Sistema de locadora de veículos utilizando C#, ASP.NET Core, Entity Framework, SQL Express e Swagger

Anna Carla Teixeira da Silva

[1] Análise e Desenvolvimento de Sistemas – Pontificia Universidade Católica de Minas Gerais
Belo Horizonte, Minas Gerais.

anna.carla@sga.pucminas.br

Abstract. *Este relatório descreve o desenvolvimento de um sistema de locadora de veículos, apresentando sua arquitetura, implementação técnica, banco de dados, testes e validação, resultados e conclusões.*

1. Introdução

1.1. Contextualização do Projeto e sua Importância

O propósito deste relatório é oferecer uma análise detalhada do projeto concebido para estimular a assimilação de conceitos de backend pelos alunos, empregando tecnologias contemporâneas e funcionalidades práticas. O projeto aborda a criação de uma plataforma de locação de veículos, utilizando recursos como CSharp, ASP.NET Core 8.0, Entity Framework, SQL e Swagger para testes. Essa iniciativa busca não apenas familiarizar os alunos com tais ferramentas, mas também proporcionar-lhes uma compreensão mais aprofundada por meio da sua aplicação em um contexto realístico.

1.2. Objetivos do Sistema de Locadora de Veículos

O projeto visa desenvolver uma API utilizando ASP.NET Core para um sistema de locadora de veículos, permitindo que os clientes efetuem reservas de veículos de forma prática e eficiente.

1.3. Principais Funcionalidades

As funcionalidades centrais do sistema estão relacionadas aos seguintes aspectos:

Clientes:

- Cadastro e gerenciamento de clientes.
- Pesquisa de todos os clientes ou de um cliente específico.
- Pesquisa do clientes e suas reservas.

Veículos:

- Cadastro e gerenciamento de veículos.
- Pesquisa de todos os veículos ou de um veículo específico.
- Pesquisa de todos os veículos disponíveis.

Locações:

- Cadastro e gerenciamento de locações.
- Pesquisa de uma locação específica.
- Pesquisa de todas as locações disponíveis.

1.4. Tecnologias Utilizadas

- **IDE:** Visual Studio Community
- **Repositório:** Github
- **Versionamento:** Git
- **Ambiente SQL:** SQL Server Management Studio
- **Teste:** Swagger

2. Requisitos Funcionais

Table 1. Requisitos Funcionais do Sistema de Locadora de Veículos

ID	Descrição	Prioridade
RF01	O sistema deve permitir o cadastro de novos clientes, incluindo informações como nome, CPF, Telefone e Endereço.	Alta
RF02	O sistema deve permitir a edição e atualização dos dados dos clientes cadastrados.	Alta
RF03	O sistema deve permitir a exclusão de clientes cadastrados.	Alta
RF04	O sistema deve permitir a pesquisa de clientes pelo id.	Média
RF05	O sistema deve permitir o cadastro de novos veículos, incluindo informações como marca, modelo, ano, valor da diária e placa.	Alta
RF06	O sistema deve permitir a edição e atualização dos dados dos veículos cadastrados.	Alta
RF07	O sistema deve permitir a exclusão de veículos cadastrados.	Alta
RF08	O sistema deve permitir a pesquisa de veículos com base no id.	Média
RF09	O sistema deve permitir a realização de reservas de veículos por parte dos clientes cadastrados.	Alta
RF10	O sistema deve permitir a edição e cancelamento de reservas de veículos pelos clientes.	Alta
RF11	O sistema deve permitir a visualização do histórico de reservas dos clientes.	Baixa
RF12	O sistema não deve que o cliente seja excluido com reservas ativas.	Alta

2.1. Descrição dos Casos de Uso Principais

- **Caso de Uso 1: Cadastro de Clientes**
  Este caso de uso permite que um funcionário da locadora cadastre um novo cliente no sistema. O funcionário insere as informações do cliente, como nome, CPF, telefone e endereço, e o sistema armazena esses dados para uso futuro.
- **Caso de Uso 2: Cadastro de Veículos**
  Este caso de uso permite que um funcionário da locadora cadastre um novo veículo

no sistema. O funcionário insere as informações do veículo, como marca, modelo, ano, valor da diária e placa, e o sistema armazena esses dados para uso futuro.

- **Caso de Uso 3: Realização de Reservas**
  Este caso de uso permite que um cliente da locadora realize uma reserva de veículo. O cliente seleciona o veículo desejado, informa a data de início e término da reserva, e o sistema verifica a disponibilidade do veículo nesse período. Se o veículo estiver disponível, a reserva é confirmada.
- **Caso de Uso 4: Gerenciamento de Reservas**
  Este caso de uso permite que um cliente gerencie suas reservas existentes. O cliente pode visualizar suas reservas ativas, editar as datas de início e término, e cancelar uma reserva se necessário.
- **Caso de Uso 5: Visualização do Histórico de Reservas**
  Este caso de uso permite que um cliente visualize seu histórico de reservas anteriores. O cliente pode ver detalhes de todas as reservas que fez no passado, incluindo datas, veículos reservados e status da reserva.

3. Arquitetura do Sistema

3.1. Detalhamento das Tecnologias Utilizadas

A estrutura utilizada para desenvolver o sistema segue o modelo MVC (Model, View, Controller), uma abordagem que divide a aplicação em três camadas distintas. A camada View é encarregada de apresentar os dados aos usuários finais. Neste projeto, não foram implementadas Views; em vez disso, essa funcionalidade foi integrada diretamente ao Swagger, que oferece uma interface interativa para testar e documentar as APIs desenvolvidas. A camada Controller, por sua vez, recebe as requisições HTTP, realiza validações e tratamentos de erros, garantindo a integridade e segurança das operações efetuadas. Por fim, a camada Model define as entidades da aplicação e cuida da manipulação dos dados.

3.2. Principais Classes e suas Responsabilidades

- **Cliente:** Esta classe mantém informações dos clientes, como nome, CPF, telefone e endereço. Além disso, ela gerencia operações como cadastro, edição e exclusão de clientes.
- **Veículo:** Responsável por representar os veículos da locadora, esta classe armazena detalhes como marca, modelo, ano, valor da diária e placa. Ela também controla operações como cadastro, edição e exclusão de veículos.
- **Reserva:** Controla as reservas de veículos feitas pelos clientes. Esta classe registra informações como data de início, data de término, o id do cliente e a do veículo. Além disso, gerencia operações como realização, edição e cancelamento de reservas.

4. Descrição do Banco de Dados

4.1. Descrição das Tabelas e Relacionamentos

A estrutura do banco de dados segue o modelo conceitual das classes principais do sistema de locadora de veículos:

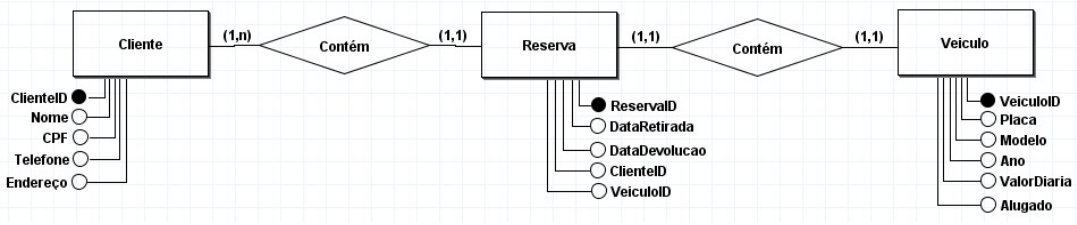

Figure 1. Modelo Conceitual do Banco de Dados

4.1.1. Tabela Cliente

ID (PK): Identificador único do cliente Nome: Nome do cliente CPF: CPF do cliente Telefone: Telefone do cliente Endereço: Endereço do cliente

4.1.2. Tabela Veículo

ID (PK): Identificador único do veículo Marca: Marca do veículo Modelo: Modelo do veículo Ano: Ano do veículo Valor Diária: Valor da diária do veículo Placa: Placa do veículo

4.1.3. Tabela Reserva

ID (PK): Identificador único da reserva Data Início: Data de início da reserva Data Término: Data de término da reserva ID Cliente (FK): Identificador único do cliente associado à reserva ID Veículo (FK): Identificador único do veículo associado à reserva

4.2. Exemplificação de Consultas SQL Relevantes

Consulta 1: Informações das Reservas com Detalhes de Cliente e Veículo

```
SELECT r.Id, r.DataRetirada, r.DataDevolucao, c.ClienteID,
c.Nome as Cliente, v.VeiculoID, v.Modelo as Veiculo
    FROM ORM.dbo.Reserva r
INNER JOIN ORM.dbo.Cliente c ON r.ClienteID = c.ID
INNER JOIN ORM.dbo.Veiculo v ON r.VeiculoID = v.ID;
```

Consulta 2: Detalhes dos Veículos com Estado de Aluguel e Informações de Reserva

```
SELECT v.Id, v.Placa, v.Modelo, v.Ano, v.ValorDiaria, v.Alugado,
       r.ReservaID, r.DataRetirada as Reserva
    FROM ORM.dbo.Veiculo v
LEFT JOIN ORM.dbo.Reserva r ON v.Id = r.VeiculoID;
```

Consulta 3: Informações dos Clientes

```
SELECT Id, Nome, CPF, Telefone, Endereco
    FROM ORM.dbo.Cliente;
```

Operação 1: Inserção de um Novo Cliente

```
INSERT INTO ORM.dbo.Cliente (Nome, CPF, Telefone, Endereco)
VALUES ('João Silva', '123.456.789-00', '(31) 99999-9999',
'Rua das Flores, 123');
```

Operação 2: Exclusão de uma Reserva

```
DELETE FROM ORM.dbo.Reserva
WHERE Id = 123;
```

5. Testes e Validação

5.1. Endpoint Clientes

5.1.1. GET /Clientes

- **Método HTTP:** GET
- **Descrição:** Retorna todos os clientes cadastrados juntamente com as reservas associadas a cada cliente.
- **Parâmetros:** Nenhum
- **Possíveis Códigos de Resposta:**
  - **200 OK:** Retorna a lista de clientes com sucesso.
  - **404 Not Found:** Se não houver nenhum cliente cadastrado.

5.1.2. GET /Clientes/{id}

- **Método HTTP:** GET
- **Descrição:** Retorna os detalhes de um cliente específico.
- **Parâmetros:**
  - **id** (obrigatório): O ID do cliente desejado.
- **Possíveis Códigos de Resposta:**
  - **200 OK:** Retorna os detalhes do cliente com sucesso.
  - **400 Bad Request:** Se o ID fornecido não for válido.
  - **404 Not Found:** Se o cliente com o ID fornecido não for encontrado.

5.1.3. POST /Clientes

- **Método HTTP:** POST
- **Descrição:** Cria um novo cliente.
- **Parâmetros:**
  - **Corpo da solicitação (obrigatório):** Objeto JSON contendo os detalhes do cliente a ser criado.
- **Possíveis Códigos de Resposta:**
  - **201 Created:** Se o cliente for criado com sucesso.
  - **400 Bad Request:** Se os dados fornecidos para o cliente forem inválidos.

5.1.4. PUT /Clientes/{id}

- **Método HTTP:** PUT
- **Descrição:** Atualiza os detalhes de um cliente existente.
- **Parâmetros:**
  - **id** (obrigatório): O ID do cliente a ser atualizado.
  - **Corpo da solicitação (obrigatório):** Objeto JSON contendo os novos detalhes do cliente.
- **Possíveis Códigos de Resposta:**
  - **204 No Content:** Se o cliente for atualizado com sucesso.
  - **400 Bad Request:** Se os dados fornecidos para o cliente forem inválidos.
  - **404 Not Found:** Se o cliente com o ID fornecido não for encontrado.

5.1.5. DELETE /Clientes/{id}

- **Método HTTP:** DELETE
- **Descrição:** Exclui um cliente existente.
- **Parâmetros:**
  - **id** (obrigatório): O ID do cliente a ser excluído.
- **Possíveis Códigos de Resposta:**
  - **204 No Content:** Se o cliente for excluído com sucesso.
  - **404 Not Found:** Se o cliente com o ID fornecido não for encontrado.

5.2. Endpoint Veículos

5.2.1. GET /Veiculos

- Método HTTP: GET
- Descrição: Retorna todos os veículos cadastrados.
- Parâmetros: Nenhum
- Possíveis Códigos de Resposta:
  - 200 OK: Retorna a lista de veículos com sucesso.
  - 404 Not Found: Se não houver nenhum veículo cadastrado.

5.2.2. GET /Veiculos/{id}

- Método HTTP: GET
- Descrição: Retorna os detalhes de um veículo específico.
- Parâmetros:
  - id (obrigatório): O ID do veículo desejado.
- Possíveis Códigos de Resposta:
  - 200 OK: Retorna os detalhes do veículo com sucesso.
  - 400 Bad Request: Se o ID fornecido não for válido.
  - 404 Not Found: Se o veículo com o ID fornecido não for encontrado.

5.2.3. POST /Veiculos

- Método HTTP: POST
- Descrição: Cria um novo veículo.
- Parâmetros:
  - Corpo da solicitação (obrigatório): Objeto JSON contendo os detalhes do veículo a ser criado.
- Possíveis Códigos de Resposta:
  - 201 Created: Se o veículo for criado com sucesso.
  - 400 Bad Request: Se os dados fornecidos para o veículo forem inválidos.

5.2.4. PUT /Veiculos/{id}

- Método HTTP: PUT
- Descrição: Atualiza os detalhes de um veículo existente.
- Parâmetros:
  - id (obrigatório): O ID do veículo a ser atualizado.
  - Corpo da solicitação (obrigatório): Objeto JSON contendo os novos detalhes do veículo.
- Possíveis Códigos de Resposta:
  - 204 No Content: Se o veículo for atualizado com sucesso.
  - 400 Bad Request: Se os dados fornecidos para o veículo forem inválidos.
  - 404 Not Found: Se o veículo com o ID fornecido não for encontrado.

5.2.5. DELETE /Veiculos/{id}

- Método HTTP: DELETE
- Descrição: Exclui um veículo existente.
- Parâmetros:
  - id (obrigatório): O ID do veículo a ser excluído.
- Possíveis Códigos de Resposta:
  - 204 No Content: Se o veículo for excluído com sucesso.
  - 404 Not Found: Se o veículo com o ID fornecido não for encontrado.

5.3. Endpoint Reservas

5.3.1. GET /Reservas

- **Método HTTP:** GET
- **Descrição:** Retorna todas as reservas cadastradas.
- **Parâmetros:** Nenhum
- **Possíveis Códigos de Resposta:**
  - **200 OK:** Retorna a lista de reservas com sucesso.
  - **404 Not Found:** Se não houver nenhuma reserva cadastrada.

5.3.2. GET /Reservas/{id}

- **Método HTTP:** GET
- **Descrição:** Retorna os detalhes de uma reserva específica.
- **Parâmetros:**
  - **id** (obrigatório): O ID da reserva desejada.
- **Possíveis Códigos de Resposta:**
  - **200 OK:** Retorna os detalhes da reserva com sucesso.
  - **400 Bad Request:** Se o ID fornecido não for válido.
  - **404 Not Found:** Se a reserva com o ID fornecido não for encontrada.

5.3.3. POST /Reservas

- **Método HTTP:** POST
- **Descrição:** Cria uma nova reserva.
- **Parâmetros:**
  - **Corpo da solicitação (obrigatório):** Objeto JSON contendo os detalhes da reserva a ser criada.
- **Possíveis Códigos de Resposta:**
  - **201 Created:** Se a reserva for criada com sucesso.
  - **400 Bad Request:** Se os dados fornecidos para a reserva forem inválidos.

5.3.4. PUT /Reservas/{id}

- **Método HTTP:** PUT
- **Descrição:** Atualiza os detalhes de uma reserva existente.
- **Parâmetros:**
  - **id** (obrigatório): O ID da reserva a ser atualizada.
  - **Corpo da solicitação (obrigatório):** Objeto JSON contendo os novos detalhes da reserva.
- **Possíveis Códigos de Resposta:**
  - **204 No Content:** Se a reserva for atualizada com sucesso.
  - **400 Bad Request:** Se os dados fornecidos para a reserva forem inválidos.
  - **404 Not Found:** Se a reserva com o ID fornecido não for encontrada.

5.3.5. DELETE /Reservas/{id}

- **Método HTTP:** DELETE
- **Descrição:** Exclui uma reserva existente.
- **Parâmetros:**
  - **id** (obrigatório): O ID da reserva a ser excluída.
- **Possíveis Códigos de Resposta:**
  - **204 No Content:** Se a reserva for excluída com sucesso.
  - **404 Not Found:** Se a reserva com o ID fornecido não for encontrada.

5.4. Identificação de Problemas Encontrados

Durante os testes, identificamos alguns problemas superficiais de validação que não foram inicialmente detectados. Um desses problemas incluía a falta de atualização das reservas do cliente. No entanto, essas questões foram prontamente abordadas e resolvidas durante o curso do projeto.

6. Resultados e Conclusões

6.1. Análise dos Resultados Obtidos

Os resultados alcançados foram considerados satisfatórios, demonstrando a eficácia do sistema em cumprir suas funções conforme o esperado.

6.2. Conclusões sobre a Eficiência do Sistema

Concluímos que, apesar de sua base simplificada, o sistema atende de forma eficiente às necessidades identificadas, demonstrando sua viabilidade e utilidade para os usuários finais.

7. Considerações Finais

7.1. Reflexões sobre Experiências Adquiridas

Durante o processo, houve oportunidades que poderiam ter sido mais bem exploradas, contribuindo para uma experiência ainda mais enriquecedora.

7.2. Sugestões para Melhorias Futuras

Acredita-se que o ensino presencial seria mais benéfico para esta disciplina, dada a sua importância e complexidade. O formato de Ensino a Distância (EAD) torna mais difícil obter suporte direto para esclarecer dúvidas, o que pode complicar o desenvolvimento técnico em cada tópico abordado. Seria vantajoso incluir mais exemplos práticos em diferentes níveis de dificuldade, evitando generalizações que possam dificultar o entendimento.

8. Referências

1. Microsoft. (n.d.). Introdução ao Entity Framework Core no ASP.NET Core MVC. Recuperado de https://learn.microsoft.com/pt-br/aspnet/core/data/ef-mvc/intro?view=aspnetcore-8.0
2. Microsoft. (n.d.). Acessar SQL: Conceitos básicos, vocabulário e sintaxe. Recuperado de https://support.microsoft.com/pt-br/topic/acessar-sql-conceitos-b%C3%A1sicos-vocabul%C3%A1rio-e-sintaxe-444d0303-cde1-424e-9a74-e8dc3e460671
3. Universidade Federal de Santa Catarina. (n.d.). Como escrever textos em LaTeX. Recuperado de https://www.inf.ufsc.br/~j.barreto/cca/tratexto/latex.htm
4. Endeavor Brasil. (n.d.). Como elaborar um pitch quase perfeito. Recuperado de https://endeavor.org.br/dinheiro/como-elaborar-um-pitch-quase-perfeito/

9. Apêndices

9.1. Documentação Adicional

9.2. Teste de Métodos com Swagger

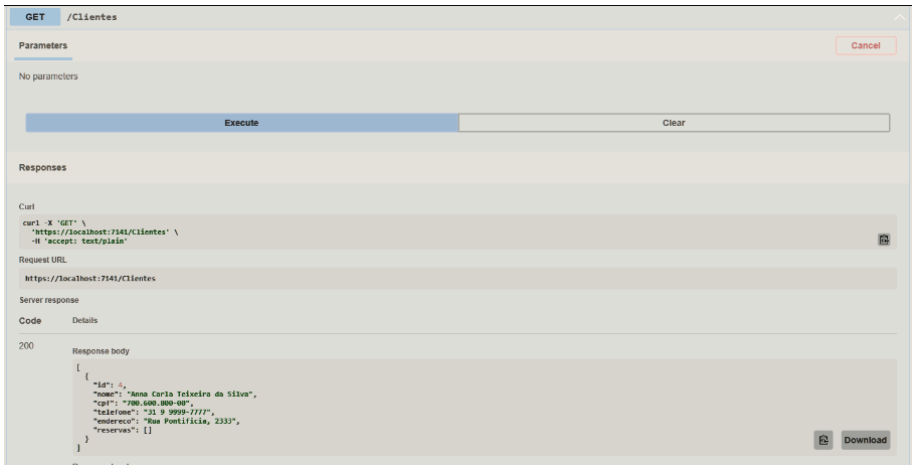

Figure 2. Método GET com Resposta (Cliente) - Swagger

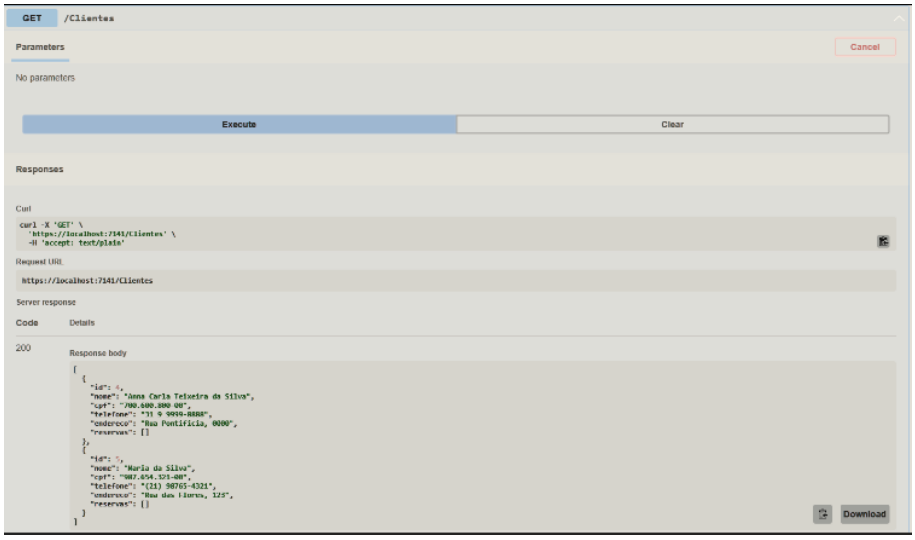

Figure 3. Método GET (Cliente) - Swagger

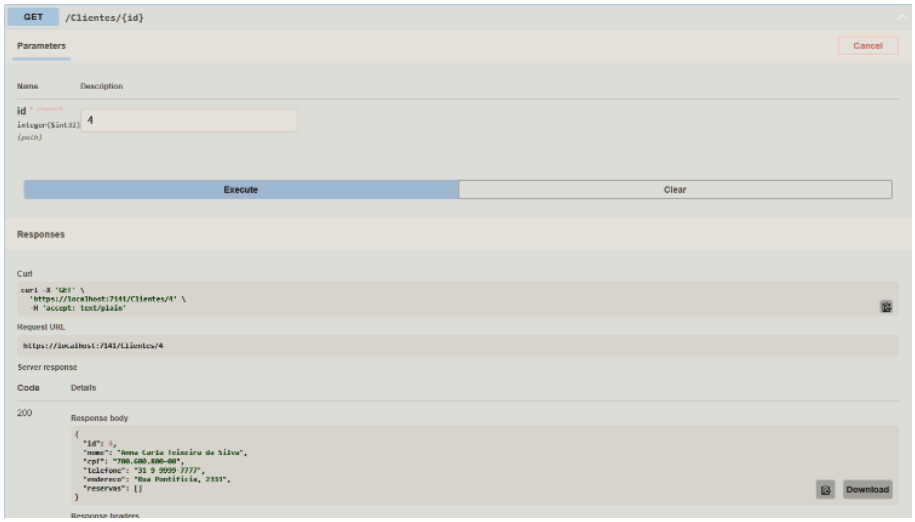

Figure 4. Método GET por id com Resposta (Cliente) - Swagger

Figure 5. Método POST (Cliente) - Swagger

Figure 6. Resposta do POST (Cliente) - Swagger

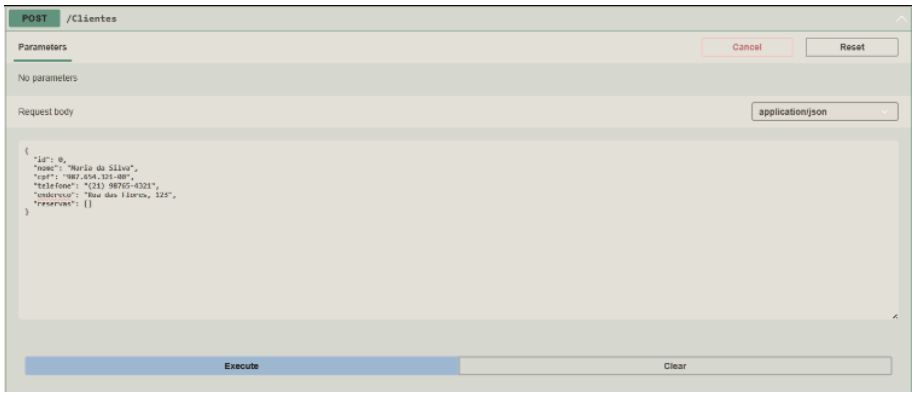

Figure 7. Método POST (2º Cliente) - Swagger

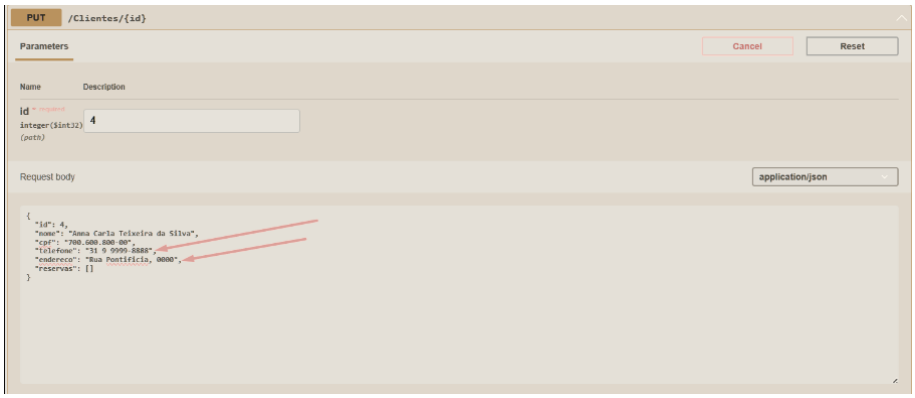

Figure 8. Método PUT (Cliente) - Swagger

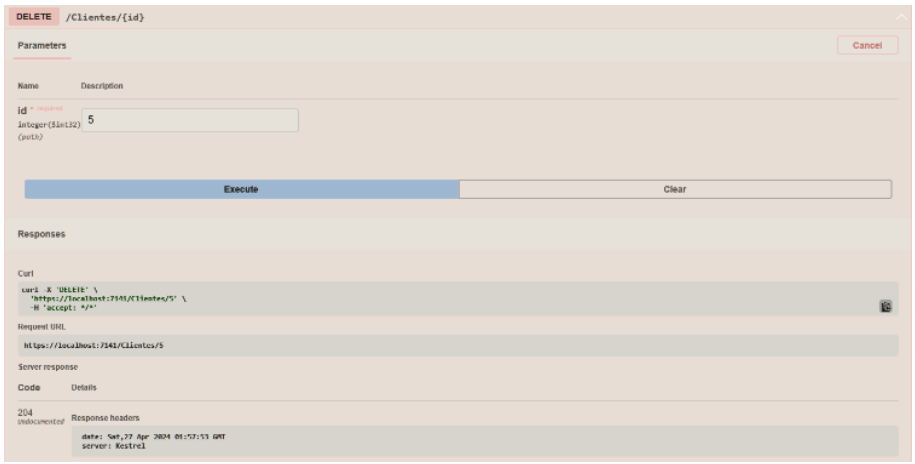

Figure 9. Método DELETE por id (Cliente) - Swagger

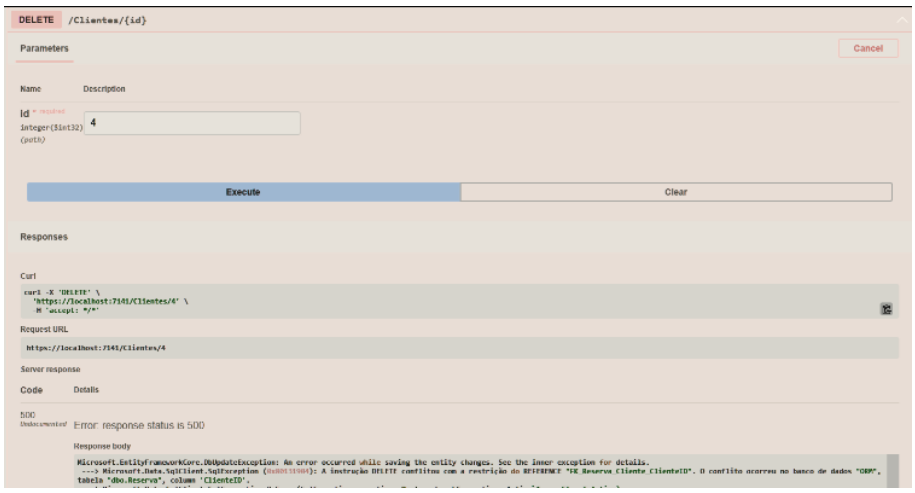

Figure 10. Método DELETE (Cliente com Reserva) - Swagger

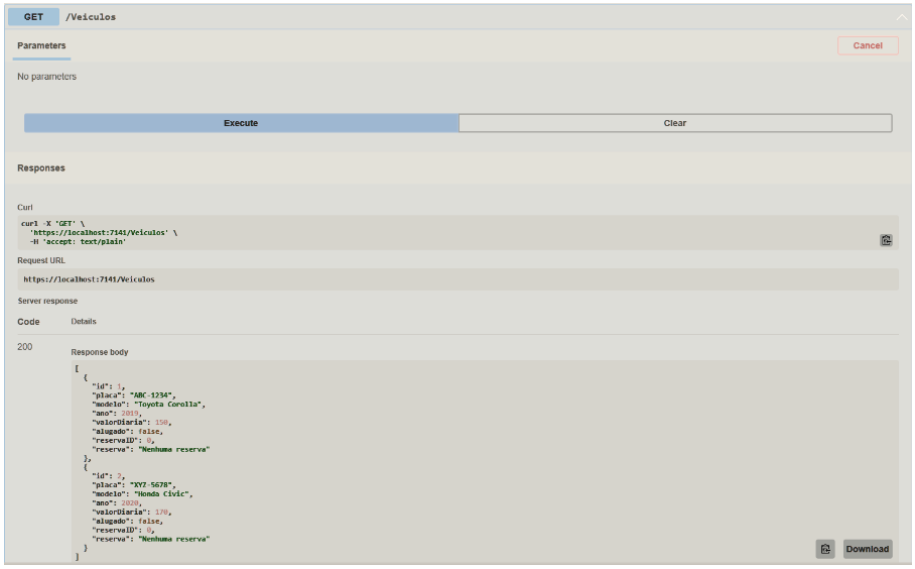

Figure 11. Método GET (Veiculos) - Swagger

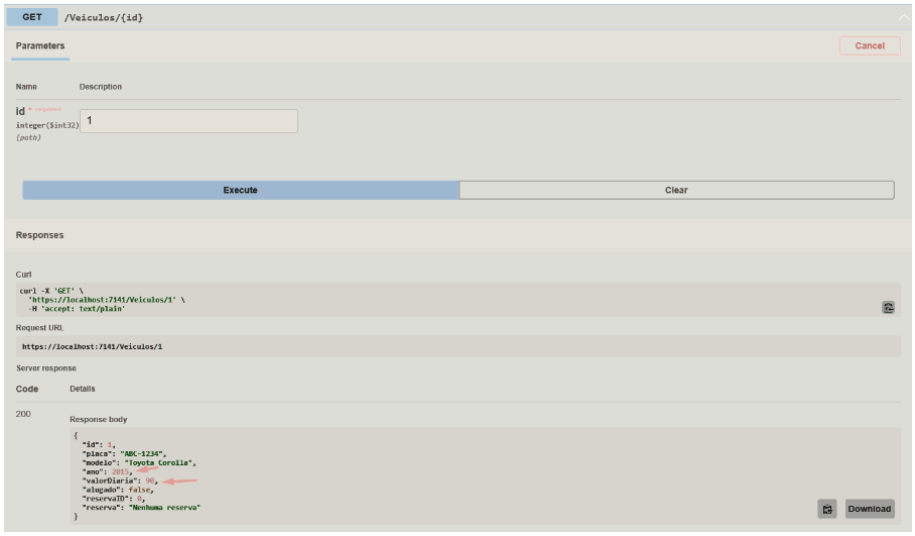

Figure 12. Método GET por id (Veiculos) - Swagger

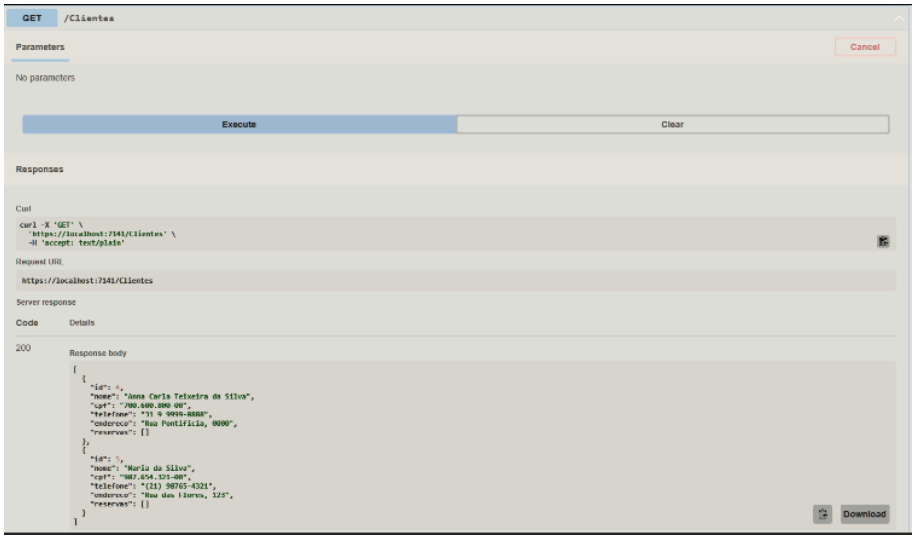

Figure 13. Método GET por Id (2) (Veiculos) - Swagger

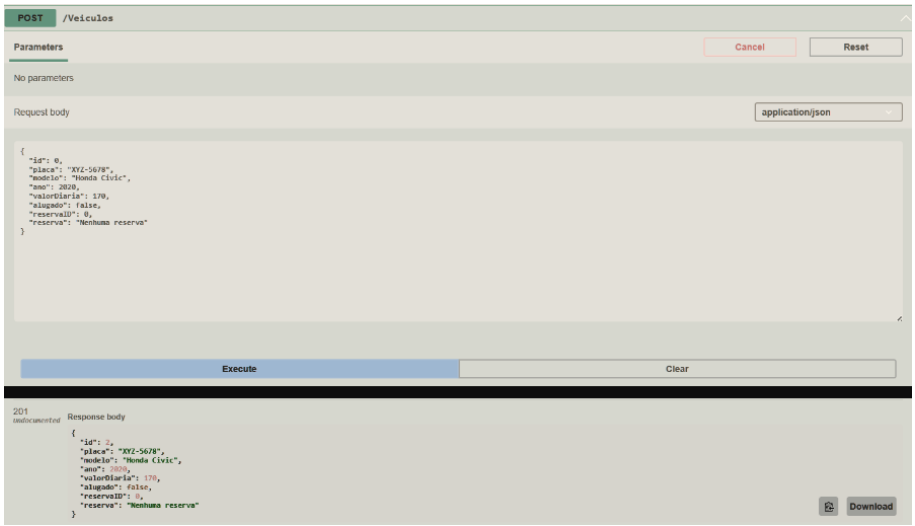

Figure 14. Método POST (Veiculos) - Swagger

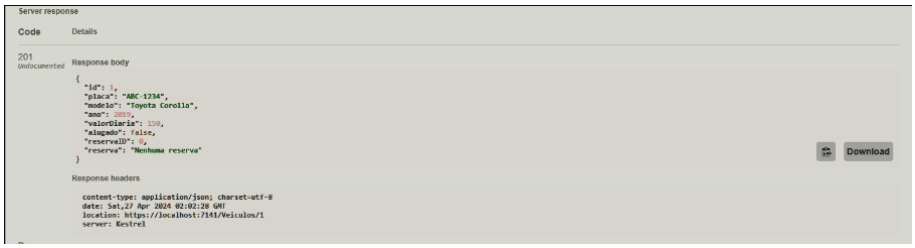

Figure 15. Método POST (Veiculo 2) - Swagger

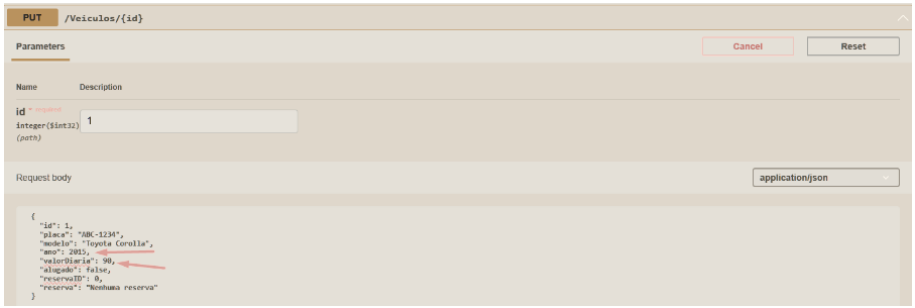

Figure 16. Método PUT (Veiculos) - Swagger

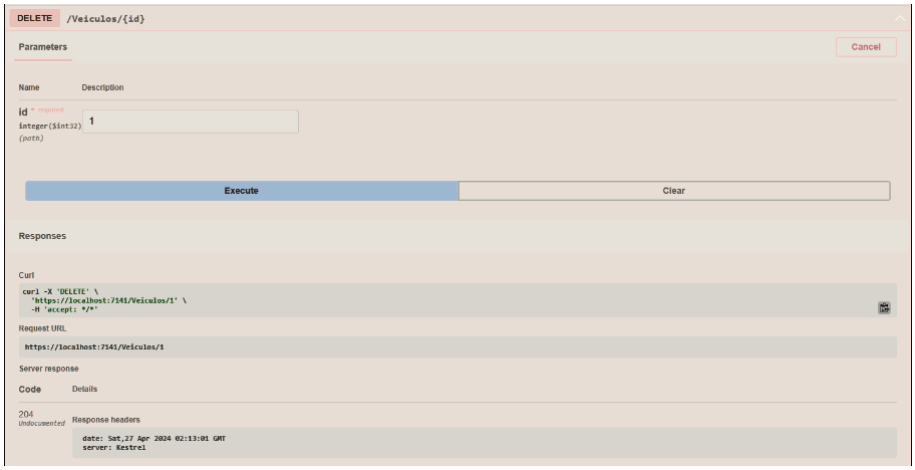

Figure 17. Método DELETE (Veiculos) - Swagger

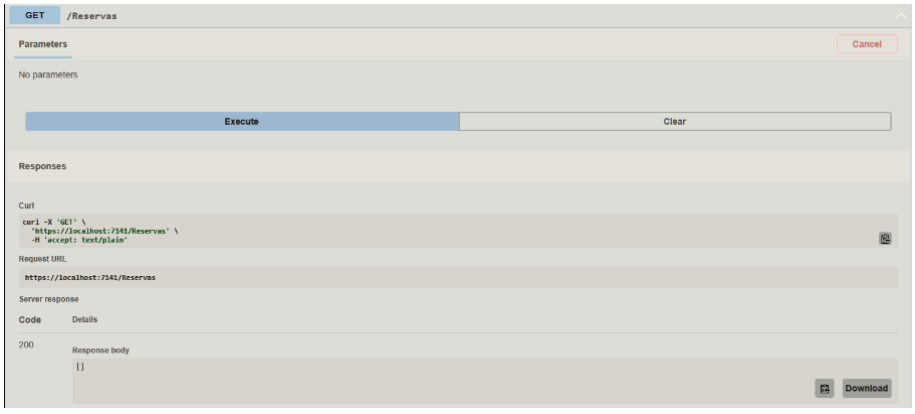

Figure 18. Método GET (Reserva) - Swagger

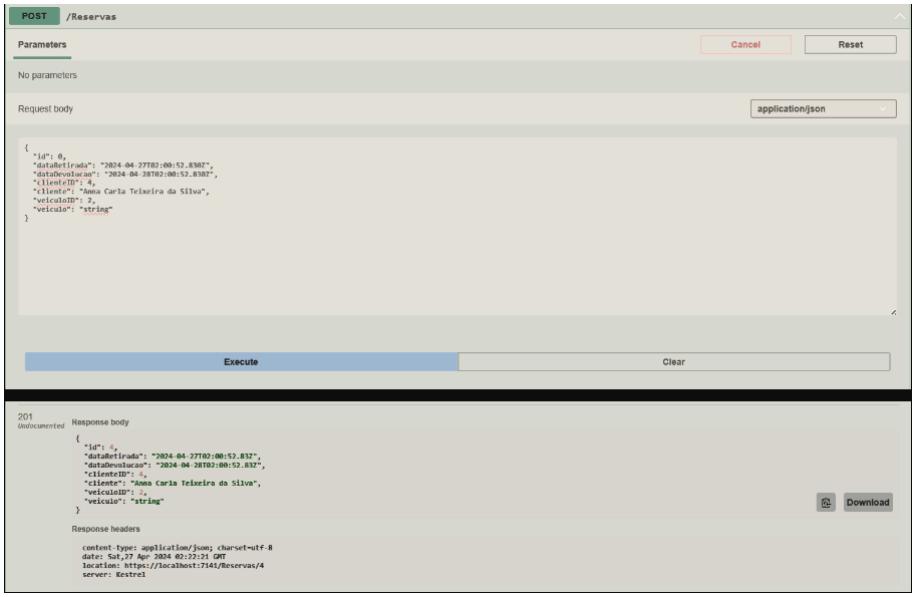

Figure 19. Método POST (Reserva) - Swagger

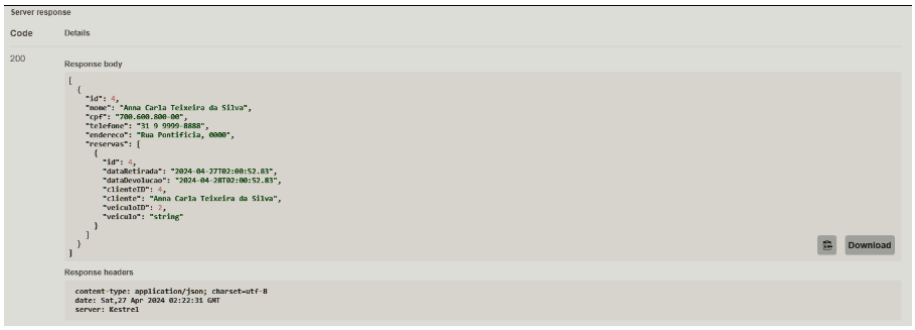

Figure 20. Método GET clientes com a Reserva (Reserva) - Swagger

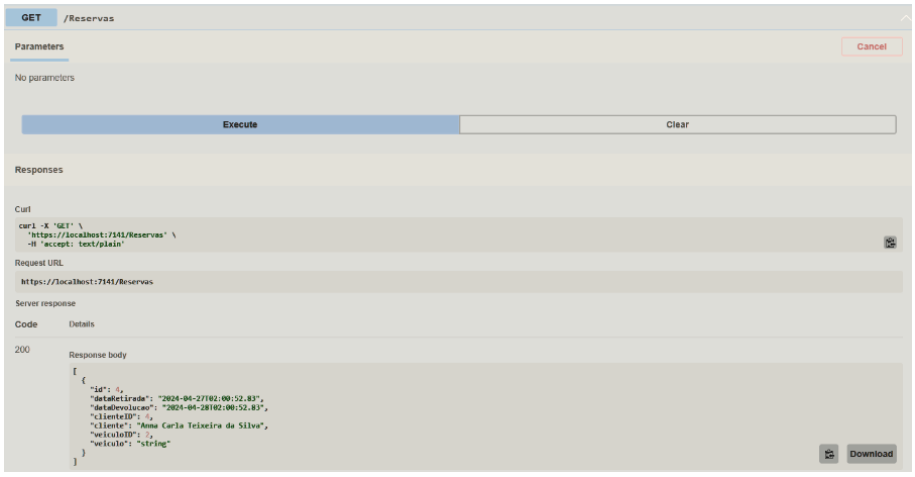

Figure 21. Método GET (Reserva) - Swagger

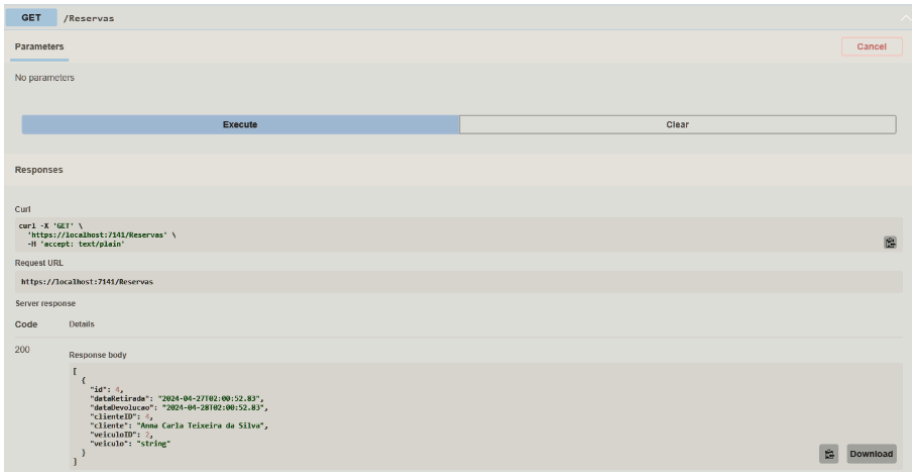

Figure 22. Método GET por id (Reserva) - Swagger

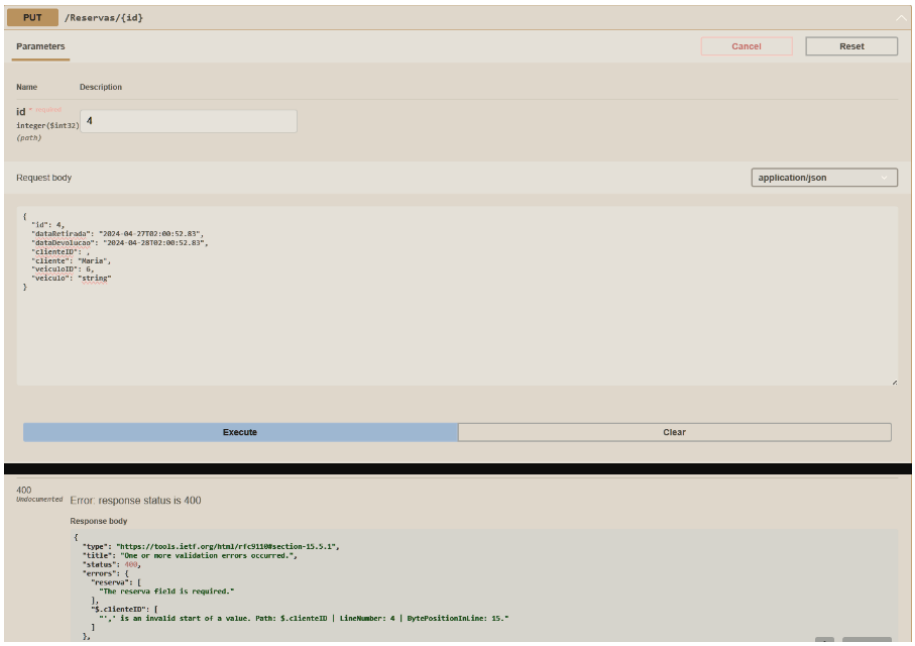

Figure 23. Método PUT por id passando reserva para outro cliente ’com erro esperado’(Reserva) - Swagger

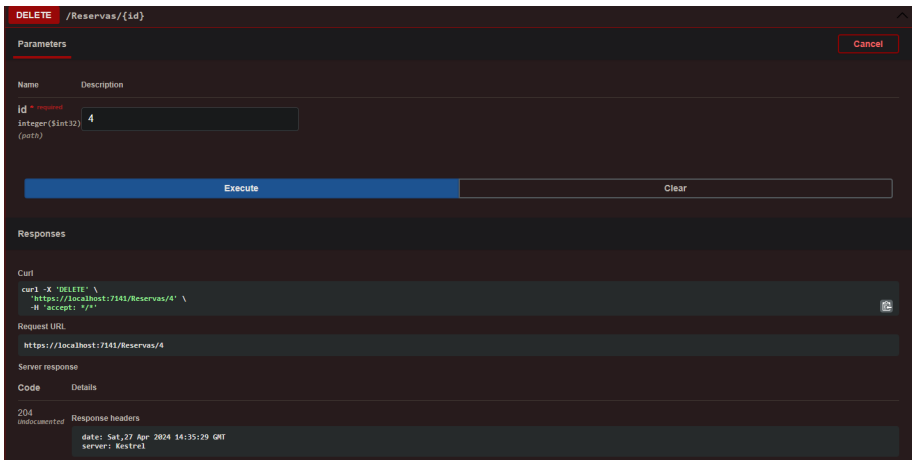

Figure 24. Método DELETE (Reserva) - Swagger

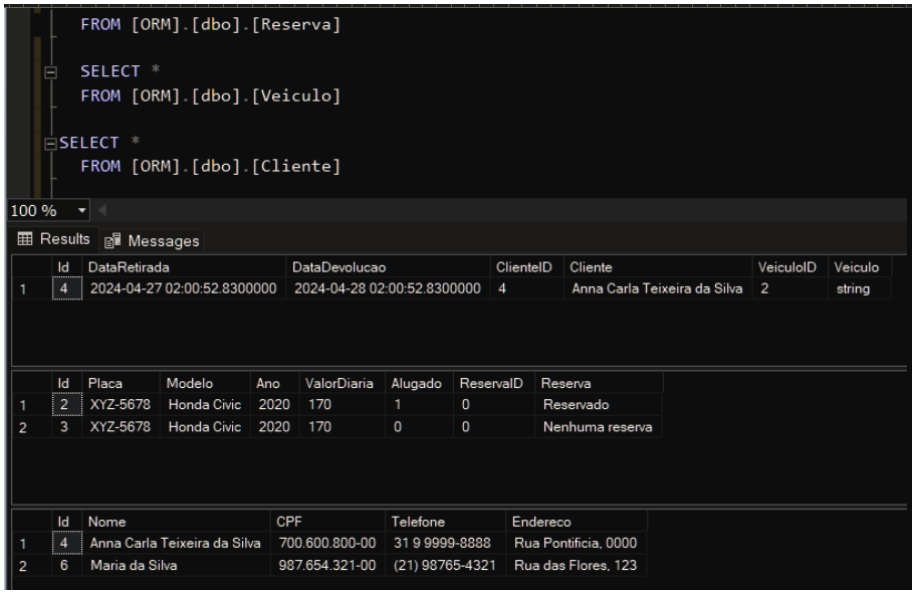

Figure 25. Banco de Dados após Testes