ServicoGestao - Sistema de Controle de Planos de Operadora

Descrição

Sistema backend desenvolvido em TypeScript seguindo a Arquitetura Limpa (Clean Architecture) proposta por Robert Martin. O sistema é responsável por gerenciar planos, clientes e assinaturas para operadoras de internet.

Tecnologias Utilizadas

Node.js - Runtime JavaScript
TypeScript - Linguagem de programação
Express.js - Framework web
SQLite - Banco de dados
Arquitetura Limpa - Padrão arquitetural

Estrutura do Projeto

```
src/
├── domain/              # Camada de Domínio
│   ├── entities/        # Entidades de negócio
│   ├── repositories/    # Interfaces dos repositórios
│   └── dtos/            # Data Transfer Objects
├── application/         # Camada de Aplicação
│   └── use-cases/       # Casos de uso
├── infrastructure/      # Camada de Infraestrutura
│   ├── database/        # Configuração do banco
│   └── repositories/    # Implementações dos repositórios
└── presentation/        # Camada de Apresentação
    ├── controllers/     # Controladores
    └── routes/          # Rotas da API
```

Princípios SOLID Aplicados

Single Responsibility Principle (SRP): Cada classe tem uma única responsabilidade
Open/Closed Principle (OCP): Classes abertas para extensão, fechadas para modificação
Liskov Substitution Principle (LSP): Interfaces podem ser substituídas por suas implementações
Interface Segregation Principle (ISP): Interfaces específicas para cada necessidade
Dependency Inversion Principle (DIP): Dependências por abstração, não por concreção

Padrões de Projeto Utilizados

Repository Pattern: Abstração do acesso a dados
Use Case Pattern: Encapsulamento da lógica de negócio
Singleton Pattern: Instância única do banco de dados

Dependency Injection: Injeção de dependências via construtor
MVC Pattern: Separação entre Model, View e Controller
Pré-requisitos

Node.js (versão 18 ou superior)
npm (gerenciador de pacotes)
Instalação e Execução

1. Instalar dependências

```
npm install
```

2. Compilar o projeto

```
npm run build
```

3. Popular banco de dados com dados iniciais

```
npm run seed
```

4. Executar o sistema

Modo desenvolvimento:
```
npm run dev
```

Modo produção:
```
npm start
```

O servidor será iniciado na porta 3000. Acesse: http://localhost:3000/health
Endpoints da API

Clientes

GET /gestao/clientes - Lista todos os clientes
Planos

GET /gestao/planos - Lista todos os planos
PATCH /gestao/planos/:idPlano - Atualiza custo mensal do plano
Assinaturas

POST /gestao/assinaturas - Cria uma assinatura
GET /gestao/assinaturas/:tipo - Lista assinaturas por tipo (TODOS|ATIVOS|CANCELADOS)
GET /gestao/assinaturascliente/:codcli - Lista assinaturas de um cliente
GET /gestao/assinaturasplano/:codplano - Lista assinaturas de um plano
Banco de Dados

O sistema utiliza SQLite como banco de dados. O arquivo database.sqlite é criado automaticamente na raiz do projeto.
Estrutura das Tabelas

clientes: codigo, nome, email
planos: codigo, nome, custoMensal, data, descricao
assinaturas: codigo, codPlano, codCli, inicioFidelidade, fimFidelidade, dataUltimoPagamento, custoFinal, descricao
pagamentos: codigo, codAss, valorPago, dataPagamento
Dados de Teste

O script de seeding cria:
12 clientes de exemplo
5 planos diferentes
6 assinaturas com cenários variados (ativas e inativas)
Regras de Negócio

Assinaturas

Fidelidade padrão de 1 ano (365 dias)
Assinatura é considerada ativa se:
Data atual está dentro do período de fidelidade
Último pagamento foi feito há no máximo 30 dias
Planos

Custo mensal deve ser maior que zero
Data de modificação é atualizada automaticamente
Variáveis de Ambiente

```
PORT=3000                # Porta do servidor
NODE_ENV=development     # Ambiente de execução
DB_PATH=./database.sqlite # Caminho do banco SQLite
```

Scripts Disponíveis

npm run build - Compila o TypeScript
npm start - Inicia em modo produção
npm run dev - Inicia em modo desenvolvimento
npm run seed - Popula banco com dados iniciais
Testes

Use a coleção do Postman fornecida (pedro_barros_Desenvolvimento_de_Sistemas_backend_Fase-1.postman_collection.json) para testar todos os endpoints.