada:	Vendedor recebe um cliente para uma troca de produtos.

Fluxo principal:	1. Vendedor seleciona a opção de “Troca de produtos” no sistema. 2. Sistema exibe a tela de troca de produtos. 3. Vendedor informa o código de barras do produto retornado pelo cliente. 4. Sistema exibe opções de troca: “Trocar por produto de mesmo valor” ou “Trocar produto por pontos”.

UC16 - Trocar um produto por outro

Objetivo:	Realizar a troca de um produto por outro.
Requisitos:	RF19, RF21
Atores:	Vendedor
Pré-condições:	O vendedor deve ter inicializado o registro da troca.
Condição de entrada:	O vendedor deve ter selecionado a opção de “Trocar por produto de mesmo valor”
Fluxo principal:	1. Vendedor seleciona a opção “Trocar por produto de mesmo valor” 2. Sistema retorna uma lista de produtos de mesmo valor. 3. Vendedor verifica com o cliente por qual produto será trocado. 4. Vendedor seleciona o produto do catálogo que será trocado. 5. Sistema atualiza o estoque de produtos retornando o produto trocado pelo cliente e removendo um item do produto levado pelo cliente.

UC17 - Trocar um produto por pontos

Objetivo:	Realizar a troca de um produto por pontos para um cliente preferencial.
Requisitos:	RF19, RF21

Atores:	Vendedor
Pré-condições:	O vendedor deve ter inicializado o registro da troca.
Condição de entrada:	O vendedor deve ter selecionado a opção de “Trocar produto por pontos”
Fluxo principal:	1. Vendedor seleciona a opção “Trocar produto por pontos” 2. Sistema calcula a quantidade de pontos do produto com base no preço do produto (1 ponto para cada R$ 10,00). 3. Sistema atualiza os pontos do cliente preferencial.

UC18 - Cancelar compra

Objetivo:	Realizar o cancelamento de uma compra.
Requisitos:	RF12, RF18
Atores:	Gerente
Pré-condições:	O usuário deve ser do tipo “Gerente”
Condição de entrada:	O vendedor recebe a solicitação para cancelar uma compra.
Fluxo principal:	1. Gerente seleciona a opção de cancelamento da venda. 2. Sistema exibe a tela de login. 3. Gerente informa as credenciais de acesso. 4. Sistema valida o acesso do usuário. 5. Sistema realiza o cancelamento da venda. 6. Sistema atualiza o estoque de produtos.

UC19 - Efetivar venda

Objetivo:	Realizar a efetivação da venda no checkout.
Requisitos:	RF1, RF2, RF5, RF10, RF12
Atores:	Vendedor

Pré-condições:	O vendedor deve ter recebido o pagamento por parte do cliente.
Condição de entrada:	O vendedor finaliza a venda no sistema.
Fluxo principal:	1. Vendedor finaliza o registro da venda. 2. Sistema emite a nota fiscal da venda. 3. Sistema imprime a nota fiscal emitida. 4. Vendedor entrega a nota fiscal para o cliente. 5. Sistema atualiza o estoque de produtos. 6. Sistema atualiza os pontos do cliente com base no valor total da venda.

5. DIAGRAMAS

Diagrama de classes

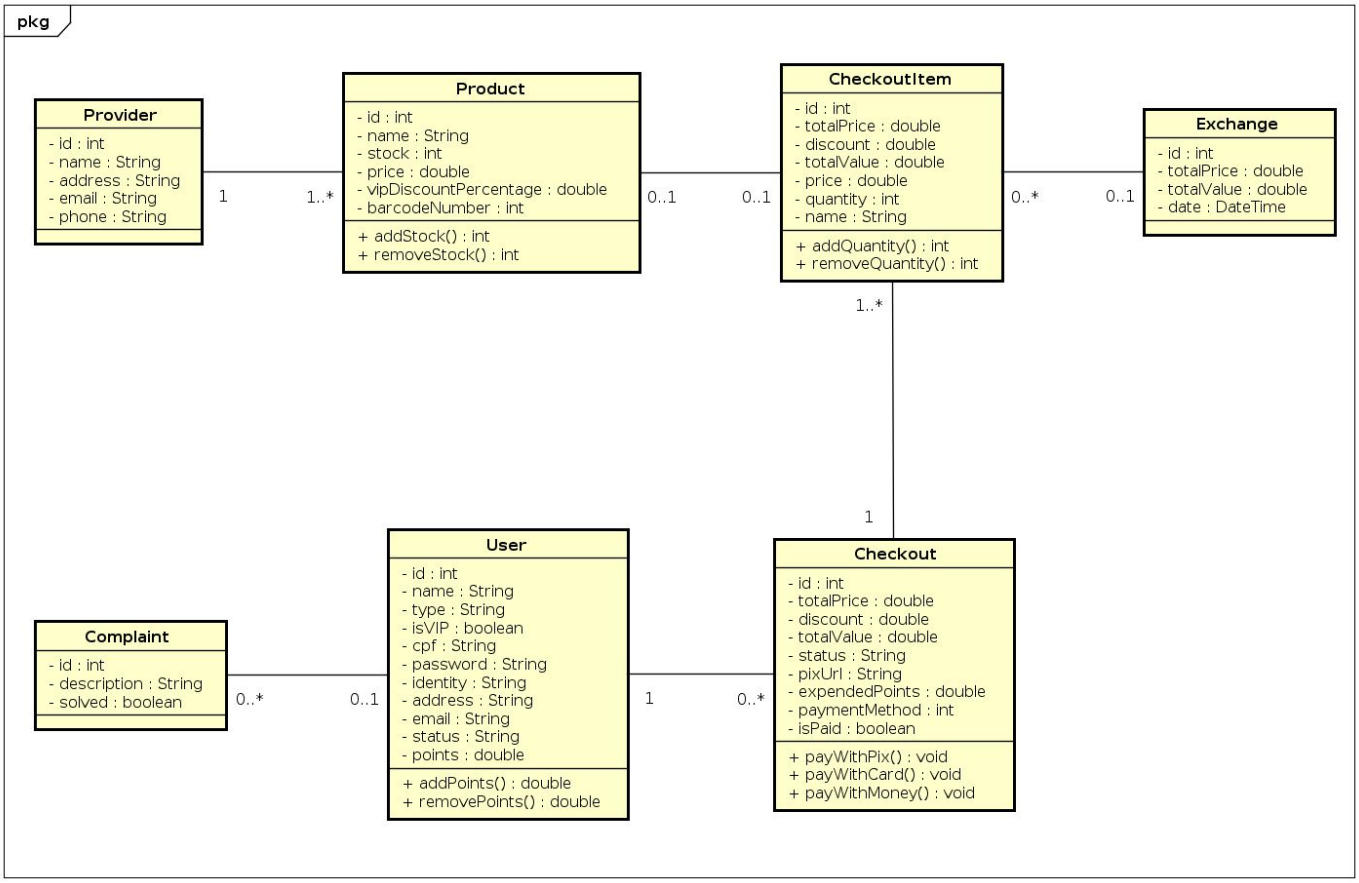


Diagrama de casos de uso

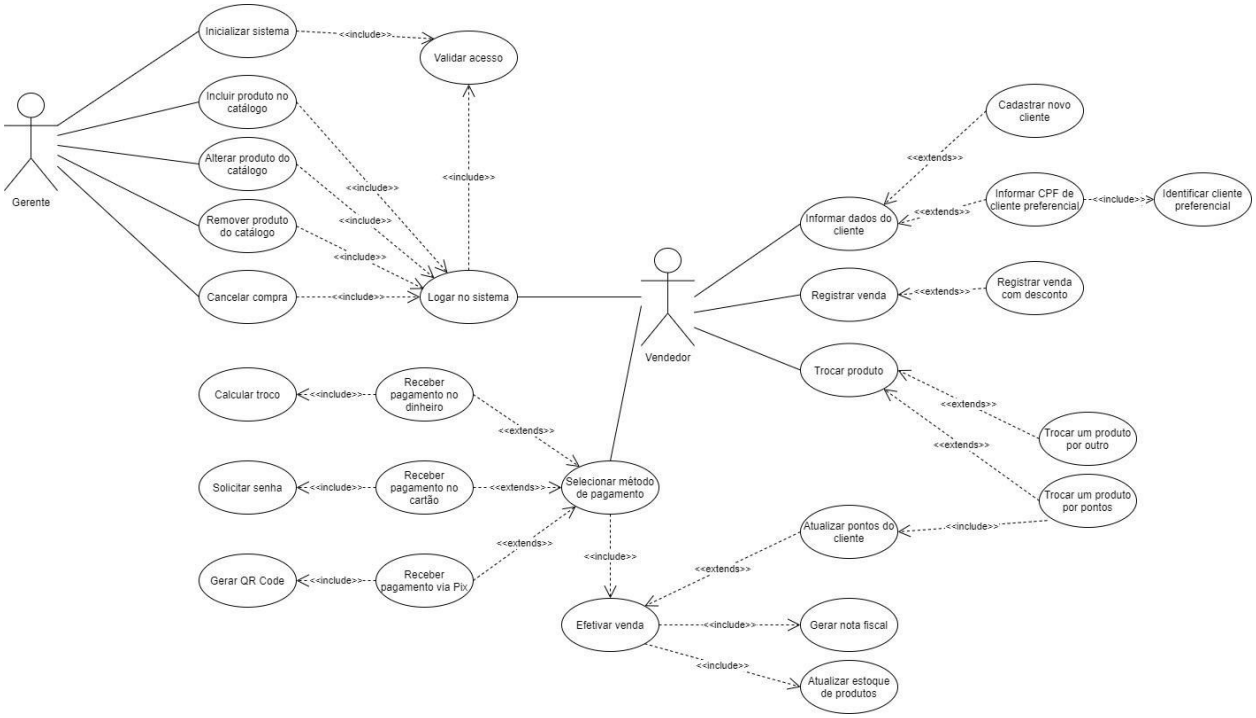
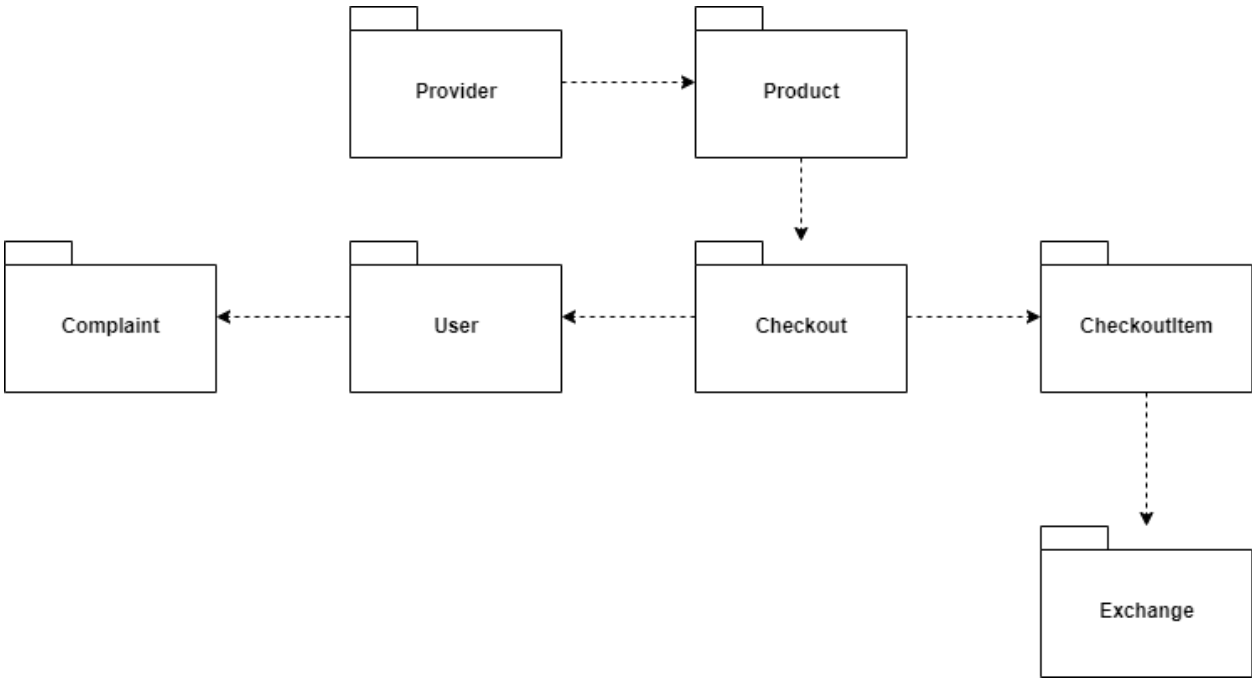


Diagrama de pacotes



6. PADRÕES DE PROJETO

Singleton

Utilize o padrão Singleton quando uma classe em seu programa deve ter apenas uma instância disponível para todos seus clientes; por exemplo, um objeto de base de dados único compartilhado por diferentes partes do programa.

Utilizado para api com a instance axios create, pois é criada uma instância de criação do axios para acessar a api, assim, todo mundo que precisa acessar a api, usa a mesma instância, no caso abaixo instance.get.

```
const instance = axios.create({
  baseURL: process.env.VUE_APP_API_URL,
  timeout: 1000,
});
```

```
        get(...args) {
          console.log('making a request...');
          return instance.get(...args);
        },
```

Chain of responsability

O Chain of Responsibility é um padrão de projeto comportamental que permite que você passe pedidos por uma corrente de handlers. Ao receber um pedido, cada handler decide se processa o pedido ou o passa adiante para o próximo handler na corrente.

auth.js > middleware que é utilizado para chamar o próximo handler (em next). Ele recebe uma requisição (req), resposta (res) e próximo (next), executa alguma lógica e chama o próximo pelo next.

```
    async function auth(req, res, next, roles = []) {
     if (req.headers.authorization) {
       const token = req.headers.authorization.split(' ')[1];
       // only let requests with allowed role go to next middleware
       if (roles.includes(token)) return next();
       const noAuthError = new Error('your role is not authorized');
       noAuthError.status = 403;
       return next(noAuthError);
     }
     const noAuthError = new Error('no authorization header');
```

```
      noAuthError.status = 401;
      return next(noAuthError);
    }

    function generateAuth(roles) {
      return (req, res, next) => auth(req, res, next, roles);
    }

    module.exports = (...roles) => generateAuth(roles);
```

checkout.js > usa o middleware auth.js

```
    const auth = require('../middlewares/auth');
    router.get('/:id', auth('cashier', 'manager'), controller.getById);
```

Mixin pattern

Encapsular e reutilizar código expressivo nos permite escrever menos código que é mais legível e nos impede de poluir o namespace global definindo variáveis no contexto global.

money.js > Mixin com função que é utilizada para formatar o dinheiro exibido no carrinho e no checkout.

```
    function moneyFormatter(raw) {
      if (!raw) return '-';
        return raw.toLocaleString('pt-BR', { minimumFractionDigits: 2,
style: 'currency', currency: 'BRL' });
    }

    export default {
      methods: {
        moneyFormatter,
      },
      filters: {
        moneyFormatter,
      },
    };
```

Checkout.vue > uso do mixin em moneyFormatter.

```
      <div class="h5">Resumo</div>
      <div>Preço      total:      {{      loadedCheckout.totalPrice      |
moneyFormatter}}</div>
      <div>Desconto: {{ loadedCheckout.discount | moneyFormatter}}</div>
      <div  class="h4  pt-2">Valor  final:  {{  loadedCheckout.totalValue  |
moneyFormatter}}</div>

      import MoneyMixin from '../mixins/money';
      mixins: [MoneyMixin],
```

Factory

O Factory Method é um padrão criacional de projeto que fornece uma interface para criar objetos em uma superclasse, mas permite que as subclasses alterem o tipo de objetos que serão criados.

api.js > usando o factory para fazer chamada api.

```
      import factory from './apiFactory';

      const api = factory.create();

      export default {
       getProducts({ name }) {
         return api.get('/products', {
           params: {
             name,
           },
           headers: {
             authorization: 'bearer cashier',
           },
         });
       },
```

apiFactory.js > é chamada no factory.create acima, fazendo com que não sejam necessárias criar várias chamadas new para a api e sim chamar o método factory.

```
      export default {
       create() {
         const instance = axios.create({
```

```
      baseURL: process.env.VUE_APP_API_URL,
      timeout: 1000,
    });
    return {
      get(...args) {
        console.log('making a request...');
        return instance.get(...args);
      },
      post(...args) {
        console.log('making a request...');
        return instance.post(...args);
      },
    };
  },
};
```

Proxy

O Proxy é um padrão de projeto estrutural que permite que você forneça um substituto ou um espaço reservado para outro objeto. Um proxy controla o acesso ao objeto original, permitindo que você faça algo ou antes ou depois do pedido chegar ao objeto original.

axiosProxy.js: usado para criar uma instância axios para controle de acesso. No caso abaixo, foi usado pelo console.log para informar quando a instância está sendo criada, antes de realmente criar a instância.

```
import axios from 'axios';

const { create } = axios;
axios.create = function myCreate(...args) {
 console.log('creating axios instance...');
 return create(...args);
};

export default axios;
```

Polling

Use polling se houver dados que mudam com frequência ou se precisarmos esperar

que o servidor faça a transição de um determinado estado.

Checkout.vue: uso do polling para ficar perguntando quando o pagamento teve o status alterado para seguir com a finalização da compra. O pagamento é realizado pelo cliente para alterar o status de pendente para pago, por exemplo.

```
                  <b-tab title="Pix" active>
                                <qrcode-vue  :value="qrvalue"  :size="300"
level="H"></qrcode-vue>
                    <div v-if="pollingInterval">
                      Confirmando compra ({{pollingCount}})...
                    </div>
                  </b-tab>
```

```
      data: () => ({
        loadedCheckout: null,
        loading: true,
        polling: false,
        pollingCount: 0,
        pollingInterval: null,
      }),
```

```
    methods: {
      pollingStatus() {
        this.pollingInterval = setInterval(() => {
          this.pollingCount += 1;
          if (this.polling === false) {
            this.polling = true;
            api
              .getCheckout({
                id: this.$route.params.id,
              })
              .then(({ data }) => {
                this.loadedCheckout = data;
              })
              .catch((error) => {
                // eslint-disable-next-line no-alert
                alert(error.message);
                clearInterval(this.pollingInterval);
              })
              .finally(() => {
```

```
                if (this.loadedCheckout.status !== 'pending') {
                  clearInterval(this.pollingInterval);
                  this.$swal
                    .fire({
                      icon: 'success',
                      title: 'Tudo certo!',
                      text: 'Pagamento confirmado.',
                      showConfirmButton: false,
                      timer: 5000,
                    })
                    .then(() => {
                      this.$router.replace({
                        name: 'cart',
                      });
                    });
                }
                setTimeout(() => {
                  this.polling = false;
                }, 200);
              });
          }
        }, 500);
    },
  },
  mounted() {
    this.pollingStatus();
  },
```