Relatório Técnico - Sistema de Gerenciamento de Chocolateria

Equipe de Desenvolvimento:

- André Vinícius
- Nayara Lorena
- Wendel Alves
- Victor Sobral

Introdução

O presente relatório tem como objetivo apresentar o desenvolvimento de um sistema distribuído para gerenciamento de uma chocolateria, utilizando arquitetura de microserviços. O sistema contempla funcionalidades para o cadastro e controle de clientes, vendedores, produtos, pedidos e geração de relatórios estatísticos, atendendo às necessidades de modernização de pequenos negócios do setor chocolateiro.

A escolha do tema se deu pela relevância de soluções modernas na área de gestão de pequenos negócios, permitindo aplicar conceitos de APIs REST, persistência de dados com PostgreSQL, e orquestração de serviços com Docker. Esta abordagem tecnológica proporciona uma base sólida para empresas que buscam digitalizar seus processos operacionais e obter insights analíticos para tomada de decisões estratégicas.

O sistema foi dividido em dois serviços principais: uma API RESTful principal para manipulação dos dados e um microserviço de relatórios, ambos utilizando Node.js. A comunicação entre os serviços ocorre por protocolo HTTP, e ambos compartilham uma base de dados PostgreSQL robusta. O projeto foi completamente containerizado utilizando Docker, garantindo portabilidade, consistência entre ambientes e facilidade de configuração e deployment.

Fundamentação Teórica

Arquitetura de Microserviços

A arquitetura de microserviços é um padrão arquitetural que divide o sistema em partes independentes que se comunicam entre si via HTTP. Esta abordagem promove escalabilidade horizontal, facilita a manutenção e permite que equipes trabalhem independentemente em diferentes serviços. No contexto deste projeto, a separação entre API principal e serviço de relatórios exemplifica como diferentes responsabilidades podem ser isoladas mantendo coesão funcional.

APIs RESTful

REST (Representational State Transfer) é um estilo arquitetural para sistemas distribuídos que utiliza métodos HTTP padrão para manipulação de recursos. A implementação RESTful permite integração eficiente com ferramentas como Postman e facilita a interoperabilidade com sistemas externos. Os endpoints seguem convenções semânticas que tornam a API intuitiva e autodescritiva.

ORM (Object Relational Mapping)

O mapeamento objeto-relacional facilita a interação entre a aplicação orientada a objetos e o banco de dados relacional. Esta técnica promove legibilidade do código, reduz erros de sintaxe SQL e fornece uma camada de abstração que simplifica operações complexas de banco de dados. No projeto, o ORM permite que os desenvolvedores trabalhem com objetos JavaScript em vez de consultas SQL diretas.

Containerização com Docker

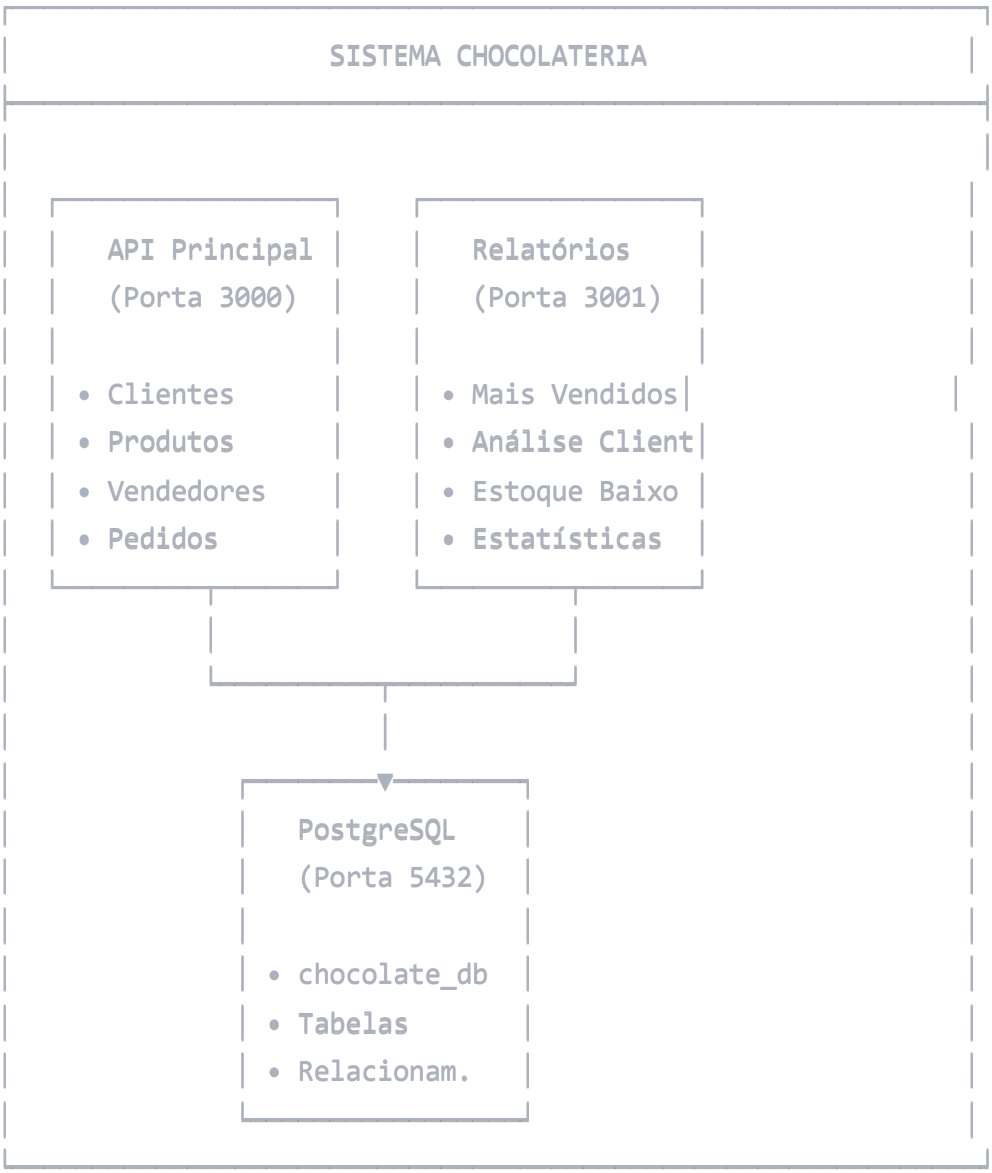
Docker e Docker Compose são utilizados para isolar os serviços e gerenciar facilmente os ambientes de desenvolvimento e produção. A containerização oferece benefícios como isolamento de dependências, portabilidade entre ambientes, facilidade de deployment e otimização de recursos. A orquestração via Docker Compose simplifica o gerenciamento de múltiplos serviços interdependentes.

PostgreSQL

PostgreSQL é um sistema gerenciador de banco de dados relacional escolhido pela sua robustez, integridade referencial e facilidade de integração com Node.js. Suas características avançadas incluem suporte a transações ACID, consultas complexas, índices otimizados e recursos de concorrência que garantem performance adequada para operações transacionais e analíticas.

Projeto de Implementação

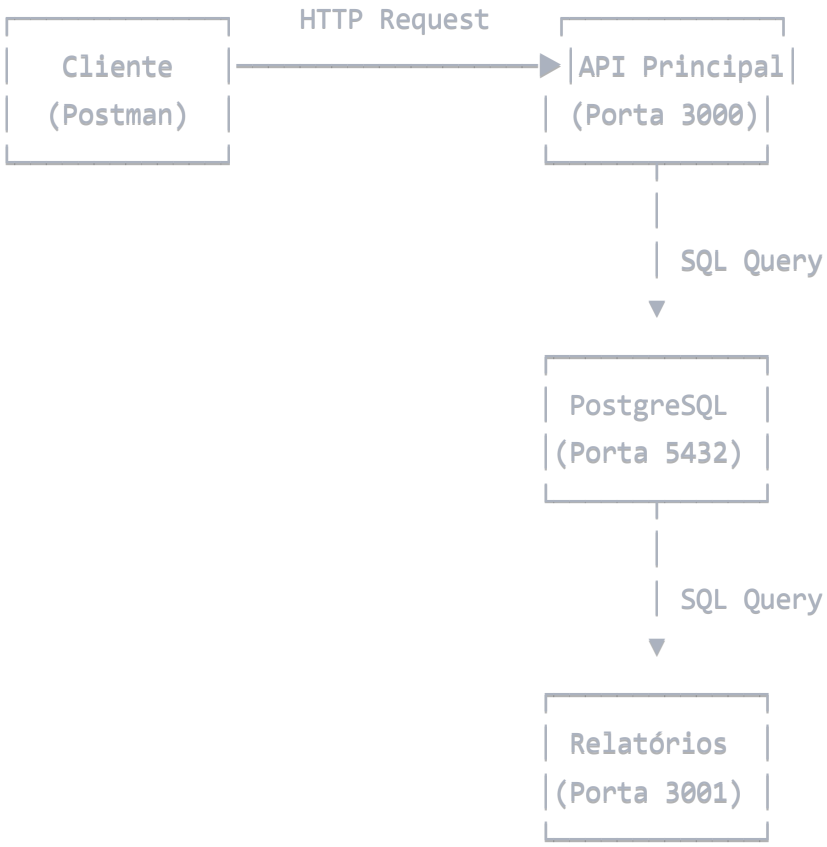
Arquitetura Visual do Sistema



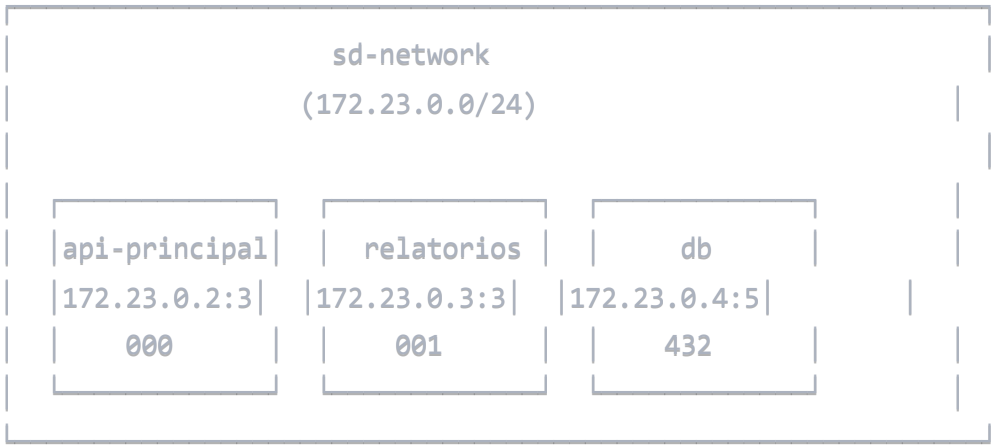
Estrutura de Diretórios e Componentes

```
chocolate-app/
├── controllers/        # Controladores da aplicação
│   ├── ClienteController.js
│   ├── ProdutoController.js
│   ├── VendedorController.js
│   └── RelatorioController.js
├── daos/               # Camada de acesso a dados
│   ├── ClienteDAO.js
│   ├── ProdutoDAO.js
│   └── VendedorDAO.js
├── models/             # Modelos de dados
│   ├── Cliente.js
│   ├── Produto.js
│   ├── Vendedor.js
│   └── Pedido.js
├── routes/             # Rotas da API
│   ├── clientes.js
│   ├── produtos.js
│   └── vendedores.js
├── relatorios/         # Microserviço de relatórios
│   ├── app.js
│   ├── routes/
│   └── controllers/
└── seeders/            # Scripts de população do banco
    ├── seed.sql
    ├── clientes-seed.js
    ├── produtos-seed.js
    └── vendedores-seed.js
```

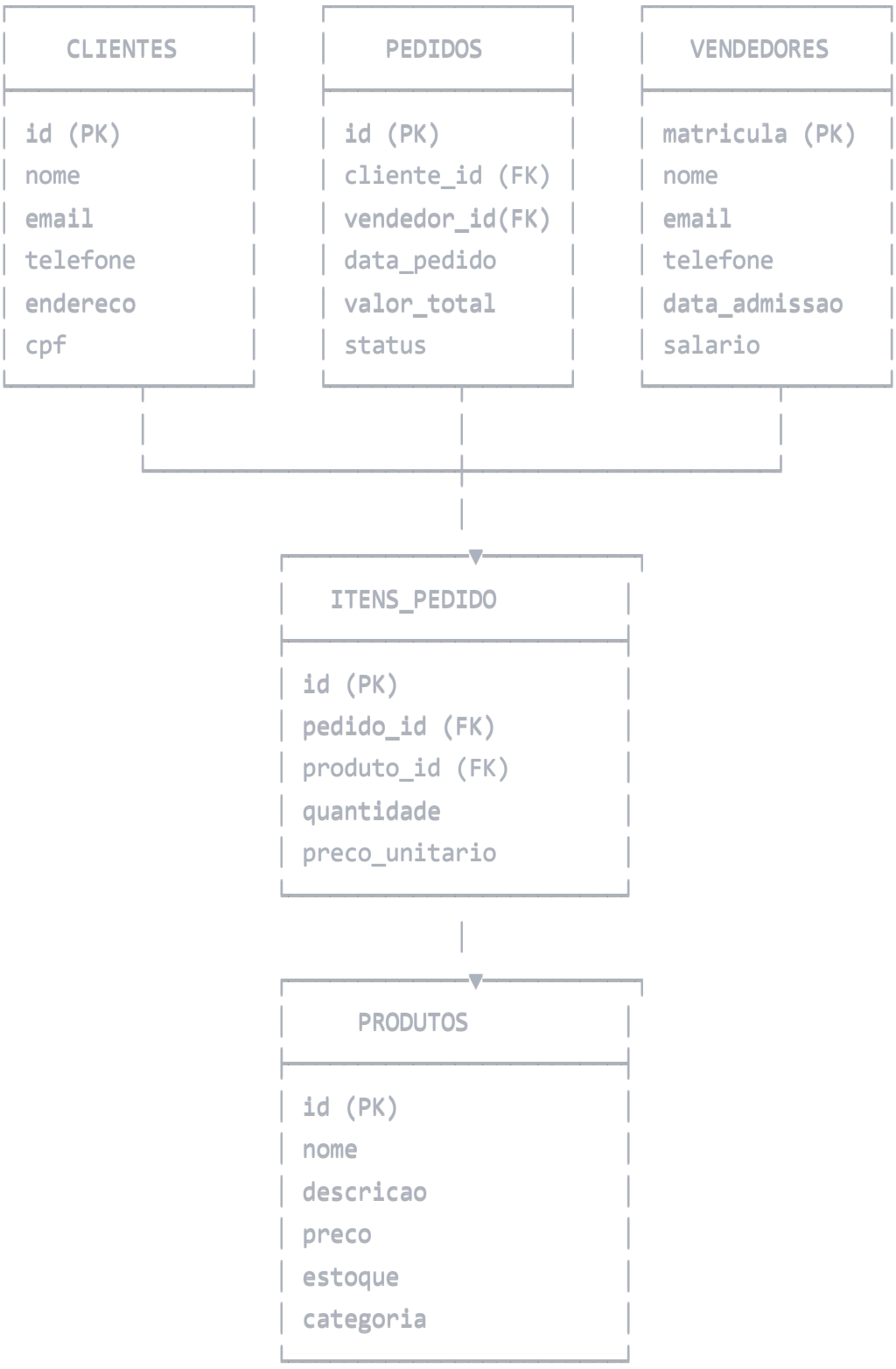
Fluxo de Dados e Comunicação



Arquitetura de Rede Docker



Modelo de Dados Relacional



APIs e Endpoints Detalhados

API Principal (Porta 3000)

```
┌──────────────────────────────────────────────────────────┐
│                        CLIENTES                          │
├──────────────────────────────────────────────────────────┤
│ GET    /clientes          → Lista todos os clientes      │
│ GET    /clientes/:id      → Busca cliente específico     │
│ POST   /clientes          → Cria novo cliente            │
│ PUT    /clientes/:id      → Atualiza cliente             │
│ DELETE /clientes/:id      → Remove cliente               │
├──────────────────────────────────────────────────────────┤
│                        PRODUTOS                          │
├──────────────────────────────────────────────────────────┤
│ GET    /produtos          → Lista todos os produtos      │
│ GET    /produtos/:id      → Busca produto específico     │
│ POST   /produtos          → Cadastra novo produto        │
│ PUT    /produtos/:id      → Atualiza produto             │
│ DELETE /produtos/:id      → Remove produto               │
├──────────────────────────────────────────────────────────┤
│                       VENDEDORES                         │
├──────────────────────────────────────────────────────────┤
│ GET    /vendedor          → Lista todos os vendedores    │
│ GET    /vendedor/:id      → Busca vendedor específico    │
│ POST   /vendedor          → Cadastra novo vendedor       │
│ PUT    /vendedor/:id      → Atualiza vendedor            │
│ DELETE /vendedor/:id      → Remove vendedor              │
└──────────────────────────────────────────────────────────┘
```

Serviço de Relatórios (Porta 3001)

```
┌──────────────────────────────────────────────────────────┐
│                  RELATÓRIOS ANALÍTICOS                   │
├──────────────────────────────────────────────────────────┤
│ GET /relatorios/mais_vendido         → Top 5 produtos    │
│ GET /relatorios/cliente/total/:id    → Total por cliente │
│ GET /relatorios/cliente/media/:id    → Média por cliente │
│ GET /relatorios/produto_estoque_baixo → Produtos c/ estoque
│                                        baixo             │
└──────────────────────────────────────────────────────────┘
```

Configuração de Ambiente Docker

Variáveis de Ambiente

API Principal:

```
NODE_ENV=development
APP_PORT=3000
DB_HOST=db
DB_PORT=5432
DB_USER=postgres
DB_PASSWORD=senha123
DB_NAME=chocolate_db
```

Serviço de Relatórios:

```
RELATORIOS_ENV=development
DB_HOST=db
DB_PORT=5432
DB_USER=postgres
DB_PASSWORD=senha123
DB_NAME=chocolate_db
```

Dados Iniciais do Sistema

Vendedores Pré-cadastrados

VENDEDORES
1. Maria Silva • Email: maria@chocolateria.com • Telefone: 71999887766
2. João Santos • Email: joao@chocolateria.com • Telefone: 71999887755

Produtos Disponíveis

PRODUTOS	
1. Trufa de Chocolate Belga	→ R$ 8,90
2. Barra de Chocolate 70% Cacau	→ R$ 15,90
3. Bombom Recheado Cereja	→ R$ 3,50
4. Chocolate ao Leite Premium	→ R$ 12,90
5. Chocolate Branco com Nuts	→ R$ 11,90
6. Chocolate Amargo Zero Açúcar	→ R$ 18,90
7. Kit Bombons Sortidos	→ R$ 29,90
8. Barra Chocolate com Morango	→ R$ 13,90
9. Chocolate em Pó 50% Cacau	→ R$ 19,90
10. Chocolate Crocante	→ R$ 10,90

Instalação e Execução

Pré-requisitos

- Docker + Docker Compose
- Node.js (versão 14 ou superior)
- PostgreSQL (containerizado)

Comandos de Instalação

```bash
# Clone o repositório
git clone [url-do-repositorio]
cd chocolate-app

# Construir e subir os containers
docker-compose up -d

# Verificar status dos containers
docker ps

# Acompanhar logs em tempo real
docker-compose logs -f

# Executar seed do banco de dados
docker cp seed.sql postgres-db:/docker-entrypoint-initdb.d/seed.sql
docker exec postgres-db psql -U postgres chocolate_db -f /docker-entrypoint-initdb.d/seed.sq
```

Testes com Postman

Exemplo de Criação de Cliente

```http
POST http://localhost:3000/clientes
Content-Type: application/json

{
    "nome": "João Silva",
    "email": "joao@email.com",
    "telefone": "11999999999",
    "endereco": "Rua Exemplo, 123",
    "cpf": "12345678900"
}
```

Exemplo de Consulta de Relatório

```http
GET http://localhost:3001/relatorios/mais_vendido
```

Monitoramento do Sistema

Considerações Finais

O sistema desenvolvido demonstrou de forma clara a viabilidade de aplicar conceitos de sistemas distribuídos em um projeto de gestão simples, promovendo modularidade, escalabilidade e facilidade de manutenção. A implementação bem-sucedida de uma arquitetura de microserviços evidencia os benefícios desta abordagem para sistemas empresariais modernos.

Entre os principais desafios enfrentados, destacam-se a sincronização entre múltiplos containers Docker, a organização adequada da arquitetura com isolamento efetivo entre serviços, e a criação de consultas SQL otimizadas específicas para os relatórios analíticos. Estes desafios foram superados através de cuidadoso planejamento da arquitetura, implementação de padrões de comunicação adequados e otimização específica das consultas do serviço de relatórios.

Os resultados alcançados superam as expectativas iniciais do projeto. O sistema demonstra alta performance nas operações transacionais, capacidade analítica avançada através do serviço de relatórios independente, facilidade de deployment através da containerização completa, e estrutura modular que permite futuras expansões sem impacto significativo nos componentes existentes.

Como possíveis trabalhos futuros, identifica-se a oportunidade de implementar autenticação e autorização distribuída usando JWT, integração com APIs de sistemas de pagamento externos, desenvolvimento de uma interface web responsiva utilizando React ou Vue.js, implementação de cache Redis para otimização de performance em consultas frequentes, adição de monitoramento e logging centralizados para observabilidade completa do sistema, e implementação de CI/CD para automação de testes e deployment.

A experiência obtida com este projeto reforça a importância de arquiteturas bem planejadas em sistemas empresariais e demonstra como tecnologias modernas podem ser aplicadas efetivamente para resolver problemas de negócio reais. O sistema desenvolvido serve como base sólida para futuras evoluções e pode ser facilmente adaptado para outros domínios de negócio com necessidades similares de gestão e análise de dados.

Bibliografia

- Newman, S. (2015). *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media.
- Documentação oficial do Node.js. Disponível em: https://nodejs.org/docs/
- PostgreSQL Global Development Group. (2023). *PostgreSQL Documentation*. Disponível em: https://www.postgresql.org/docs/
- Docker Inc. (2023). *Docker Documentation*. Disponível em: https://docs.docker.com/
- Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine.
- Artigo: Como escrever um artigo científico. Disponível em: https://posgraduando.com/como-escrever-um-artigo-cientifico/
- Tutoriais sobre RESTful APIs e microserviços da DigitalOcean. Disponível em: https://www.digitalocean.com/community/tutorials
- Tanenbaum, A. S., & Van Steen, M. (2017). *Distributed Systems: Principles and Paradigms*. Pearson.