Guia de Operação do Sistema de Gestão de Reservas

1. Introdução

Este documento serve como guia técnico completo para a execução, teste e validação do sistema de gestão de reservas de restaurante desenvolvido no âmbito da unidade curricular de Desenvolvimento/Operação de Software. O sistema implementa uma API REST com funcionalidades completas de CRUD para reservas, incluindo conteinerização, CI/CD automatizado e análise de qualidade de código.

2. Estrutura de Diretórios

```
project-root/
├── src/RestaurantReservations.API/          # Código fonte da API
├── tests/RestaurantReservations.UnitTests/  # Testes unitários
├── helm/restaurant-api/                     # Configuração Kubernetes
├── docker-compose.yml                       # Orquestração local
├── Jenkinsfile                              # Pipeline CI/CD
└── sonar-project.properties                 # Configuração SonarQube
```

3. Execução do Sistema

3.1. Ambiente de Desenvolvimento Local

3.1.1. Execução com Docker Compose

Na raiz do projeto:

Parar serviços existentes:

- docker-compose down

Construir e iniciar containers:

- docker-compose up --build -d

URLs de acesso:

- Swagger UI:

- *http://localhost:5000/swagger*

- Health Check:

- *http://localhost:5000/api/reservations/status*

Serviços iniciados:

- restaurant-reservations-api (Porta 5000)
- restaurant-reservations-sqlserver (Porta 1433)

3.1.2. Execução nativa .NET

- cd src/RestaurantReservations.API
- dotnet restore
- dotnet run

URL de acesso:

- *http://localhost:5188/swagger*

3.2. Verificação do Estado dos Serviços

Verificar containers em execução:

- docker-compose ps

Verificar logs da aplicação:

- docker-compose logs api

4. Execução de Testes Unitários

4.1. Execução Completa com Cobertura

- cd tests/RestaurantReservations.UnitTests
- dotnet test --collect:"XPlat Code Coverage" --verbosity normal

4.2. Execução Específica

Executar apenas um teste:

- dotnet test --filter "GetAllReservationsAsync_Should_Return_All_Reservations"

Executar testes de uma classe específica:

- dotnet test --filter "FullyQualifiedName~ReservationServiceTests"

Executar sem relatório de cobertura:

- dotnet test

5. Gestão de Containers Docker

5.1. Comandos Essenciais

Na raiz do projeto:

Iniciar todos os serviços:

- docker-compose up -d

Parar todos os serviços:

- docker-compose down

Reconstruir e reiniciar:

- docker-compose up --build -d

Ver logs em tempo real:

- docker-compose logs -f api
- docker-compose logs -f sqlserver

Ver estado dos containers:

- docker-compose ps

6. Validação Funcional da API

6.1. Testes Manuais via Swagger

Acessar http://localhost:5000/swagger permite testar todos os endpoints:

Endpoints disponíveis:

- GET /api/reservations - Listar todas reservas
- GET /api/reservations/{id} - Obter reserva específica
- POST /api/reservations - Criar nova reserva
- PUT /api/reservations/{id} - Atualizar reserva
- DELETE /api/reservations/{id} - Cancelar reserva
- GET /api/reservations?date={date} - Filtrar por data

6.2. Validação de Funcionalidades Críticas

6.2.1. Teste de Conflito de Horário:

Criar primeira reserva:

```
'{
   "customerName": "Cliente 1",
   "reservationDate": "2026-06-30",
   "reservationTime": "19:30:00",
   "tableNumber": 5,
   "numberOfPeople": 4
 }'
```

Tentar criar reserva conflitante (deve retornar 409):

```
'{
   "customerName": "Cliente 2",
   "reservationDate": "2026-06-30",
   "reservationTime": "19:30:00",
   "tableNumber": 5,
   "numberOfPeople": 2
 }'
```

6.2.2. Teste de Validação de Data Futura

Tentar criar reserva no passado (deve falhar):

```
'{
   "customerName": "Cliente Teste",
   "reservationDate": "2026-01-22",
   "reservationTime": "12:00:00",
   "tableNumber": 1,
   "numberOfPeople": 2
 }'
```