Projeto de Desenvolvimento de Sistemas Backend - Fase 2

Introdução

Este documento descreve o projeto desenvolvido para a Fase 2 da disciplina de Desenvolvimento de Sistemas Backend, parte do curso de Desenvolvimento de Sistemas. O projeto consiste em uma arquitetura de microsserviços composta por quatro serviços independentes: **servico-gestao**, **servico-faturamento**, **servico-planos-ativos**, e **api-gateway**. O objetivo é implementar um sistema para gerenciar planos, cobranças, e planos ativos, utilizando tecnologias modernas como TypeScript, NestJS, MySQL, MongoDB, RabbitMQ, e Express, com integração via REST e mensageria assíncrona.

O projeto segue os princípios da *Clean Architecture* e os padrões SOLID, garantindo modularidade, escalabilidade, e manutenibilidade. Este README fornece uma visão geral da arquitetura, instruções para configuração e execução, e detalhes sobre os desafios enfrentados durante o desenvolvimento.

Arquitetura do Sistema

A arquitetura do projeto é baseada em microsserviços, com cada serviço desempenhando uma função específica e interagindo por meio de interfaces REST e mensageria RabbitMQ. Abaixo, detalha-se cada componente:

Microsserviços

- **servico-gestao** (Porta 3000):
  - **Função**: Gerencia planos, incluindo criação, consulta, e exclusão.
  - **Tecnologias**: TypeScript, NestJS, MySQL (persistência), RabbitMQ (consumo de eventos).
  - **Endpoints**:
    - `POST /planos` : Cria um novo plano.
    - `GET /planos/:id` : Consulta um plano por ID.
    - `DELETE /planos/:id` : Exclui um plano por ID.
  - **Integração**: Consome eventos `cobranca.created` do RabbitMQ para registrar logs na tabela `cobranca_logs` .
- **servico-faturamento** (Porta 3001):

    - **Função**: Gerencia cobranças, permitindo a criação de novas cobranças.
    - **Tecnologias**: TypeScript, NestJS, MongoDB (persistência), RabbitMQ (publicação de eventos).
    - **Endpoint**:
        - `POST /cobrancas` : Cria uma nova cobrança.
    - **Integração**: Publica eventos `cobranca.created` no RabbitMQ.
- **servico-planos-ativos** (Porta 3002):
    - **Função**: Gerencia planos ativos associados a clientes.
    - **Tecnologias**: TypeScript, NestJS, MongoDB (persistência), REST (integração com `servico-gestao`).
    - **Endpoints**:
        - `POST /planos-ativos` : Cria um plano ativo.
        - `GET /planos-ativos/cliente/:clienteId` : Consulta planos ativos por cliente ID.
    - **Integração**: Consulta o `servico-gestao` via REST para obter detalhes de planos.
- **api-gateway** (Porta 3003):
    - **Função**: Ponto único de entrada, roteando requisições para os outros microsserviços.
    - **Tecnologias**: TypeScript, NestJS, Express, Swagger (documentação).
    - **Endpoints**: Replica os endpoints dos outros serviços, acessíveis via `http://localhost:3003` .
    - **Integração**: Encaminha requisições HTTP para os serviços correspondentes.

Camadas da Clean Architecture

Cada microsserviço segue a *Clean Architecture*, com as seguintes camadas:

- **Domínio**: Define entidades (ex.: `Plano` , `Cobranca` , `PlanoAtivo`) e interfaces de repositórios (ex.: `IPlanoRepository`), contendo a lógica de negócios independente de tecnologias externas.
- **Aplicação**: Implementa serviços (ex.: `PlanoService`) que orquestram a lógica de negócios, utilizando as interfaces do domínio.
- **Interface**: Expõe controladores (ex.: `PlanoController`) que mapeiam requisições HTTP para ações do sistema.
- **Infraestrutura**: Contém implementações concretas de repositórios (ex.: `PlanoRepositoryMySQL` , `CobrancaRepositoryMongo`) e integrações com bancos de dados e filas.

Integrações

- **REST**: O `servico-planos-ativos` utiliza chamadas HTTP para consultar o `servico-gestao` (`GET /planos/:id`).
- **RabbitMQ**: O `servico-faturamento` publica eventos `cobranca.created` , que são consumidos pelo `servico-gestao` para registro em `cobranca_logs` .

- **Swagger**: O `api-gateway` oferece documentação interativa em `http://localhost:3003/api`.

Princípios SOLID Aplicados

Os princípios SOLID foram rigorosamente aplicados para garantir a qualidade do código:

- **Single Responsibility Principle (S)**: Cada classe possui uma única responsabilidade (ex.: `PlanoService` gerencia lógica de negócios, `PlanoRepositoryMySQL` cuida da persistência).
- **Open/Closed Principle (O)**: Interfaces como `IPlanoRepository` permitem extensões sem modificações no código existente.
- **Liskov Substitution Principle (L)**: Repositórios concretos substituem interfaces sem alterar o comportamento esperado.
- **Interface Segregation Principle (I)**: Interfaces são específicas, contendo apenas os métodos necessários.
- **Dependency Inversion Principle (D)**: Serviços dependem de abstrações, facilitando a injeção de dependências via NestJS.

Desafios Enfrentados

Durante o desenvolvimento da Fase 2 do projeto, foram enfrentados diversos desafios técnicos que exigiram análise detalhada, pesquisa aprofundada e ajustes precisos para garantir a funcionalidade do sistema. Cada obstáculo foi superado por meio de soluções técnicas cuidadosamente implementadas, resultando em um sistema robusto e integrado. A seguir, apresenta-se a relação dos principais desafios encontrados e as respectivas soluções adotadas:

Desafio	Solução
Erro `Cannot find name 'Inject'` (servico-planos-ativos)	Importar `Inject` do `@nestjs/common`.
Autenticação falha no MongoDB	Atualizar credenciais no `.env`.
Tabela `cobranca_logs` vazia (RabbitMQ)	Ajustar formatação de `dataVencimento`.
Roteamento incorreto no `api-gateway`	Testar com Postman e ajustar `ProxyService`.

- **Erro de Injeção de Dependência**: No serviço `servico-planos-ativos`, foi identificado o erro `Cannot find name 'Inject'`, que impedia a inicialização correta do módulo. Após análise, constatou-se que o problema decorria da ausência da importação do decorador `Inject`. A

solução consistiu em adicionar a linha `import { Inject } from '@nestjs/common'` ao arquivo `PlanoAtivoService`, permitindo a injeção de dependências pelo framework NestJS.

- **Falha de Autenticação no MongoDB**: Nos serviços `servico-faturamento` e `servico-planos-ativos`, ocorreu o erro `MongoServerError: bad auth : authentication failed`, indicando falha na conexão com o MongoDB Atlas. A resolução envolveu a revisão das credenciais configuradas no arquivo `.env`, atualizando o valor de `MONGODB_URI` com as informações corretas de usuário e senha, e subsequente validação da conexão no painel do MongoDB Atlas.
- **Tabela `cobranca_logs` Vazia**: No serviço `servico-gestao`, apesar de as mensagens do RabbitMQ serem recebidas, a tabela `cobranca_logs` no banco MySQL permanecia sem registros. A investigação revelou que o campo `dataVencimento` estava em um formato incompatível com o esperado pelo MySQL. A solução foi implementada no `CobrancaMessagingController`, ajustando a formatação da data para o padrão `YYYY-MM-DD`, o que permitiu o correto armazenamento dos logs.
- **Roteamento Incorreto no `api-gateway`**: Durante a configuração do serviço `api-gateway`, observou-se que as requisições eram roteadas de forma inadequada, gerando erros HTTP. Para solucionar o problema, foram realizados testes exaustivos utilizando o Postman, identificando falhas na lógica de encaminhamento do `ProxyService`. Ajustes foram aplicados ao serviço, otimizando o tratamento de requisições e garantindo o roteamento correto para os microsserviços.

Pré-requisitos

Para executar o projeto, os seguintes softwares e ferramentas são necessários:

- **Node.js**: Versão 18.x ou superior.
- **MySQL**: Banco de dados relacional para o `servico-gestao`.
- **MongoDB Atlas**: Banco de dados NoSQL para os serviços `servico-faturamento` e `servico-planos-ativos`.
- **RabbitMQ**: Servidor de mensageria, executado localmente ou via Docker.
- **Postman**: Ferramenta para testar os endpoints da API.
- **Docker** (opcional): Para executar o RabbitMQ em um contêiner.

Instalação

1. Clone o repositório do projeto ou extraia o arquivo compactado.
2. Para cada microsserviço (`servico-gestao`, `servico-faturamento`, `servico-planos-ativos`, `api-gateway`), instale as dependências:

```
cd brenda_desenvolvimento-de_sistemas-backend-fase-2/<servico>
npm install
```

Configuração

MySQL (servico-gestao)

1. Crie um banco de dados chamado `servico_gestao`.
2. Execute os seguintes comandos SQL para criar as tabelas necessárias:

```
CREATE TABLE planos (
    id INT PRIMARY KEY,
    nome VARCHAR(255) NOT NULL,
    valor DECIMAL(10,2) NOT NULL
);

CREATE TABLE cobranca_logs (
    cobranca_id INT PRIMARY KEY,
    cliente_id INT NOT NULL,
    valor DECIMAL(10,2) NOT NULL,
    status VARCHAR(50) NOT NULL,
    data_vencimento DATE NOT NULL
);
```

3. Configure as credenciais do MySQL em `servico-gestao/src/infraestrutura/database/mysqlconfig.ts`:

```
export const mysqlConfig = {
    host: 'localhost',
    user: 'seu_usuario',
    password: 'sua_senha',
    database: 'servico_gestao'
};
```

MongoDB Atlas (servico-faturamento e servico-planos-ativos)

1. Crie clusters no MongoDB Atlas para cada serviço.
2. Configure as credenciais nos arquivos `.env` de cada serviço:

- Para `servico-faturamento/.env` :

```
MONGODB_URI=mongodb+srv://<user>:<password>@cluster0.mongodb.net/servico_faturamento?retryWrites
RABBITMQ_URL=amqp://guest:guest@localhost:5672
```

- Para `servico-planos-ativos/.env` :

```
MONGODB_URI=mongodb+srv://<user>:<password>@cluster0.mongodb.net/servico_planos_ativos?retryWrit
GESTAO_BASE_URL=http://localhost:3000
```

RabbitMQ

1. Instale e inicie o RabbitMQ localmente ou use um contêiner Docker:

```
docker run -d --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3-management
```

2. Acesse o painel de gerenciamento em `http://localhost:15672` (usuário: `guest` , senha: `guest`).

API Gateway

1. Configure o arquivo `api-gateway/.env` :

```
GESTAO_BASE_URL=http://localhost:3000
FATURAMENTO_BASE_URL=http://localhost:3001
PLANOS_ATIVOS_BASE_URL=http://localhost:3002
```

Execução

1. Inicie cada microsserviço em terminais separados:

```
cd brenda_desenvolvimento-de_sistemas-backend-fase-2/servico-gestao
npm start

cd brenda_desenvolvimento-de_sistemas-backend-fase-2/servico-faturamento
npm start

cd brenda_desenvolvimento-de_sistemas-backend-fase-2/servico-planos-ativos
npm start

cd brenda_desenvolvimento-de_sistemas-backend-fase-2/api-gateway
npm start
```

2. Os serviços estarão disponíveis nas seguintes URLs:
    - `servico-gestao`： `http://localhost:3000`
    - `servico-faturamento`： `http://localhost:3001`
    - `servico-planos-ativos`： `http://localhost:3002`
    - `api-gateway`： `http://localhost:3003`
3. Acesse a documentação Swagger do `api-gateway` em `http://localhost:3003/api` .

Testes

Testes Unitários

Para executar os testes unitários de cada serviço:

```
cd brenda_desenvolvimento-de_sistemas-backend-fase-2/<servico>
npm run test
```

Testes de API

1. Importe a coleção Postman `Brenda_Desenvolvimento_de_Sistemas_backend_Fase-2.postman_collection.json` no Postman.
2. Execute as requisições nas pastas `Gestao` , `Faturamento` , `Planos Ativos` , e `API Gateway` para testar os endpoints.
3. Verifique os resultados nos bancos de dados:
    - MySQL: `SELECT * FROM planos;` e `SELECT * FROM cobranca_logs;` no banco `servico_gestao` .
    - MongoDB: Consulte as coleções `cobrancas` (banco `servico_faturamento`) e `planoativos` (banco `servico_planos_ativos`).

Conclusão

O projeto da Fase 2 foi concluído com sucesso, implementando uma arquitetura de microsserviços robusta e integrada. A utilização de *Clean Architecture*, princípios SOLID, e tecnologias como NestJS, MySQL, MongoDB, e RabbitMQ resultou em um sistema escalável e manutenível. Os desafios enfrentados durante o desenvolvimento foram superados com soluções técnicas precisas, demonstrando a capacidade de resolver problemas complexos em um ambiente de desenvolvimento backend.