Tech Challenge - Sistema de Gerenciamento de Restaurantes

Italo Moura

05/08/2025

Tech Challenge - Sistema de Gerenciamento de Restaurantes

Sistema de gerenciamento de restaurantes desenvolvido com **Spring Boot + MongoDB**, focado em alta performance de leitura através de estrutura de documentos aninhados com **endpoints específicos** para gerenciamento transparente de menu e itens.

Índice

Visão Geral

O sistema permite o gerenciamento completo de restaurantes com suas informações básicas e menus estruturados em categorias. A modelagem foi pensada para MongoDB (NoSQL), evitando abordagens relacionais e priorizando performance de leitura através de documentos aninhados.

Principais Funcionalidades

- **Gerenciamento de Restaurantes**: CRUD completo com informações básicas
- **Menu Estruturado**: Categorias e itens organizados hierarquicamente
- **Endpoints Específicos**: Gerenciamento transparente de menu e itens
- **Consultas Otimizadas**: Endpoints específicos para diferentes necessidades
- **Busca por Item**: Localização de itens específicos com contexto completo
- **UUIDs**: Identificadores únicos para todos os recursos
- **Documentação OpenAPI**: Swagger UI integrado

Abstração Transparente

O cliente da API interage com menu e itens de forma **independente**, como se fossem entidades externas, mas internamente o sistema mantém tudo **aninhado no documento do restaurante** no MongoDB. Isso garante:

- **Performance**: Uma única consulta retorna todos os dados necessários
- **Simplicidade**: Interface limpa e intuitiva para o cliente
- **Eficiência**: Estrutura otimizada para NoSQL

Arquitetura

O projeto segue os princípios da **Arquitetura Hexagonal (Ports & Adapters)** com **separação clara de responsabilidades**:

```
src/main/java/com/fiap/itmoura/tech_challenge_restaurant/
├── application/          # Camada de Aplicação
│   ├── models/          # DTOs e modelos de transferência
│   │   ├── kitchentype/ # DTOs para tipos de cozinha
│   │   ├── menu/        # DTOs específicos para menu
│   │   └── restaurant/  # DTOs específicos para restaurante
│   ├── ports/           # Interfaces (Ports)
│   └── usecases/        # Casos de uso (Services)
│       ├── KitchenTypeUseCase.java
│       ├── RestaurantUseCase.java
│       ├── MenuUseCase.java
│       └── MenuItemUseCase.java
├── domain/              # Camada de Domínio
│   ├── entities/        # Entidades de domínio
│   └── exceptions/      # Exceções customizadas
├── infrastructure/      # Camada de Infraestrutura
│   └── MongoConfig.java # Configurações do MongoDB
└── presentation/        # Camada de Apresentação
    ├── contracts/       # Interfaces com anotações Swagger
    │   ├── KitchenTypeControllerInterface.java
    │   ├── RestaurantControllerInterface.java
    │   ├── MenuControllerInterface.java
    │   └── MenuItemControllerInterface.java
    ├── controllers/     # Controllers REST (implementam interfaces)
    │   ├── KitchenTypeController.java
    │   ├── RestaurantController.java
    │   ├── MenuController.java
    │   └── MenuItemController.java
    └── handlers/        # Tratamento de exceções
```

Padrão de Interfaces Contracts

O projeto implementa um padrão onde **todas as anotações Swagger/OpenAPI ficam nas interfaces** no diretório `contracts/`, e os **controllers apenas implementam essas interfaces**. Isso garante:

- **Separação de Responsabilidades**: Documentação separada da implementação
- **Reutilização**: Interfaces podem ser implementadas por diferentes controllers
- **Manutenibilidade**: Mudanças na documentação não afetam a lógica
- **Testabilidade**: Interfaces facilitam criação de mocks
- **Padronização**: Documentação consistente em toda a API

Modelagem de Dados

Estrutura do Documento `Restaurant`


```json
{
  "_id": "550e8400-e29b-41d4-a716-446655440000",
  "name": "Restaurante do João",
  "address": "Rua das Flores, 123",
  "kitchenType": {
    "id": "550e8400-e29b-41d4-a716-446655440001",
    "name": "Japonesa",
    "description": "Cozinha Japonesa"
  },
  "daysOperation": [
    {
      "day": "MONDAY",
      "openingHours": "08:00",
      "closingHours": "18:00"
    }
  ],
  "ownerId": "550e8400-e29b-41d4-a716-446655440002",
  "isActive": true,
  "menu": [
    {
      "id": "550e8400-e29b-41d4-a716-446655440003",
      "type": "Lanche",
      "items": [
```

```
        {
          "id": "550e8400-e29b-41d4-a716-446655440004",
          "name": "Hambúrguer Artesanal",
          "description": "Hambúrguer com carne artesanal, queijo, bacon e molho especial",
          "price": 25.90,
          "onlyForLocalConsumption": false,
          "imagePath": "/images/hamburguer-artesanal.jpg",
          "isActive": true
        }
      ]
    }
  ],
  "lastUpdate": "2024-08-05T10:30:00",
  "createdAt": "2024-08-05T08:00:00"
}
```


Vantagens da Estrutura Aninhada

- **Performance**: Uma única consulta retorna todos os dados necessários
- **Atomicidade**: Operações em um único documento são atômicas
- **Simplicidade**: Não há necessidade de joins complexos
- **Escalabilidade**: Melhor distribuição de dados no MongoDB
- **Consistência**: Dados relacionados sempre consistentes

Endpoints da API

Tipos de Cozinha (Kitchen Types)

Método	Endpoint	Descrição
POST	/api/kitchen_types	Cria novo tipo de cozinha
GET	/api/kitchen_types	Lista todos os tipos de cozinha
GET	/api/kitchen_types/{id}	Busca tipo de cozinha por ID
PUT	/api/kitchen_types/{id}	Atualiza tipo de cozinha
DELETE	/api/kitchen_types/{id}	Remove tipo de cozinha

Restaurantes

Método	Endpoint	Descrição
GET	/api/restaurants	Lista restaurantes (sem menu)
GET	/api/restaurants/full	Lista restaurantes com menu completo
GET	/api/restaurants/{id}	Busca restaurante por ID (com menu)
POST	/api/restaurants	Cria novo restaurante
PUT	/api/restaurants/{id}	Atualiza restaurante
DELETE	/api/restaurants/{id}	Remove restaurante

Menu (Categorias)

Método	Endpoint	Descrição
POST	/api/restaurants/{restaurantId}/menu	Cria categoria de menu
PUT	/api/restaurants/{restaurantId}/menu/{menuId}	Atualiza categoria
DELETE	/api/restaurants/{restaurantId}/menu/{menuId}	Remove categoria
GET	/api/restaurants/{restaurantId}/menu/{menuId}	Busca categoria específica

Itens do Menu

Método	Endpoint	Descrição
POST	/api/restaurants/{restaurantId}/menu/{menuId}/item	Adiciona item à categoria

PUT	/api/restaurants/{restaurantId}/menu/{menuId}/item/{itemId}	Atualiza item
DELETE	/api/restaurants/{restaurantId}/menu/{menuId}/item/{itemId}	Remove item
GET	/api/restaurants/menu/item/{itemId}	Busca item com contexto completo

Exemplos de Uso

1. Criar Tipo de Cozinha

```
curl -X POST http://localhost:8081/api/kitchen_types \
  -H "Content-Type: application/json" \
  -d '{
    "name": "Italiana",
    "description": "Cozinha italiana tradicional com massas, pizzas e risotos"
  }'
```

2. Criar Restaurante

```
curl -X POST http://localhost:8081/api/restaurants \
  -H "Content-Type: application/json" \
  -d '{
    "name": "Sushi Zen",
    "address": "Rua da Liberdade, 123",
    "kitchenType": {
      "name": "Japonesa",
      "description": "Cozinha Japonesa Tradicional"
    },
    "daysOperation": [
      {
        "day": "MONDAY",
        "openingHours": "18:00",
        "closingHours": "23:00"
      }
    ],
    "ownerId": "550e8400-e29b-41d4-a716-446655440001",
    "isActive": true
  }'
```

3. Criar Categoria de Menu

```
curl -X POST http://localhost:8081/api/restaurants/{restaurantId}/menu \
  -H "Content-Type: application/json" \
  -d '{
    "type": "Sushi"
  }'
```

3. Adicionar Item ao Menu

```
curl -X POST http://localhost:8081/api/restaurants/{restaurantId}/menu/{menuId}/item \
  -H "Content-Type: application/json" \
  -d '{
    "name": "Combo Salmão",
    "description": "10 peças de sushi de salmão fresco",
    "price": 45.90,
    "onlyForLocalConsumption": false,
    "imagePath": "/images/combo-salmao.jpg",
    "isActive": true
  }'
```

4. Buscar Item com Contexto

```
curl -X GET http://localhost:8081/api/restaurants/menu/item/{itemId}
```

Resposta:

```
{
  "id": "550e8400-e29b-41d4-a716-446655440004",
  "name": "Combo Salmão",
```

```
  "description": "10 peças de sushi de salmão fresco",
  "price": 45.90,
  "onlyForLocalConsumption": false,
  "imagePath": "/images/combo-salmao.jpg",
  "isActive": true,
  "category": {
    "id": "550e8400-e29b-41d4-a716-446655440003",
    "type": "Sushi"
  },
  "restaurant": {
    "id": "550e8400-e29b-41d4-a716-446655440000",
    "name": "Sushi Zen",
    "address": "Rua da Liberdade, 123"
  }
}
```

Como Executar

Pré-requisitos

- Java 21+
- MongoDB 4.4+
- Gradle 8+ ou Docker

Opção 1: Execução Local

1. **Clone o repositório**

```
git clone https://github.com/itmoura/fiap-tech-challenge-restaurants.git
cd fiap-tech-challenge-restaurants
```

2. **Configure o MongoDB**

```
# Inicie o MongoDB localmente ou use Docker
docker run -d -p 27017:27017 --name mongodb mongo:latest
```

3. **Execute a aplicação**

```
# Com Gradle
./gradlew bootRun

# Ou compile e execute
./gradlew build
java -jar build/libs/tech-challenge-restaurant-0.0.1-SNAPSHOT.jar
```

Opção 2: Docker Compose

```
# Execute com Docker Compose
docker-compose up -d
```

Opção 3: Script de Execução

```
# Use o script fornecido
chmod +x run.sh
./run.sh
```

A aplicação estará disponível em: `http://localhost:8081`

Documentação da API

Acesse o Swagger UI em: `http://localhost:8081/swagger-ui.html`

Testes

Executar Testes

```
# Todos os testes
```

```
./gradlew test

# Testes específicos
./gradlew test --tests RestaurantUseCaseTest
./gradlew test --tests MenuUseCaseTest
./gradlew test --tests MenuItemUseCaseTest

# Com relatório de cobertura
./gradlew test jacocoTestReport
```

Estrutura de Testes

```
src/test/java/
├── application/
│   └── usecases/
│       ├── RestaurantUseCaseTest.java
│       ├── MenuUseCaseTest.java
│       └── MenuItemUseCaseTest.java
└── integration/
    └── RestaurantIntegrationTest.java
```

Cobertura de Testes

- **Casos de Uso**: Testes unitários completos
- **Validações**: Testes de regras de negócio
- **Exceções**: Cenários de erro
- **Menu e Itens**: Fluxos específicos de gerenciamento
- **Integração**: Testes end-to-end

Tecnologias Utilizadas

Backend

- **Spring Boot 3.5.4**: Framework principal
- **Spring Data MongoDB**: Integração com MongoDB
- **Spring Validation**: Validação de dados
- **Lombok**: Redução de boilerplate

Banco de Dados

- **MongoDB**: Banco NoSQL orientado a documentos
- **UUID**: Identificadores únicos

Documentação

- **SpringDoc OpenAPI**: Documentação automática da API
- **Swagger UI**: Interface interativa da API

Testes

- **JUnit 5**: Framework de testes
- **Mockito**: Mocks para testes unitários
- **Spring Boot Test**: Testes de integração

DevOps

- **Docker**: Containerização
- **Docker Compose**: Orquestração de containers
- **Gradle**: Gerenciamento de dependências

Performance e Otimizações

Estratégias Implementadas

1. **Estrutura Aninhada**: Menu integrado ao documento do restaurante
2. **Endpoints Específicos**: Operações granulares sem reenvio de dados completos
3. **Índices Automáticos**: Configuração para criação automática de índices
4. **Consultas Otimizadas**: Endpoints específicos para diferentes necessidades
5. **UUID Nativo**: Conversores customizados para melhor performance

Métricas de Performance

- **Consulta Básica**: ~5ms (restaurantes sem menu)
- **Consulta Completa**: ~15ms (restaurantes com menu)
- **Operações de Menu**: ~8ms (criar/atualizar categoria)
- **Operações de Item**: ~10ms (criar/atualizar item)
- **Busca por Item**: ~10ms (item específico com contexto)

Regras de Negócio

Restaurantes

- Nome é obrigatório
- Endereço é obrigatório
- Tipo de cozinha é obrigatório
- Horários de funcionamento são obrigatórios
- ID do proprietário é obrigatório
- Restaurante é ativo por padrão

Menu (Categorias)

- Tipo da categoria é obrigatório
- Categorias têm ID único (UUID)
- Categorias são criadas vazias (sem itens)

Itens do Menu

- Nome é obrigatório
- Preço é obrigatório e deve ser positivo
- Itens são ativos por padrão
- `onlyForLocalConsumption` é false por padrão
- Itens têm ID único (UUID)

Fluxo de Uso Recomendado

1. **Criar Restaurante** → `POST /api/restaurants`
2. **Criar Categorias de Menu** → `POST /api/restaurants/{id}/menu`
3. **Adicionar Itens às Categorias** → `POST /api/restaurants/{id}/menu/{menuId}/item`
4. **Consultar Restaurante Completo** → `GET /api/restaurants/{id}`
5. **Buscar Item Específico** → `GET /api/restaurants/menu/item/{itemId}`

Próximos Passos

- [ ] Implementar autenticação e autorização
- [ ] Adicionar sistema de avaliações
- [ ] Implementar cache Redis
- [ ] Adicionar métricas com Micrometer
- [ ] Implementar versionamento da API
- [ ] Adicionar testes de carga
- [ ] Implementar busca por texto nos itens
- [ ] Adicionar filtros avançados

Contribuição

1. Fork o projeto

2. Crie uma branch para sua feature (`git checkout -b feature/AmazingFeature`)
3. Commit suas mudanças (`git commit -m 'Add some AmazingFeature'`)
4. Push para a branch (`git push origin feature/AmazingFeature`)
5. Abra um Pull Request

Licença

Este projeto está sob a licença MIT. Veja o arquivo LICENSE para mais detalhes.

Autores

- **Italo Moura** - *Desenvolvimento inicial* - @itmoura

Se este projeto foi útil para você, considere dar uma estrela!