CENTRO UNIVERSITÁRIO FATEC ITAPETININGA

Curso de Análise e Desenvolvimento de Sistemas – 5º Ciclo Noturno

SISTEMA DE GESTÃO DE PEDIDOS

Documentação Técnica do Projeto Integrador

Integrantes do Grupo:

- Matheus Marques Fernandes Vieira
- Vinicius Santos de Oliveira
- Pedro Luiz Baptista Sardela
- Matheus de Oliveira Vieira
- Alex Lima de Moraes

Professor: Maylon Pires Macedo
Disciplina: Engenharia de Software / Desenvolvimento Web

Link do GitHub: https://github.com/Grupo-dos-Grupos/maiconsoft

Descrição:
Este documento apresenta a documentação técnica completa do **Sistema de Gestão de Pedidos**, desenvolvido como projeto acadêmico do curso de **Análise e Desenvolvimento de Sistemas**.
O sistema tem como objetivo aplicar os princípios de **Engenharia de Software**, **Arquitetura MVC** e **Desenvolvimento Web Full Stack**, utilizando **Node.js, Express, React e PostgreSQL**, com integração de **Inteligência Artificial (Google Gemini)** para automação de descrições de produtos.

SUMÁRIO

3. INTRODUÇÃO

3.1 Contexto

Este documento apresenta a documentação técnica completa do Sistema de Gestão de Pedidos, desenvolvido como projeto acadêmico/profissional utilizando tecnologias modernas de desenvolvimento web.

3.2 Escopo

O sistema abrange o gerenciamento completo de clientes, produtos e pedidos, seguindo o padrão arquitetural MVC (Model-View-Controller) e utilizando PostgreSQL como banco de dados relacional.

3.3 Público-Alvo

Esta documentação destina-se a desenvolvedores, analistas de sistemas e demais profissionais de TI que necessitem compreender, manter ou expandir o sistema.

4. OBJETIVOS DO SISTEMA

4.1 Objetivo Geral

Desenvolver um sistema web completo para gestão de pedidos, permitindo o controle eficiente de clientes, produtos e suas transações comerciais.

4.2 Objetivos Específicos

- Implementar CRUD completo para gerenciamento de clientes

- Implementar CRUD completo para gerenciamento de produtos
- Implementar CRUD completo para gerenciamento de pedidos
- Estabelecer relacionamentos entre entidades (Cliente-Pedido-Produto)
- Calcular automaticamente o total dos pedidos
- Integrar inteligência artificial para geração de descrições de produtos
- Desenvolver interface responsiva e intuitiva

5. FUNCIONALIDADES

5.1 Funcionalidades Obrigatórias

5.1.1 Padrão MVC

O sistema foi desenvolvido seguindo rigorosamente a arquitetura Model-View-Controller, separando as responsabilidades em três camadas distintas:

- **Model:** Representa a estrutura de dados e lógica de negócio
- **View:** Interface do usuário (React)
- **Controller:** Intermediação entre Model e View

5.1.2 CRUD de Clientes

Gerenciamento completo de clientes com as seguintes operações:

- **Create:** Cadastro de novos clientes
- **Read:** Listagem e busca de clientes
- **Update:** Atualização de dados cadastrais
- **Delete:** Exclusão de clientes

Campos gerenciados:

- Nome completo
- E-mail
- Telefone

5.1.3 CRUD de Produtos

Sistema completo de gerenciamento de produtos incluindo:

- Cadastro de novos produtos
- Listagem com filtros
- Atualização de informações
- Exclusão de produtos
- Geração automática de descrições via IA (Google Gemini)

Campos gerenciados:

- Nome do produto
- Preço
- Descrição

5.1.4 CRUD de Pedidos

Funcionalidade mais complexa do sistema, permitindo:

- Criação de pedidos associados a clientes
- Adição de múltiplos produtos ao pedido
- Cálculo automático do valor total
- Controle de status do pedido
- Gestão de quantidades e preços unitários

Relacionamentos:

- 1 Pedido pertence a 1 Cliente
- 1 Pedido pode conter N Produtos
- 1 Produto pode estar em N Pedidos

5.1.5 Tratamento de Exceções

Implementação robusta de tratamento de erros:

- Middleware centralizado de erros
- Validação de dados de entrada

- Mensagens de erro descritivas
- Logs de erros para debugging
- Respostas HTTP apropriadas

5.1.6 Diagramas

Documentação visual completa incluindo:

- Diagrama de Arquitetura Cliente-Servidor
- Diagrama Entidade-Relacionamento (ER)
- Diagrama do padrão MVC

5.2 Funcionalidades Extras

5.2.1 Bootstrap

Integração completa do framework Bootstrap 5:

- Design System profissional
- Componentes responsivos
- Grid system
- Utilitários de CSS
- Interface moderna e acessível

5.2.2 Integração Google Gemini

Implementação de Inteligência Artificial para:

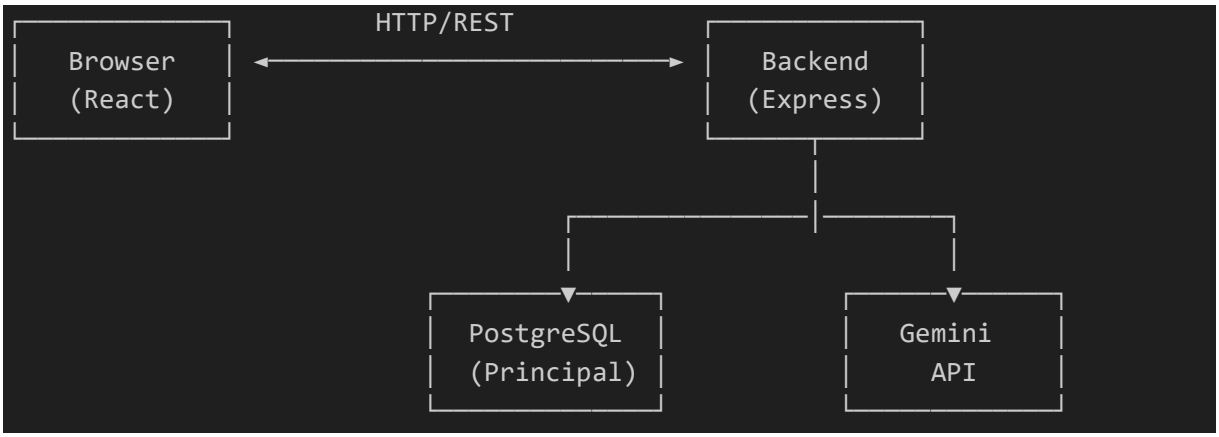
- Geração automática de descrições de produtos
- Otimização de conteúdo
- Melhor experiência do usuário
- Redução do tempo de cadastro

6. ARQUITETURA DO SISTEMA

6.1 Visão Geral

O sistema utiliza arquitetura Cliente-Servidor com separação clara entre frontend e backend.

6.2 Diagrama de Arquitetura



6.3 Camadas da Aplicação

6.3.1 Camada de Apresentação (Frontend)

- **Tecnologia:** React 18
- **Responsabilidade:** Interface do usuário
- **Componentes:** Pages, Components, Services

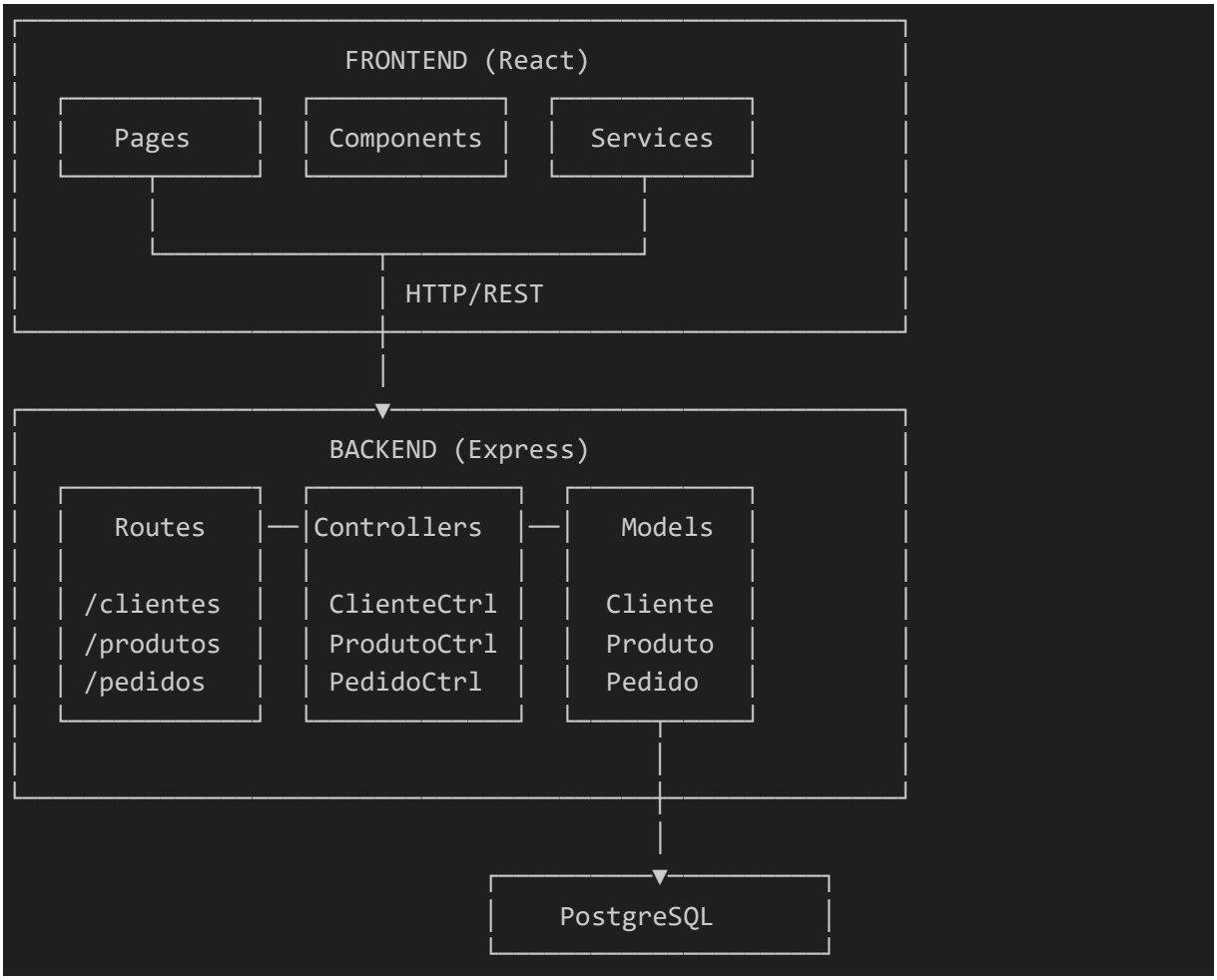
6.3.2 Camada de Aplicação (Backend)

- **Tecnologia:** Node.js + Express
- **Responsabilidade:** Lógica de negócio e API REST
- **Componentes:** Routes, Controllers, Services

6.3.3 Camada de Dados

- **Tecnologia:** PostgreSQL
- **Responsabilidade:** Persistência e integridade dos dados
- **Componentes:** Models, Database Config

6.4 Padrão MVC



7. TECNOLOGIAS UTILIZADAS

7.1 Backend

7.1.1 Node.js

- **Versão:** 18+
- **Descrição:** Runtime JavaScript server-side
- **Justificativa:** Performance, grande comunidade, ecosistema robusto

7.1.2 Express.js

- **Versão:** 4.x
- **Descrição:** Framework web minimalista
- **Justificativa:** Simplicidade, flexibilidade, middleware system

7.1.3 PostgreSQL

- **Versão:** 14+
- **Descrição:** Banco de dados relacional open-source
- **Justificativa:** Robustez, ACID compliance, features avançadas

7.1.4 pg (node-postgres)

- **Descrição:** Cliente PostgreSQL para Node.js
- **Uso:** Conexão e queries ao banco de dados

7.1.5 Google Gemini API

- **Descrição:** API de inteligência artificial
- **Uso:** Geração automática de descrições de produtos

7.1.6 Jest

- **Descrição:** Framework de testes
- **Uso:** Testes unitários e de integração

7.2 Frontend

7.2.1 React

- **Versão:** 18
- **Descrição:** Biblioteca JavaScript para UI
- **Justificativa:** Component-based, virtual DOM, grande comunidade

7.2.2 React Router DOM

- **Descrição:** Biblioteca de roteamento
- **Uso:** Navegação entre páginas SPA

7.2.3 Bootstrap 5

- **Descrição:** Framework CSS
- **Uso:** Design System e componentes responsivos

7.2.4 React Bootstrap

- **Descrição:** Componentes Bootstrap para React
- **Uso:** Integração Bootstrap com React

7.2.5 Axios

- **Descrição:** Cliente HTTP
- **Uso:** Requisições à API

7.2.6 React Hot Toast

- **Descrição:** Biblioteca de notificações
- **Uso:** Feedback visual para o usuário

8. MODELAGEM DE DADOS

8.1 Diagrama Entidade-Relacionamento

8.2 Descrição das Entidades

8.2.1 Clientes

Armazena informações dos clientes do sistema.

Campos:

- `id` (SERIAL, PK): Identificador único
- `nome` (VARCHAR(255), NOT NULL): Nome completo
- `email` (VARCHAR(255), UNIQUE, NOT NULL): E-mail
- `telefone` (VARCHAR(20)): Número de telefone
- `created_at` (TIMESTAMP): Data de cadastro
- `updated_at` (TIMESTAMP): Data da última atualização

Restrições:

- Email único no sistema
- Nome obrigatório

8.2.2 Produtos

Catálogo de produtos disponíveis.

Campos:

- `id` (SERIAL, PK): Identificador único
- `nome` (VARCHAR(255), NOT NULL): Nome do produto
- `preco` (DECIMAL(10,2), NOT NULL): Preço unitário
- `descricao` (TEXT): Descrição detalhada
- `created_at` (TIMESTAMP): Data de cadastro
- `updated_at` (TIMESTAMP): Data da última atualização

Restrições:

- Preço deve ser maior que zero
- Nome obrigatório

8.2.3 Pedidos

Registros de pedidos realizados.

Campos:

- `id` (SERIAL, PK): Identificador único
- `cliente_id` (INTEGER, FK, NOT NULL): Referência ao cliente
- `data` (TIMESTAMP, NOT NULL): Data do pedido
- `status` (VARCHAR(50), NOT NULL): Status atual
- `total` (DECIMAL(10,2), NOT NULL): Valor total
- `created_at` (TIMESTAMP): Data de criação

- `updated_at` (TIMESTAMP): Data da última atualização

Status possíveis:

- Pendente
- Em Processamento
- Enviado
- Entregue
- Cancelado

Restrições:

- Cliente obrigatório
- Total calculado automaticamente

8.2.4 PedidoProduto (Tabela de Relacionamento)

Relaciona pedidos com produtos (N:N).

Campos:

- `pedido_id` (INTEGER, FK, PK): Referência ao pedido
- `produto_id` (INTEGER, FK, PK): Referência ao produto
- `quantidade` (INTEGER, NOT NULL): Quantidade solicitada
- `preco_unitario` (DECIMAL(10,2), NOT NULL): Preço no momento da compra
- `subtotal` (DECIMAL(10,2), NOT NULL): Quantidade × Preço unitário

Restrições:

- Quantidade deve ser maior que zero
- Subtotal calculado automaticamente

8.3 Relacionamentos

8.3.1 Cliente → Pedido (1:N)

- Um cliente pode ter vários pedidos
- Um pedido pertence a apenas um cliente
- Relacionamento obrigatório

8.3.2 Pedido ↔ Produto (N:N)

- Um pedido pode conter vários produtos
- Um produto pode estar em vários pedidos
- Relacionamento intermediado pela tabela PedidoProduto

9. ESTRUTURA DO PROJETO

9.1 Estrutura Backend

```
backend/
├── config/
│   └── database.js       # Configuração PostgreSQL
├── models/               # Models (MVC)
│   ├── Cliente.js
│   ├── Produto.js
│   └── Pedido.js
├── controllers/          # Controllers (MVC)
│   ├── ClienteController.js
│   ├── ProdutoController.js
│   └── PedidoController.js
├── routes/               # Routes (MVC)
│   ├── clientes.js
│   ├── produtos.js
│   └── pedidos.js
├── services/
│   └── openaiService.js  # Integração Google Gemini
├── middleware/
│   └── errorHandler.js   # Tratamento de erros
├── tests/                # Testes
├── .env                  # Variáveis de ambiente
├── package.json
└── server.js             # Entrada da aplicação
```

9.2 Estrutura Frontend

```
frontend/
├── public/
│   └── index.html
├── src/
│   ├── components/
```

```
│   │   └── Layout.js
│   ├── pages/
│   │   ├── Dashboard.js
│   │   ├── ClientesList.js
│   │   ├── ClienteForm.js
│   │   ├── ProdutosList.js
│   │   ├── ProdutoForm.js
│   │   ├── PedidosList.js
│   │   └── PedidoForm.js
│   ├── services/
│   │   └── api.js
│   ├── App.js
│   ├── index.js
│   └── index.css
├── package.json
└── README.md
```

9.3 Descrição dos Componentes

9.3.1 Backend

config/database.js

- Configuração da conexão com PostgreSQL
- Pool de conexões
- Tratamento de erros de conexão

models/

- Definição das entidades
- Métodos de acesso ao banco
- Validações de dados

controllers/

- Lógica de negócio
- Processamento de requisições
- Validações de entrada

routes/

- Definição de endpoints
- Mapeamento de rotas para controllers

- Middlewares de validação

services/openaiService.js

- Integração com Google Gemini
- Geração de descrições
- Tratamento de erros da API

middleware/errorHandler.js

- Captura de erros
- Formatação de respostas de erro
- Logging

9.3.2 Frontend

components/Layout.js

- Menu de navegação
- Estrutura comum das páginas
- Footer

pages/

- Páginas da aplicação
- Formulários de cadastro
- Listagens com filtros

services/api.js

- Configuração do Axios
- Interceptors
- Métodos de requisição

10. INSTALAÇÃO E CONFIGURAÇÃO

10.1 Pré-requisitos

Software necessário:

- Node.js 18 ou superior
- PostgreSQL 14 ou superior

- npm ou yarn
- Git (opcional)

Verificar instalações:

bash
```
node --version
npm --version
psql --version
```

10.2 Configuração do Banco de Dados

10.2.1 Criar Banco de Dados

Via psql:

sql
```
CREATE DATABASE sistema_pedidos;
```

Ou via pgAdmin:

1. Abrir pgAdmin
2. Conectar ao servidor PostgreSQL
3. Clicar com botão direito em "Databases"
4. Selecionar "Create" → "Database"
5. Nomear como "sistema_pedidos"

10.2.2 Scripts SQL

As tabelas serão criadas automaticamente na primeira execução através dos Models.

10.3 Variáveis de Ambiente

Criar arquivo `.env` na pasta `backend/`:

env
```
# PostgreSQL
DB_HOST=localhost
DB_PORT=5432
DB_NAME=sistema_pedidos
DB_USER=postgres
DB_PASSWORD=postgres
```

```
# Google Gemini (opcional)
GEMINI_API_KEY=sua-chave-gemini-aqui

# Servidor
PORT=3001
NODE_ENV=development
```

10.4 Instalação

10.4.1 Backend

bash

```
# Navegar para pasta backend
cd backend

# Instalar dependências
npm install

# Executar em desenvolvimento
npm run dev

# Ou em produção
npm start
```

10.4.2 Frontend

bash

```
# Navegar para pasta frontend
cd frontend

# Instalar dependências
npm install

# Executar em desenvolvimento
npm start

# Build para produção
npm run build
```

10.5 Verificação da Instalação

1. Acessar http://localhost:3000 (Frontend)

2. Verificar se a página carrega corretamente
3. Testar endpoint de saúde: http://localhost:3001/api/health
4. Tentar criar um cliente de teste

11. ENDPOINTS DA API

11.1 Clientes

11.1.1 Listar Todos os Clientes

```
GET /api/clientes
```

Resposta de Sucesso (200):

json

```
[
  {
    "id": 1,
    "nome": "João Silva",
    "email": "joao@email.com",
    "telefone": "(11) 98765-4321",
    "created_at": "2024-01-15T10:30:00Z",
    "updated_at": "2024-01-15T10:30:00Z"
  }
]
```

11.1.2 Buscar Cliente por ID

```
GET /api/clientes/:id
```

Parâmetros:

- `id` (path): ID do cliente

Resposta de Sucesso (200):

json

```
{
  "id": 1,
  "nome": "João Silva",
  "email": "joao@email.com",
  "telefone": "(11) 98765-4321"
}
```

Resposta de Erro (404):

json

```
{
  "error": "Cliente não encontrado"
}
```

11.1.3 Criar Cliente

```
POST /api/clientes
```

Body:

json

```
{
  "nome": "Maria Santos",
  "email": "maria@email.com",
  "telefone": "(11) 91234-5678"
}
```

Resposta de Sucesso (201):

json

```
{
  "id": 2,
  "nome": "Maria Santos",
  "email": "maria@email.com",
  "telefone": "(11) 91234-5678"
}
```

11.1.4 Atualizar Cliente

```
PUT /api/clientes/:id
```

Body:

json

```
{
  "nome": "Maria Santos Silva",
  "email": "maria.silva@email.com",
  "telefone": "(11) 91234-5678"
}
```

Resposta de Sucesso (200):

json

```
{
  "id": 2,
  "nome": "Maria Santos Silva",
  "email": "maria.silva@email.com",
  "telefone": "(11) 91234-5678"
}
```

11.1.5 Excluir Cliente

```
DELETE /api/clientes/:id
```

Resposta de Sucesso (200):

json

```
{
  "message": "Cliente excluído com sucesso"
}
```

11.2 Produtos

11.2.1 Listar Todos os Produtos

```
GET /api/produtos
```

Resposta de Sucesso (200):

json

```
[
  {
    "id": 1,
    "nome": "Notebook Dell",
    "preco": 3500.00,
    "descricao": "Notebook para uso profissional..."
```

```
  }
]
```

11.2.2 Criar Produto

```
POST /api/produtos
```

Body:

json
```
{
  "nome": "Mouse Logitech",
  "preco": 89.90,
  "descricao": "Mouse ergonômico wireless",
  "generateDescription": false
}
```

Com geração automática de descrição:

json
```
{
  "nome": "Mouse Logitech MX Master 3",
  "preco": 89.90,
  "generateDescription": true
}
```

11.3 Pedidos

11.3.1 Listar Todos os Pedidos

```
GET /api/pedidos
```

Resposta de Sucesso (200):

json
```
[
  {
    "id": 1,
    "cliente_id": 1,
    "cliente_nome": "João Silva",
    "data": "2024-01-15T14:30:00Z",
    "status": "Pendente",
    "total": 3589.90,
    "produtos": [
```

```
      {
        "produto_id": 1,
        "nome": "Notebook Dell",
        "quantidade": 1,
        "preco_unitario": 3500.00,
        "subtotal": 3500.00
      },
      {
        "produto_id": 2,
        "nome": "Mouse Logitech",
        "quantidade": 1,
        "preco_unitario": 89.90,
        "subtotal": 89.90
      }
    ]
  }
]
```

11.3.2 Criar Pedido

`POST /api/pedidos`

Body:

json

```
{
  "clienteId": 1,
  "status": "Pendente",
  "produtos": [
    {
      "produtoId": 1,
      "quantidade": 2
    },
    {
      "produtoId": 3,
      "quantidade": 1
    }
  ]
}
```

Resposta de Sucesso (201):

json

```
{
  "id": 2,
  "cliente_id": 1,
  "data": "2024-01-15T15:00:00Z",
  "status": "Pendente",
  "total": 7089.90
}
```


11.4 Dashboard

11.4.1 Obter Métricas

```
GET /api/dashboard
```

Resposta de Sucesso (200):

json


```
{
  "totalClientes": 50,
  "totalProdutos": 120,
  "totalPedidos": 230,
  "valorTotalPedidos": 125430.50
}
```


12. INTERFACE DO USUÁRIO

12.1 Design System

Framework: Bootstrap 5

Características:

- Design responsivo
- Componentes reutilizáveis
- Grid system de 12 colunas
- Utilitários CSS
- Tema customizável

12.2 Páginas da Aplicação

12.2.1 Dashboard

- Visão geral do sistema
- Cards com métricas principais
- Gráficos (se implementado)
- Ações rápidas

12.2.2 Listagem de Clientes

- Tabela responsiva
- Busca e filtros
- Ações: Editar, Excluir
- Paginação
- Botão "Novo Cliente"

12.2.3 Formulário de Cliente

- Campos: Nome, Email, Telefone
- Validação em tempo real
- Botões: Salvar, Cancelar
- Feedback de erro/sucesso

12.2.4 Listagem de Produtos

- Grid de cards com produtos
- Informações: Nome, Preço, Descrição
- Ações: Editar, Excluir
- Botão "Novo Produto"

12.2.5 Formulário de Produto

- Campos: Nome, Preço, Descrição
- Checkbox para gerar descrição com IA
- Preview da descrição gerada
- Validação de preço

12.2.6 Listagem de Pedidos

- Tabela com pedidos
- Informações: Cliente, Data, Status, Total
- Filtros por status
- Visualização de detalhes
- Badge de status colorido

12.2.7 Formulário de Pedido

- Seleção de cliente (dropdown)
- Adicionar múltiplos produtos
- Campos por produto: Seleção, Quantidade
- Cálculo automático de subtotais
- Cálculo do total geral
- Seleção de status

12.3 Componentes Comuns

12.3.1 Navbar

- Logo do sistema
- Menu de navegação
- Links: Dashboard, Clientes, Produtos, Pedidos
- Responsivo (hamburger menu em mobile)

12.3.2 Cards

- Container para informações
- Header com título
- Body com conteúdo
- Footer com ações

12.3.3 Tabelas

- Responsivas (scroll horizontal em mobile)
- Striped rows
- Hover effect
- Ações por linha

12.3.4 Formulários

- Labels descritivos
- Inputs com validação
- Mensagens de erro
- Botões de ação

12.3.5 Modais

- Confirmação de exclusão
- Visualização de detalhes
- Backdrop escurecido

12.3.6 Toasts

- Notificações de sucesso
- Notificações de erro
- Auto-dismiss
- Posição superior direita

12.4 Responsividade

Breakpoints Bootstrap:

- Extra Small (xs): < 576px
- Small (sm): ≥ 576px
- Medium (md): ≥ 768px
- Large (lg): ≥ 992px
- Extra Large (xl): ≥ 1200px
- Extra Extra Large (xxl): ≥ 1400px

Adaptações:

- Menu hambúrguer em mobile
- Cards empilhados em telas pequenas
- Tabelas com scroll horizontal
- Formulários em coluna única.

CONCLUSÃO

O **Sistema de Gestão de Pedidos** atingiu com êxito todos os objetivos propostos, demonstrando a aplicação prática dos conceitos de **Engenharia de Software** e **Desenvolvimento Web**.

Através da utilização do **padrão arquitetural MVC (Model-View-Controller)**, o sistema foi estruturado de forma modular e organizada, permitindo fácil manutenção, expansão e reutilização de código. As funcionalidades de **CRUD completo** para clientes, produtos e pedidos foram implementadas com sucesso, garantindo a integridade dos dados e a interação eficiente entre as camadas de aplicação e banco de dados.

A integração com o **PostgreSQL** proporcionou uma base de dados relacional robusta, enquanto o **frontend desenvolvido em React** com o uso do **Bootstrap 5** assegurou uma interface moderna, responsiva e intuitiva. Além disso, a inclusão da **API do Google Gemini** trouxe um diferencial tecnológico, automatizando a geração de descrições de produtos e demonstrando a aplicação de **inteligência artificial** no contexto do sistema.

Com diagramas que representam a **arquitetura cliente-servidor**, a **modelagem entidade-relacionamento** e o **padrão MVC**, a documentação oferece uma visão completa do funcionamento interno do sistema e de suas principais interações.

Em síntese, o projeto cumpre integralmente os requisitos da disciplina, servindo como uma implementação sólida e escalável, apta a ser consumida por outras aplicações web. O resultado final comprova o domínio dos fundamentos de análise, modelagem e desenvolvimento de sistemas modernos, alinhando teoria e prática de forma coerente e eficaz.