

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

.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.

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** /api/Reservas
- Parameters:** A table with one parameter:

Name	Description
data	string(\$date-time) (query)
- Execute:** A blue button to execute the request.
- Responses:** A section showing the request and response details.
 - Request:** curl -X 'GET' -H 'accept: */*' 'http://localhost:5259/api/Reservas'
 - Request URL:** http://localhost:5259/api/Reservas
 - Server response:** Code 200, Details: Response body.
 - Response body:** A JSON array of reservation objects. The first object is:

```
{  "id": 3,  "nomeCliente": "Miguel",  "dataReserva": "2025-02-02T00:00:00",  "horarioReserva": "20:30:00",  "numeroMesa": 0,  "numeroPessoas": 5,  "dataCriacao": "2025-02-02T18:19:59.176049"}
```
 - Response headers:**

```
content-type: application/json; charset=utf-8
date: Sat, 02 Feb 2025 18:29:36 GMT
server: Kestrel
transfer-encoding: chunked
```
- Responses table:**

Code	Description	Links
200	OK	No links

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

GET /api/Reservas/{id}

Parameters

Name	Description
id * required	Integer(\$int32) (path)

1

Execute **Clear**

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:5259/api/Reservas/1' \
  -H 'accept: */*' \
```

Request URL

http://localhost:5259/api/Reservas/1

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": 1, "nomeCliente": "Fardeen", "dataReserva": "2025-02-02T00:00:00", "horarioReserva": "20:30:00", "numeroMesa": 4, "numeroPessoas": 10, "dataCriacao": "2025-02-02T18:08:06.4924896" }</pre> <p>Response headers</p> <pre>content-type: application/json; charset=utf-8 date: Sun, 02 Feb 2025 18:08:06 GMT location: http://localhost:5259/api/Reservas/1 server: Kestrel transfer-encoding: chunked</pre>

- POST /reservations - Criar uma nova reserva.

Curl

```
curl -X 'POST' \
  'http://localhost:5259/api/Reservas' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "nomeCliente": "Fardeen",
    "dataReserva": "2025-02-02",
    "horarioReserva": "20:30",
    "numeroMesa": 4,
    "numeroPessoas": 10,
    "dataCriacao": "2025-02-02T18:07:38.348Z"
  }'
```

Request URL

http://localhost:5259/api/Reservas

Server response

Code	Details
201	<p>Response body</p> <pre>{ "id": 1, "nomeCliente": "Fardeen", "dataReserva": "2025-02-02T00:00:00", "horarioReserva": "20:30:00", "numeroMesa": 4, "numeroPessoas": 10, "dataCriacao": "2025-02-02T18:08:06.4924896+00:00" }</pre> <p>Response headers</p> <pre>content-type: application/json; charset=utf-8 date: Sun, 02 Feb 2025 18:08:06 GMT location: http://localhost:5259/api/Reservas/1 server: Kestrel transfer-encoding: chunked</pre>

Responses

Code	Description	Links
201	Created	No links

SQLQuery1.sql - 12...auranteDB (sa (80))

```

SELECT TOP (1000) [Id]
, [NomeCliente]
, [DataReserva]
, [HorarioReserva]
, [NumeroMesa]
, [NumeroPessoas]
, [DataCriacao]
FROM [RestauranteDB].[dbo].[Reservations]

```

90 %

Results Messages

	Id	NomeCliente	DataReserva	HorarioReserva	NumeroMesa	NumeroPessoas	DataCriacao
1	1	Fardeen	2025-02-02 00:00:00.0000000	20:30:00.0000000	4	10	2025-02-02 18:08:06.4924896

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

```

{
  "nomeCliente": "Fardeen",
  "dataReserva": "2025-02-02",
  "horarioReserva": "20:30",
  "numeroMesa": 4,
  "numeroPessoas": 10,
  "dataCriacao": "2025-02-02T18:07:38.340Z"
}

```

Execute Clear

Responses

Curl

```

curl -X 'POST' \
  'http://localhost:5259/api/Reservas' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "nomeCliente": "Fardeen",
    "dataReserva": "2025-02-02",
    "horarioReserva": "20:30",
    "numeroMesa": 4,
    "numeroPessoas": 10,
    "dataCriacao": "2025-02-02T18:07:38.340Z"
  }'

```

Request URL

http://localhost:5259/api/Reservas

Server response

Code Details

409 Error: Conflict

Response body

Já existe uma reserva para esta mesa num intervalo de 1 hora e 30 minutos.

Download

Response headers

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

Responses

Curl

```
curl -X 'PUT' \
  http://localhost:5259/api/Reservas/1' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "id": 1,
    "nomeCliente": "Miguel",
    "dataReserva": "2025-02-02",
    "horarioReserva": "21:00",
    "numeroMesa": 4,
    "numeroPessoas": 5,
    "dataCriacao": "2025-02-02T18:10:07.121Z"
  }'
```

Request URL

http://localhost:5259/api/Reservas/1

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": 1, "nomeCliente": "Miguel", "dataReserva": "2025-02-02T00:00:00", "horarioReserva": "21:00:00", "numeroMesa": 4, "numeroPessoas": 5, "dataCriacao": "2025-02-02T18:06:46.4924896" }</pre> <p>Response headers</p>

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

DELETE /api/Reservas/{id}

Parameters

Name Description

id * required
integer(\$int32) 2
(path)

Execute Clear

Responses

Curl

```
curl -X 'DELETE' \
  http://localhost:5259/api/Reservas/2' \
  -H 'accept: */*' \
```

Request URL

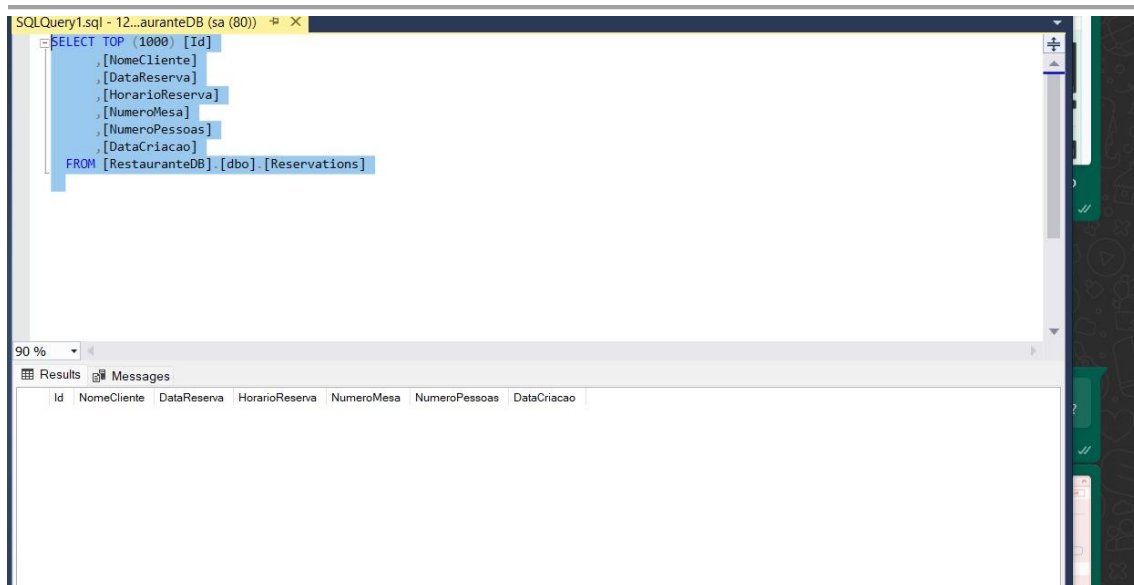
http://localhost:5259/api/Reservas/2

Server response

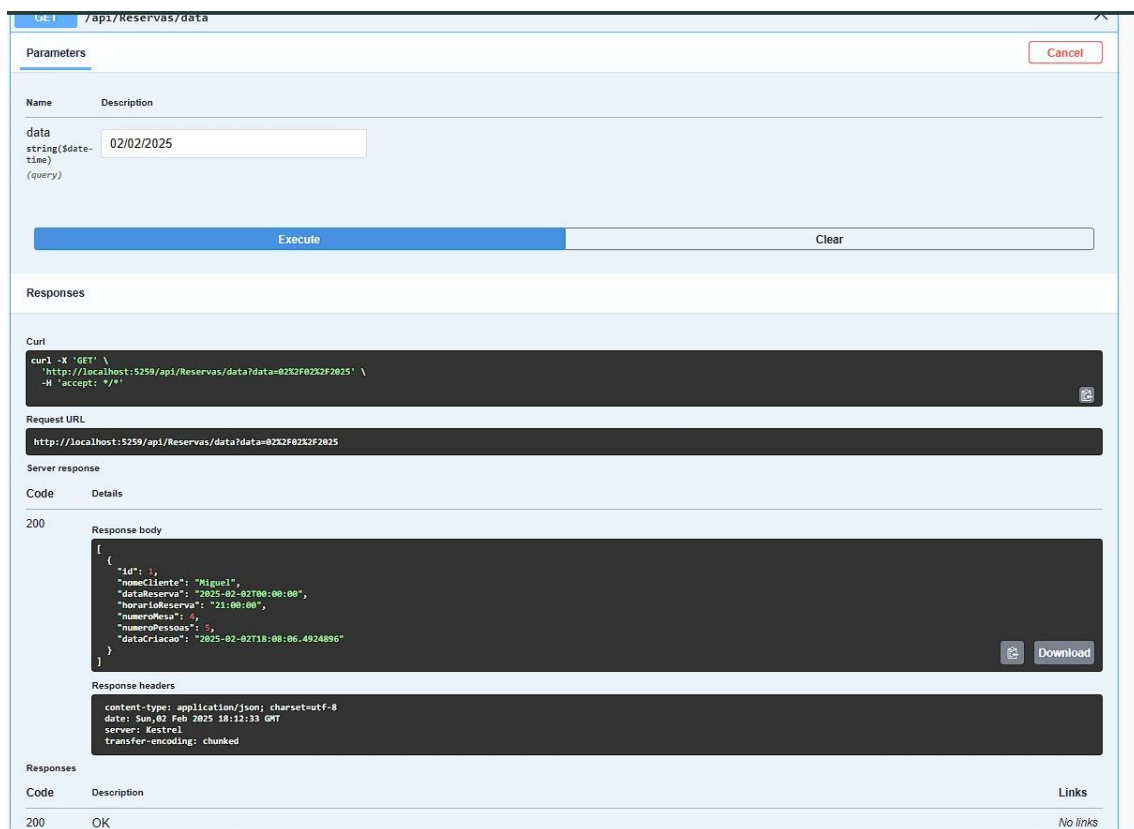
Code	Details
204	<p>Response headers</p> <pre>date: Sun, 02 Feb 2025 18:14:52 GMT server: Kestrel</pre>

Responses

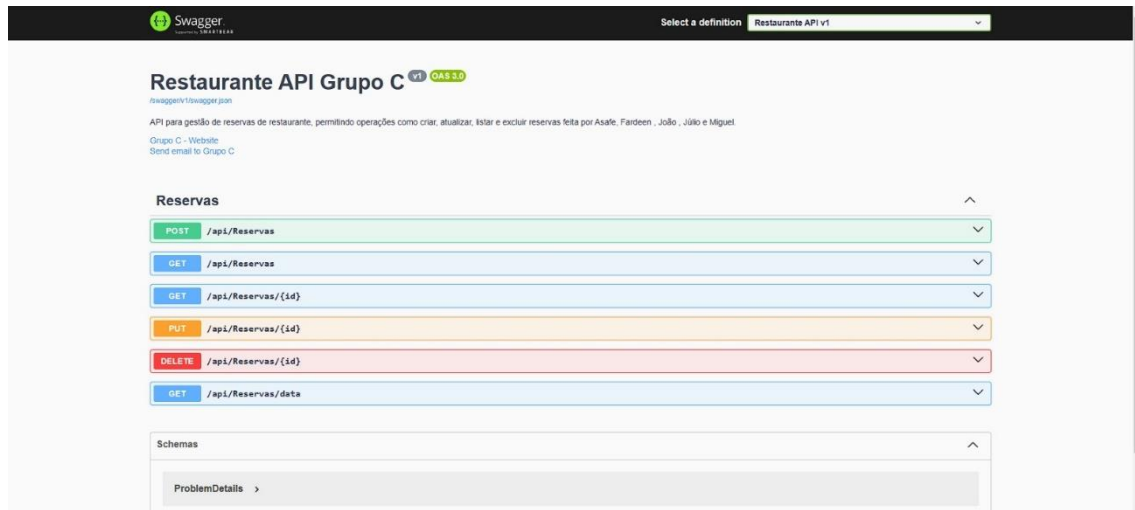
Code	Description	Links
204	No Content	No links



- 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 migração: `dotnet ef migrations add RestauranteDB`

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

Correr a api: `dotnet run`

Reservas

POST /api/Reservas

Parameters Cancel Reset

No parameters

Request body application/json

```
{
  "nomeCliente": "string",
  "dataReserva": "2025-02-01",
  "horarioReserva": "14:30:00",
  "numeroMesa": 1,
  "numeroPessoas": 2,
  "dataCriacao": "2025-02-01T19:15:22.529Z"
}
```

Execute Clear

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

- 127.0.0.1,1440 (SQL Server 16.0.4165.4 - sa)
 - Databases
 - System Databases
 - Database Snapshots
 - RestauranteDB
 - Database Diagrams
 - Tables
 - System Tables
 - FileTables
 - External Tables
 - Graph Tables
 - dbo.__EFMigrationsHistory
 - dbo.Reservations
 - Columns
 - Id (PK, int, not null)
 - NomeCliente (nvarchar(100), not null)
 - DataReserva (datetime2(7), not null)
 - HorarioReserva (time(7), not null)
 - NumeroMesa (int, not null)
 - NumeroPessoas (int, not null)
 - DataCriacao (datetime2(7), not null)
 - Keys
 - Constraints
 - Triggers
 - Indexes
 - Statistics
 - Dropped Ledger Tables
 - Views
 - External Resources
 - Synonyms
 - Programmability

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.
 - Detecção 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.