Relatório Final - Projeto DevOps

CTeSP em Desenvolvimento Ágil de Software - ERM

Desenvolvimento / Operação de Software

2024/25 - 1º Semestre

Grupo C

Asafe Klain - 1231622

João Andrade - 1231570

Júlio Sousa - 1231571

Fardeen Munaf - 1231567

Miguel Ribeiro -1232159

Índice

1. Introdução 3
2. Tecnologias Utilizadas 3
3. Planeamento e Distribuição de Tarefas 5
4. Implementação do Projeto 6
   4.1. Desenvolvimento da Web API 6
   4.2. Base de Dados 11
   4.3. Virtualização e Containerização 13
   4.4. Testes Unitários 13
   4.5. Pipeline Automatizado (CI/CD) 14
5. Desafios e Melhorias Futuras 14
   Dificuldades Encontradas 14
   Possíveis Melhorias 15
6. Conclusão 15

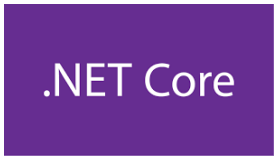
1. Introdução

Este documento apresenta o desenvolvimento do projeto DevOps para a gestão de reservas de mesas em restaurantes. O objetivo foi criar uma Web API REST utilizando .Net Core, SQL Server, Docker e integração com ferramentas de CI/CD como Jenkins e SonarQube.

Durante o projeto, foram aplicadas práticas de automação, controle de qualidade e containerização. O presente relatório descreve o processo de desenvolvimento, dificuldades encontradas e melhorias sugeridas.

2. Tecnologias Utilizadas

Para o desenvolvimento deste projeto, utilizamos diversas tecnologias que garantiram a eficiência, escalabilidade e qualidade do sistema:



.Net Core: Framework utilizado para a construção da Web API, garantindo robustez e suporte para múltiplas plataformas.

Docker: Ferramenta de containerização que facilitou a implantação e gerenciamento do ambiente de execução da API.

SQL Server: Banco de dados relacional utilizado para armazenar informações das reservas de forma segura e eficiente.

Vagrant: Utilizado para a criação de máquinas virtuais para o ambiente de desenvolvimento e testes.

Jenkins: Ferramenta de automação para integração contínua (CI) e entrega contínua (CD), garantindo um fluxo de desenvolvimento ágil e automatizado.

SonarQube: Utilizado para análise de qualidade do código e identificação de possíveis vulnerabilidades.

Swagger: Ferramenta para documentação e testes interativos dos endpoints da API.

xUnit: Framework utilizado para a criação e execução de testes unitários, garantindo a confiabilidade do sistema.

3. Planeamento e Distribuição de Tarefas

Para organização do trabalho, a equipe dividiu as responsabilidades da seguinte forma:

- **Teste unitarios: João/Fardeen**
- **Integração com docker: Miguel**
- **Base de dados, vagrant, pipeline e jenkins: Júlio/Miguel**
- **Desenvolvimento de API: Fardeen**
- **Documentação / Relatório: Asafe**

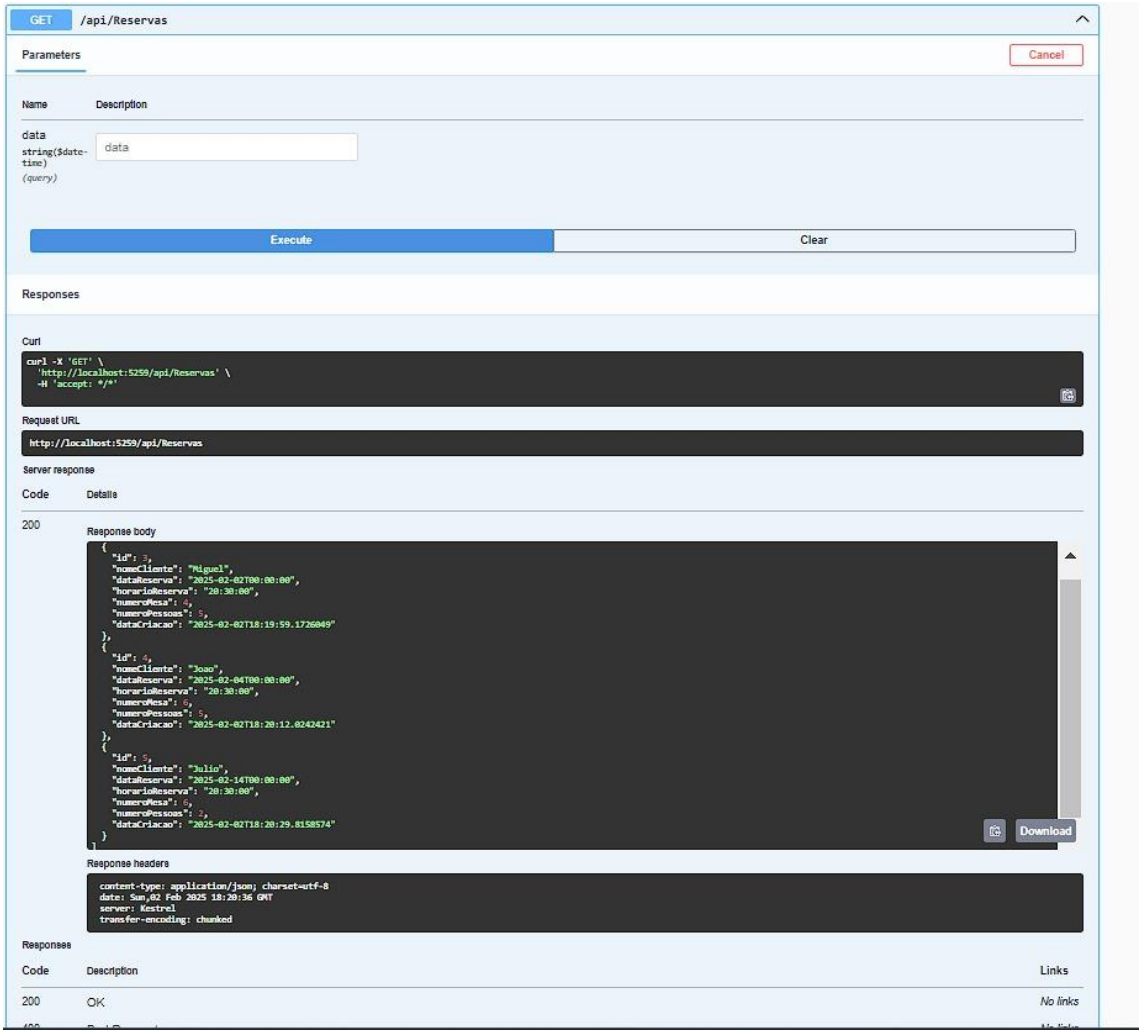
As tarefas foram geridas através do repositório GitHub, utilizando a branch dev para desenvolvimento e a branch main como versão final para avaliação.

4. Implementação do Projeto

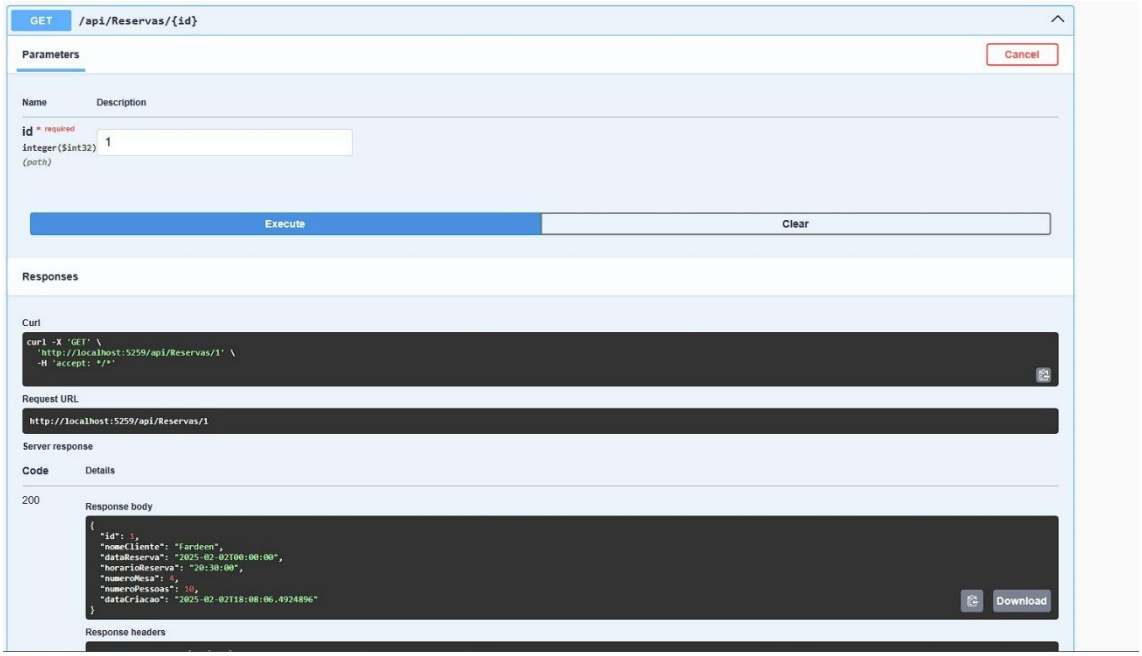
4.1. Desenvolvimento da Web API

A API foi desenvolvida em .Net Core seguindo o padrão REST. Os endpoints implementados foram:

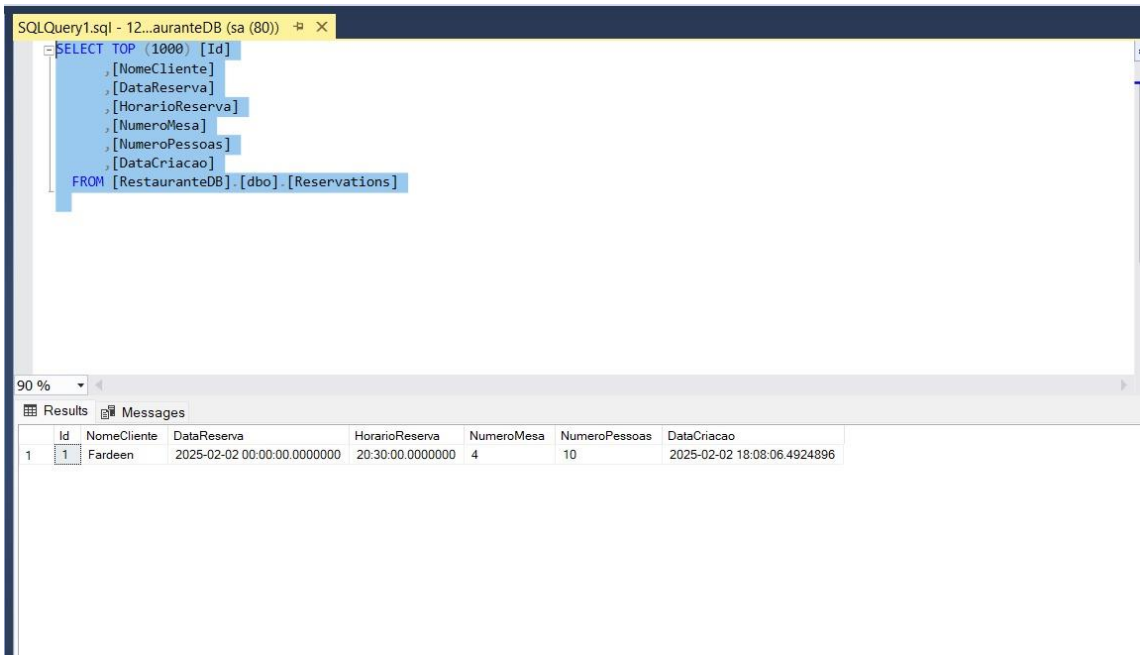
- GET /reservations - Listar todas as reservas.



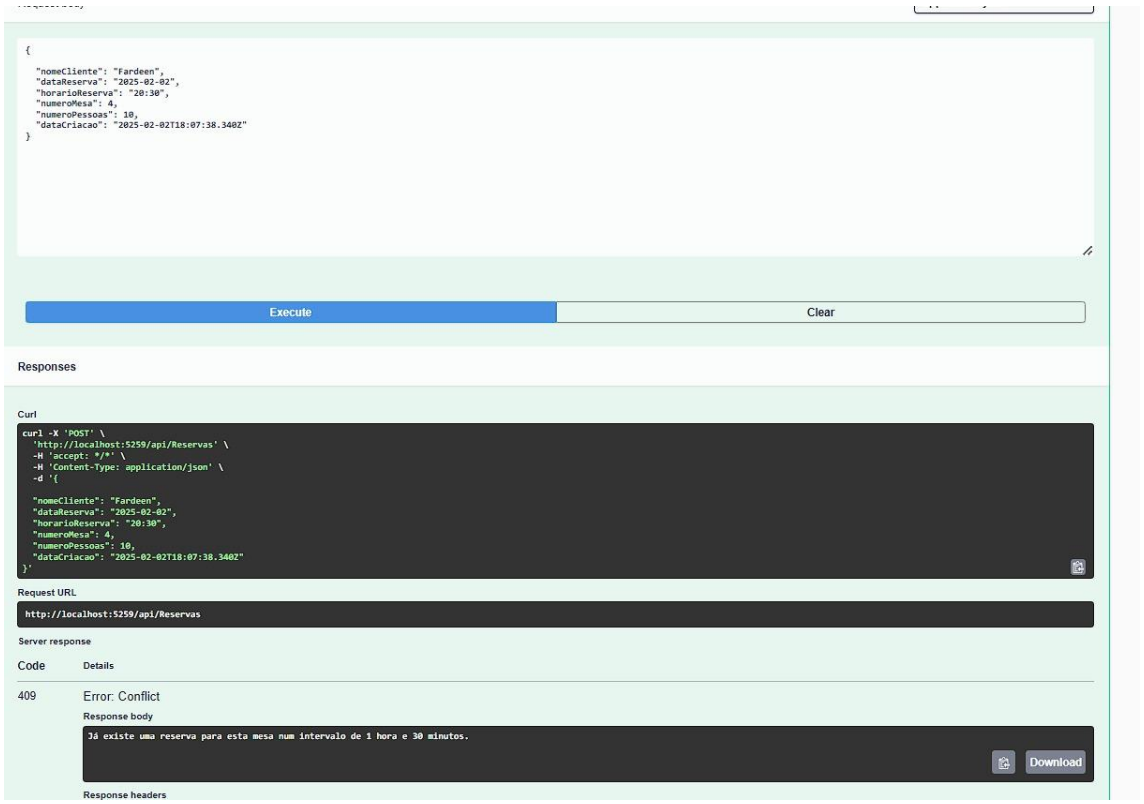
- GET /reservations/{id} - Retornar detalhes de uma reserva.



- POST /reservations - Criar uma nova reserva.

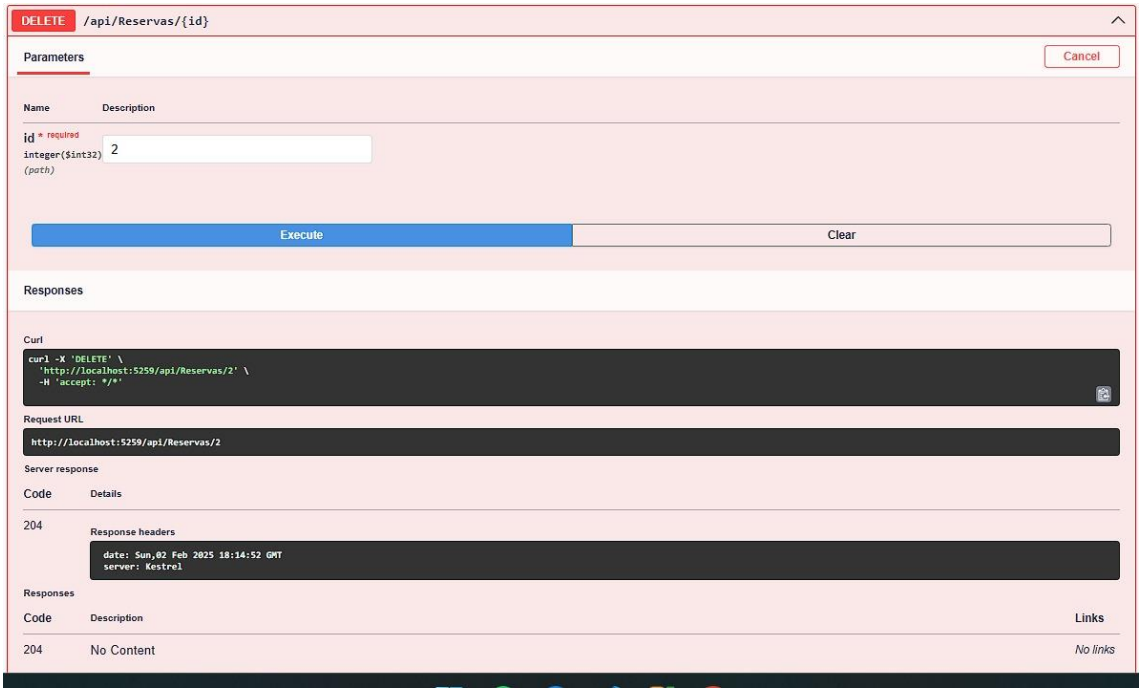


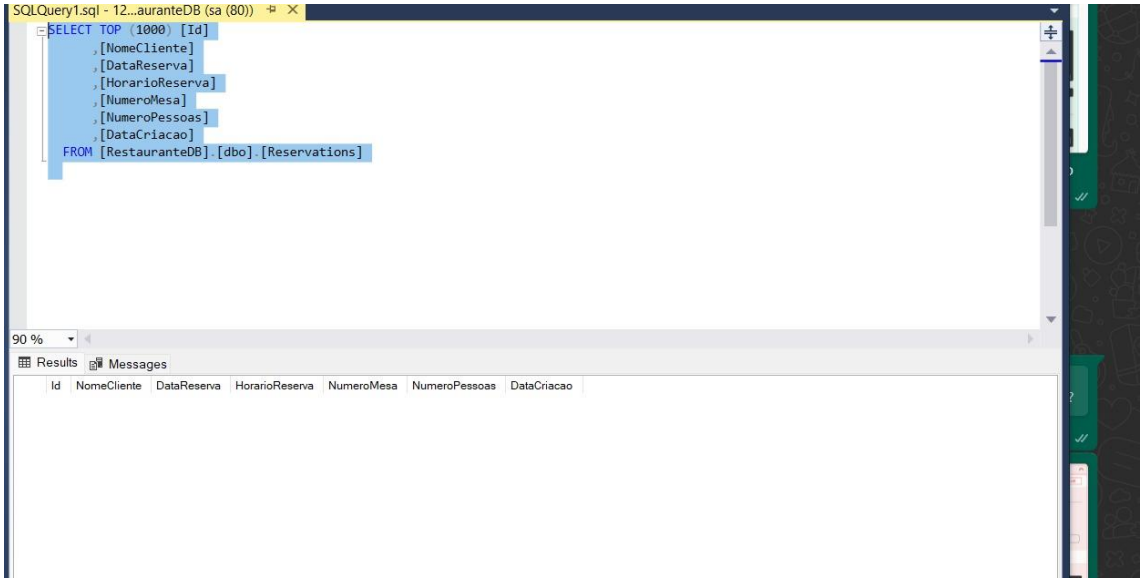
Conflito quando reservamos a mesma mesa para o intervalo de 1h e 30min:



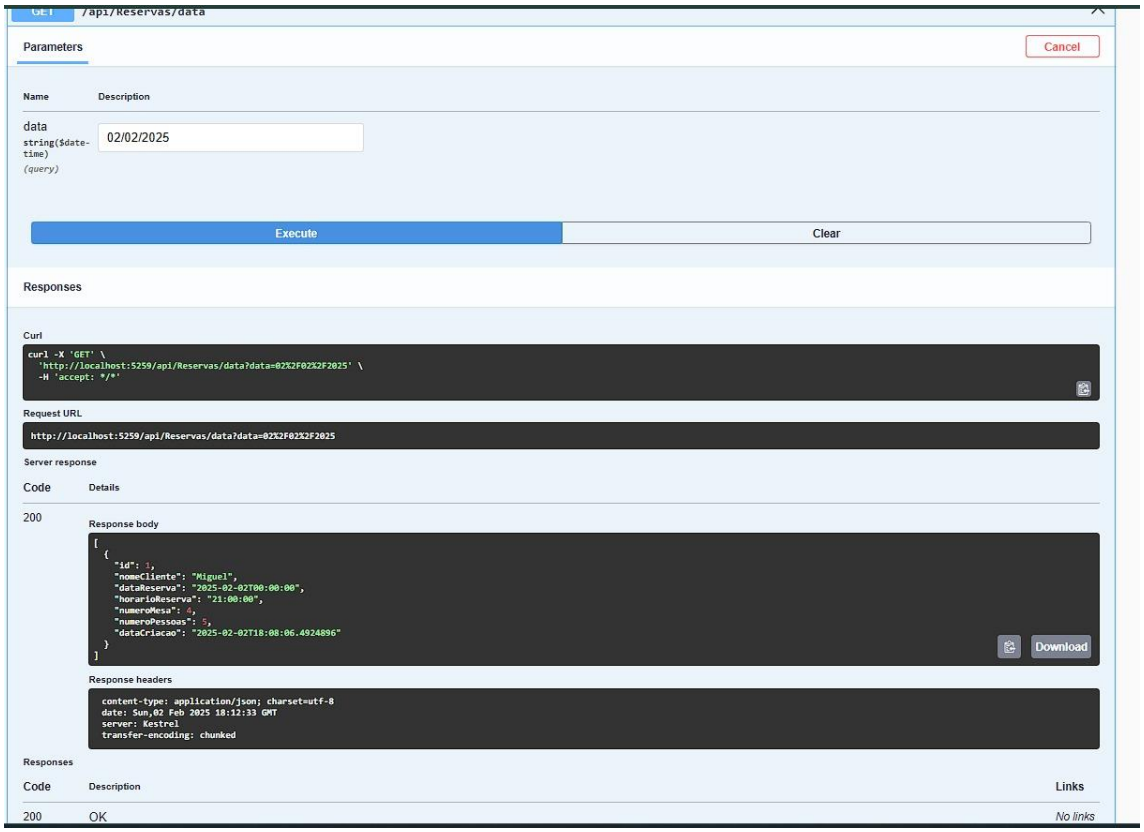
- PUT /reservations/{id} - Atualizar uma reserva existente.

- DELETE /reservations/{id} - Cancelar uma reserva.

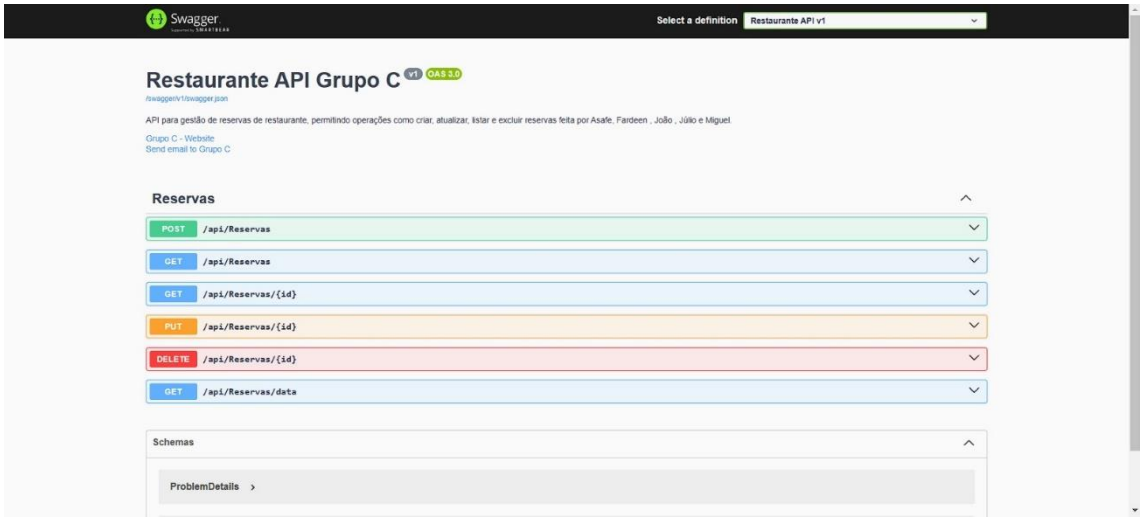




- GET /reservations?date={date} - Listar reservas de uma data específica.



Swagger foi utilizado para documentação da API, facilitando os testes e interação com os endpoints.



A API foi projetada para suportar até 30 mesas, com um máximo de 20 pessoas por mesa. Implementamos uma função para impedir que uma mesma mesa seja reservada dentro de um intervalo de 90 minutos, garantindo que não haja conflitos de horário.

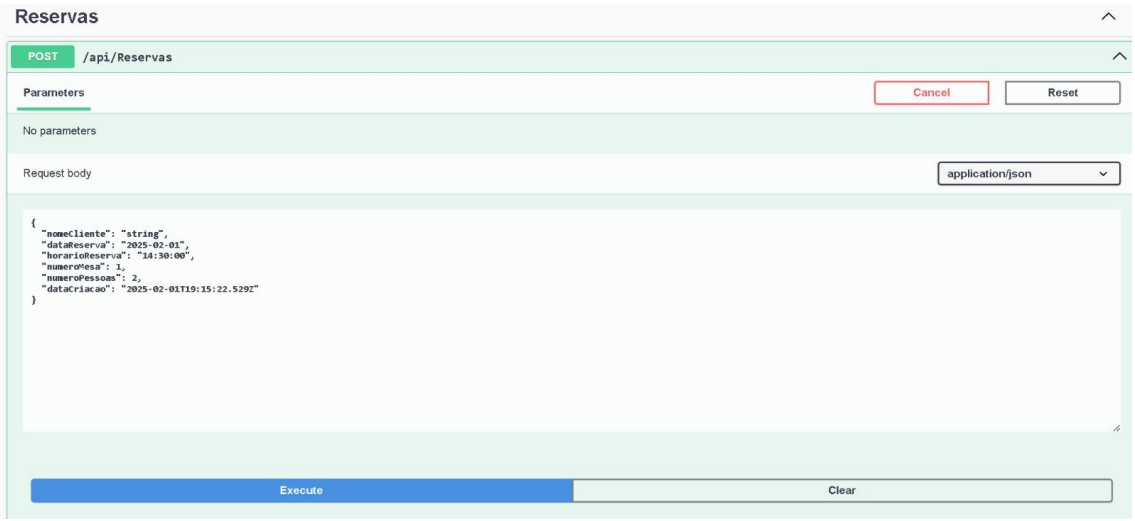
4.2. Base de Dados

A base de dados foi implementada utilizando SQL Server, com a estrutura da tabela Reservations. O modelo foi gerado automaticamente via **Code First**.

Para criar uma migraçao: dotnet ef migrations add RestauranteDB

Para criar a base de dados e tabelas: dotnet ef database update

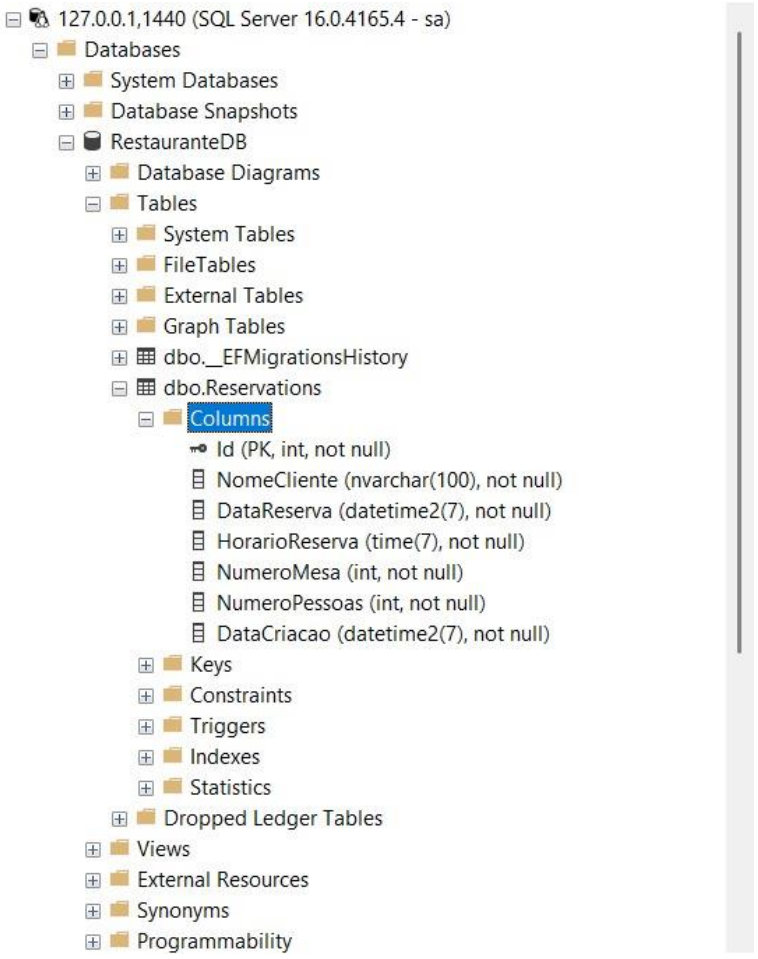
Correr a api: dotnet run




```
select * from Reservations
```

Results | Messages

Id	NomeCliente	DataReserva	HorarioReserva	NumeroMesa	NumeroPessoas	DataCriacao
1	string	2025-02-01 00:00:00.0000000	14:30:00.0000000	1	2	2025-02-01 19:35:27.3999133



4.3. Virtualização e Containerização

- Configuração de uma **VM com Vagrant** para hospedar o SQL Server.
- Criação de um **Dockerfile** para containerização da API.
- Integração da API com a base de dados na VM ou no container Docker.
- Observação: Não foi feita a configuração da máquina no Vagrant para desligar o firewall ou outras configurações de segurança.

4.4. Testes Unitários

- Foram desenvolvidos testes unitários utilizando **xUnit** para verificar funcionalidades como:
    - Criação de reservas.
    - Deteção de conflitos.
    - Listagem de reservas.

- **Dificuldade encontrada:** Inicialmente, os testes foram criados na pasta errada, o que impediu a execução correta. Após a correção, os testes passaram a funcionar corretamente.

- **Coverage:** Tivemos dificuldade em configurar corretamente a cobertura de testes, então geramos um **HTML com os dados do coverage** para referência.

4.5. Pipeline Automatizado (CI/CD)

- Configuração do **Jenkins** para:
    - Build e execução de testes unitários.
    - Análise de qualidade de código com **SonarQube**.
    - Empacotamento da API e deployment em container Docker.

- **Dificuldade encontrada:** Inicialmente tivemos problemas de conexão no **Vagrant**, o que dificultou a comunicação com a API. O problema foi resolvido ajustando configurações da rede.

5. Desafios e Melhorias Futuras

Dificuldades Encontradas

- Configuração do **coverage**, resolvida com a geração de um HTML.
- Testes inicialmente criados na **pasta errada**, o que impediu a execução.
- Problemas de **conexão com Vagrant**, corrigidos ao ajustar configuração de rede.
- Falta de configuração da máquina no Vagrant para desligar firewall e outras otimizações.

Possíveis Melhorias

- Melhorar a gestão de cobertura de testes.
- Implementar novas funcionalidades, como gerenciamento de usuários.
- Melhor otimização da pipeline para reduzir o tempo de execução.
- Revisar configuração do Vagrant para melhorias de segurança e desempenho.

6. Conclusão

Este projeto permitiu aplicar conceitos fundamentais de DevOps, como CI/CD, containerização e automação de testes. Apesar dos desafios, conseguimos implementar uma solução funcional, validada por testes unitários e integrada ao Jenkins e SonarQube. A experiência adquirida contribuirá para futuros projetos profissionais.