

Relatório de Funcionamento

Api com MVC e SQL Server Express

Arthur Henrique da Silva Proença¹

¹Pontifícia Universidade Católica de Minas Gerais
– Betim – MG – Brazil

ahsproenca@sga.pucminas.br

Abstract. *This article describes the challenges and information about developing a car rental system using C 6.0 or higher, Entity Framework, SQL Express and Swagger. The project was organized in stages, starting with the creation of the conceptual model and the implementation of the entities in C, followed by the development of the backend with ASP.NET Core, the creation of APIs for CRUD operations.*

Resumo. *Esse artigo descreve os desafios e informações sobre o desenvolvimento de um sistema de locadora de veículos utilizando C 6.0 ou superior, Entity Framework, SQL Express e Swagger. O projeto foi organizado em etapas, começando com a criação do modelo conceitual e a implementação das entidades em C, seguido pelo desenvolvimento do backend com ASP.NET Core, a criação de APIs para operações CRUD.*

1. Introdução

1.1. Contextualização

A sociedade, ao longo da história, sempre careceu de novos serviços cada vez mais específicos para melhor qualidade de vida como saúde, educação e segurança. E como a mobilidade é um quesito considerado padrão para se alcançar todos os outros hoje em dia, surge a necessidade de criar sistemas de negócios que buscam gerir e atender essa demanda, na qual serve como ponte para que mais pessoas tenham acesso a tais serviços. Por isso, com o aumento da busca por serviços de aluguel de veículos, as locadoras devem ter um sistema eficaz para administrar sua frota, clientes e reservas. Um sistema de locação de veículos informatizado não apenas aumenta a produtividade, mas também garante atendimento ao cliente mais rápido e preciso, facilitando na tomada de decisões estratégicas com base em melhores informações.

1.2. Objetivos

O principal objetivo deste projeto é desenvolver um sistema robusto e eficiente para que os funcionários possam gerenciar uma locadora de veículos. Especificamente, o sistema visa:

Automatizar Processos: Reduzir a carga de trabalho manual, minimizar erros humanos e aumentar a eficiência operacional.

Gerenciar Frota de Veículos: Manter um registro detalhado dos veículos disponíveis, suas condições e disponibilidade.

Gerenciar Clientes: Facilitar o cadastro e a consulta de informações dos clientes, assegurando um serviço mais personalizado e eficiente.

Gerenciar Reservas: Permitir a criação, atualização e cancelamento de reservas de forma fácil e rápida.

1.3. Tecnologias usadas

1.3.1. C sharp 6.0

Linguagem de programação orientada a objetos para desenvolvimento do backend, incluindo controllers da API, definições de entidades e lógica de negócios.

1.3.2. ASP.NET Core

Framework de desenvolvimento web open-source da Microsoft, conhecido por sua boa performance, modularidade e suporte multiplataforma (Windows, macOS e Linux). Utiliza uma arquitetura modular que permite adicionar apenas os componentes necessários. Utilizado para criar APIs RESTful que expõem os endpoints para operações CRUD sobre as entidades do sistema (Veículo, Cliente, Reserva).

1.3.3. Entity Framework Core

Mapeamento de objetos relacionais (ORM) simplifica o acesso aos dados em bancos de dados relacionais. Um modelo de programação orientado a objetos na qual permite a interação com o banco de dados. Utilizado para mapear classes de entidades do C sharp para tabelas no banco de dados SQL Express. As operações CRUD (criar, ler, atualizar e apagar) e a definição de relacionamentos entre entidades são facilitadas por ele.

1.3.4. SQL Express

Versão de entrada e gratuita do Microsoft SQL Server é ideal para aplicações de desenvolvimento e médio a pequeno porte. Apresenta segurança robusta, backups automáticos, integridade referencial, índices e bom suporte. Serve principalmente como um banco de dados relacional para armazenar informações sobre locadoras de veículos.

1.3.5. Swagger

Swagger é uma ferramenta que fornece aos desenvolvedores e testers uma interface interativa para documentar e testar as APIs que eles criaram.

1.3.6. Visual Studio 2022

IDEs (Integrated Development Environments) da Microsoft que oferece ferramentas robustas para desenvolvimento, depuração e teste de aplicações. Será usado para escrever, depurar e testar o código C sharp.

1.3.7. Git Hub

O GitHub, uma plataforma de hospedagem de código, que facilita o versionamento e o trabalho em equipe no desenvolvimento de software. Mantém as versões do código-fonte sob controle, administra branches. Além disso, os recursos do GitHub, como pull requests, issues e integração contínua, ajudam na organização e na qualidade dos projetos.

2. Requisitos funcionais

O diagrama de casos de uso mostrado na figura 1 demonstra bem como o sistema irá funcionar. Primeiramente há um ator "funcionário", na qual será responsável por todas as ações no sistema. Dentre os requisitos funcionais principais estão "cadastrar cliente, cadastrar veículo e cadastrar reserva. Caso, após realização de tais atividades, surgir a necessidade, o funcionário tem a opção de excluir, ou atualizar os dados já cadastrados, isso é perceptível via "Extend"s mostrados. Além de tais funções básicas, ainda possuem a listagem dos dados que não precisam ser feitas obrigatoriamente após algum cadastro, essa atividade pode ser feita até mesmo quando o banco de dados estiver vazio.

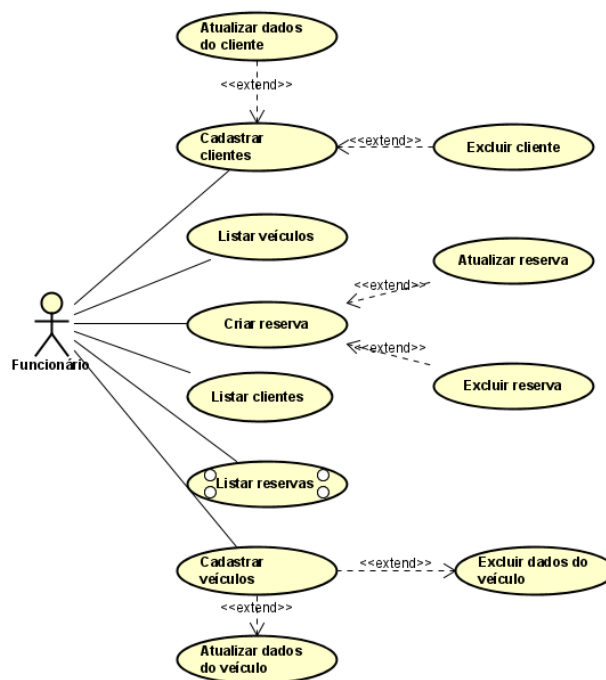


Figura 1. Exemplo do diagrama de Casos de uso feito no app Astah UML

3. Arquitetura do Sistema

3.1. Descrição de Arquitetura Utilizada

O sistema de locadora de veículos utiliza a arquitetura MVC (Model-View-Controller), uma abordagem amplamente adotada que facilita a separação de responsabilidades, promovendo uma estrutura de código organizada e modular. No contexto deste

projeto, que se trata de um backend de API, a ênfase é principalmente nas partes de Model e Controller.

3.2. Camadas do Sistema

3.2.1. Camada de Modelos

A camada Model inclui as entidades e o contexto do Entity Framework que interage com o banco de dados. As principais responsabilidades incluem:

Entidades: Definição das classes que representam as tabelas do banco de dados (e.g., Veiculo, Cliente, Reserva).

Contexto do Banco de Dados: Classe derivada de DbContext que gerencia a conexão com o banco de dados e as operações CRUD.

3.2.2. Camada de Controladores

A camada Controller gerencia as requisições HTTP, chamando os métodos apropriados no Model e retornando respostas. As principais responsabilidades incluem:

Definição de Endpoints: Métodos que mapeiam para rotas HTTP específicas.
Manipulação de Requisições: Processamento de entrada e saída de dados.

Tratamento de Erros: Garantir que as respostas adequadas sejam retornadas em caso de erros.

3.3. Diagrama de Classes

O diagrama de classes do sistema representado na figura 2 possui três entidades: Veiculo, Cliente e Reserva. A entidade Veiculo encapsula atributos como Id, Modelo, Marca, Ano, Placa e Disponibilidade, representando os veículos disponíveis para locação. A entidade Cliente inclui atributos como Id, Nome, CPF, Endereço e Telefone, representando os clientes da locadora. A entidade Reserva relaciona Cliente e Veiculo, incorporando atributos como ClienteId, VeiculoId e pagamento. As relações entre essas classes são geridas pelo Migration "LocadoraContextModelSnapshot", que define o contexto do banco de dados, incluindo as tabelas e as chaves estrangeiras para manter a integridade referencial entre as entidades.

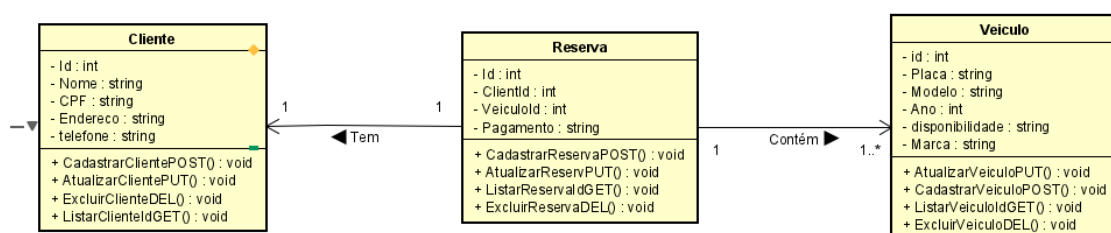


Figura 2. Diagrama de Classes feito no app Astah UML

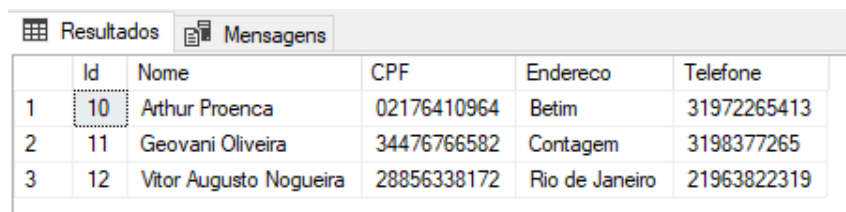
4. Descrição do Banco de Dados

4.0.1. Modelo de Dados

O modelo de dados do sistema de locadora de veículos foi desenvolvido para atender às necessidades operacionais e de negócios, garantindo que os dados fossem armazenados e recuperados com integridade e eficiência. As três entidades principais do modelo são o veículo, o cliente e a reserva. O banco mapeia cada entidade para uma tabela.

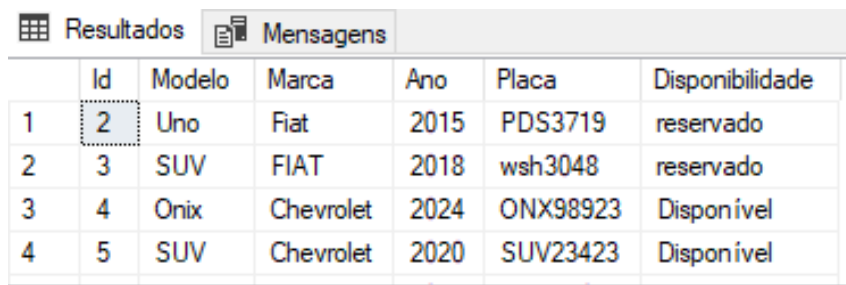
4.1. Descrição das Tabelas

Readaptando a descrição do diagrama de classes para as tabelas do banco, cada classe recebeu sua tabela com seus respectivos atributos, sendo o ID a chave primária de cada uma delas e na tabela reserva possuindo a chave estrangeira do cliente que fez a reserva e do veículo que foi reservado. As figuras 3, 4 e 5 mostram as respectivas tabelas citadas e alguns exemplos de dados cadastrados.



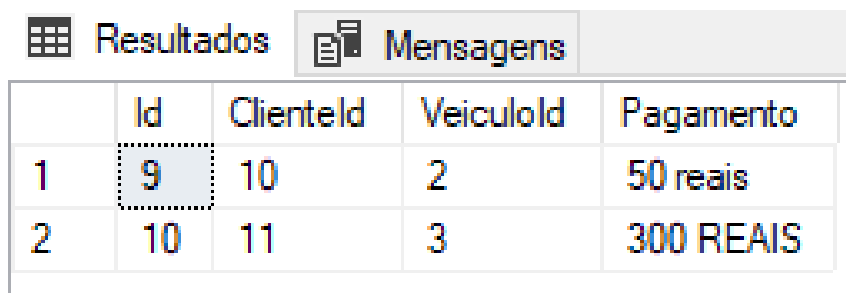
	Id	Nome	CPF	Endereco	Telefone
1	10	Arthur Proenca	02176410964	Betim	31972265413
2	11	Geovani Oliveira	34476766582	Contagem	3198377265
3	12	Vitor Augusto Nogueira	28856338172	Rio de Janeiro	21963822319

Figura 3. Demonstração da tabela de clientes no SQL Express



	Id	Modelo	Marca	Ano	Placa	Disponibilidade
1	2	Uno	Fiat	2015	PDS3719	reservado
2	3	SUV	FIAT	2018	wsh3048	reservado
3	4	Onix	Chevrolet	2024	ONX98923	Disponível
4	5	SUV	Chevrolet	2020	SUV23423	Disponível

Figura 4. Demonstração da tabela de veículos no SQL Express



	Id	ClientId	VeiculoId	Pagamento
1	9	10	2	50 reais
2	10	11	3	300 REAIS

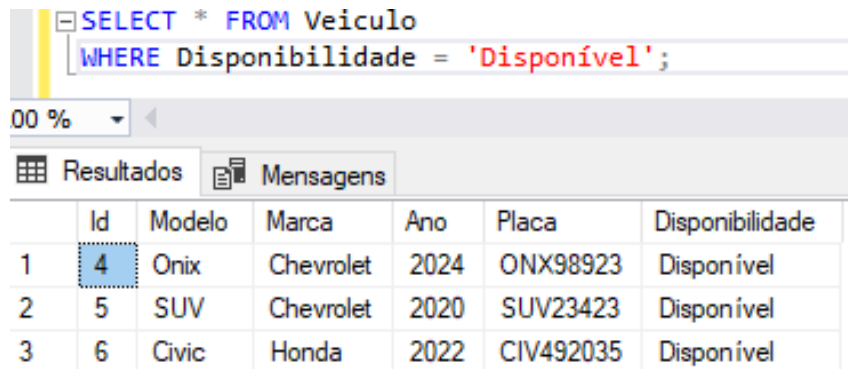
Figura 5. Demonstração da tabela de reservas no SQL Express

4.2. Exemplos de consulta ao Banco de Dados

4.2.1. Consulta com atributos específicos

Foi feito alguns testes para provar a filtragem de dados, segue a seguir os exemplos com imagens.

- Verificação de veículos disponíveis para reserva:



```
SELECT * FROM Veiculo
WHERE Disponibilidade = 'Disponível';
```

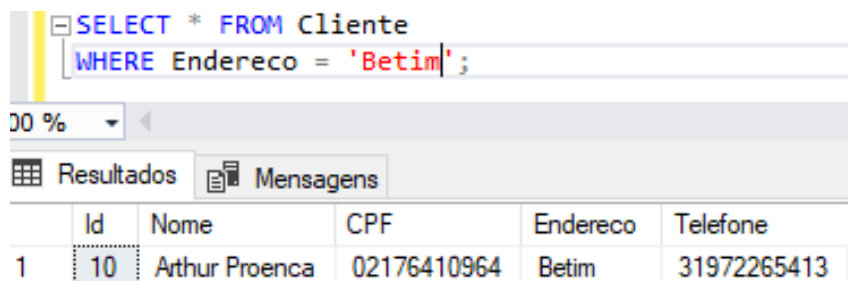
00 %

Resultados Mensagens

	Id	Modelo	Marca	Ano	Placa	Disponibilidade
1	4	Onix	Chevrolet	2024	ONX98923	Disponível
2	5	SUV	Chevrolet	2020	SUV23423	Disponível
3	6	Civic	Honda	2022	CIV492035	Disponível

Figura 6. Resultado de carros disponíveis

- Busca por Cliente de determinado endereço:



```
SELECT * FROM Cliente
WHERE Endereco = 'Betim';
```

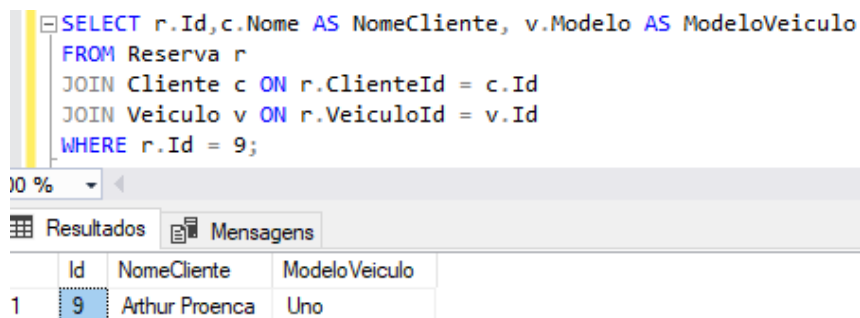
00 %

Resultados Mensagens

	Id	Nome	CPF	Endereco	Telefone
1	10	Arthur Proenca	02176410964	Betim	31972265413

Figura 7. Resultado de endereços

- Consulta de detalhes de uma reserva específica, incluindo nome do cliente e modelo do veículo:



```
SELECT r.Id, c.Nome AS NomeCliente, v.Modelo AS ModeloVeiculo
FROM Reserva r
JOIN Cliente c ON r.ClienteId = c.Id
JOIN Veiculo v ON r.VeiculoId = v.Id
WHERE r.Id = 9;
```

00 %

Resultados Mensagens

	Id	NomeCliente	ModeloVeiculo
1	9	Arthur Proenca	Uno

Figura 8. Resultado de mesclagem de tabelas

5. Testes e Validação

Os testes servem para garantir que o sistema funcione corretamente e que todos os requisitos sejam atendidos. Alguns testes foram realizados no sistema de locadora de veículos para garantir a funcionalidade, integridade dos dados e desempenho.

Primeiramente foi feito um teste de criação de dados via Swagger. Foi feito o teste de criação de um novo cliente e de acordo com a própria verificação do Swagger, o sistema conseguiu salvar os dados, como mostrado na figura 9.

Code	Details
201 <i>Undocumented</i>	<p>Response body</p> <pre>{ "id": 13, "nome": "Larissa da Silva", "cpf": "45528930071", "endereco": "Betim", "telefone": "31928361171", "reserva": null }</pre>

Figura 9. Confirmação de cadastro

Outro teste feito foi o de listagem de itens de veículos no banco, tal verificação foi feita com a funcionalidade GET do sistema e funcionou com sucesso.

Code	Details
200	<p>Response body</p> <pre>{ { "id": 4, "modelo": "Onix", "marca": "Chevrolet", "ano": 2024, "placa": "ONX98923", "disponibilidade": "Disponível", "reserva": null }, { "id": 5, "modelo": "SUV", "marca": "Chevrolet", "ano": 2020, "placa": "SUV23423", "disponibilidade": "Disponível", "reserva": null }, { "id": 6, "modelo": "Civic", "marca": "Honda", "ano": 2022, "placa": "CIV492035", "disponibilidade": "Disponível", "reserva": null } }</pre>

Figura 10. Listagem do banco de veículos

Foi feito o teste de atualização de dados, na qual Foi feito com o Cliente de ID 11. O dado atualizado foi o de endereço.

```
{
  "id": 11,
  "nome": "Geovani Oliveira",
  "cpf": "34476766582",
  "endereco": "Contagem",
  "telefone": "3198377265",
  "reserva": null
},
```

Figura 11. Dados antigos do Cliente 11

```
{
  "id": 11,
  "nome": "Geovani Oliveira",
  "cpf": "34476766582",
  "endereco": "Betim",
  "telefone": "3198377265",
  "reserva": null
},
```

Figura 12. Dados novos do Cliente 11

A cobertura de testes foi abordada de maneira abrangente, garantindo que os principais componentes do sistema fossem testados. Foram realizados testes do CRUD para que funcionassem corretamente conforme as especificações.

6. Resultados e Conclusões

O desenvolvimento desse trabalho resultou em uma aplicação simples, mas com capacidade de gerenciar veículos, clientes e reservas de forma eficiente. As funcionalidades implementadas incluem operações CRUD completas, garantindo a integridade dos dados. As tecnologias utilizadas, como ASP.NET Core e Entity Framework Core, proporcionaram um desenvolvimento ágil e um desempenho adequado. O uso do Swagger facilitou a validação e o uso dos endpoints da API, enquanto as práticas de validação e controle de concorrência garantiram a qualidade dos dados e a integridade das operações.

Com base na documentação construída, é perceptível que o sistema atende às necessidades básicas de uma locadora de veículos, oferecendo uma solução eficaz e preparada para futuras atualizações. Dentre as futuras melhorias poderiam ser construídas uma interface de usuário, sistema de notificações e criação de relatórios semanais ou mensais. Dessa forma, o sistema se tornaria mais utilizável e melhor para as inúmeras demandas que encontramos na realidade atual.

7. Bibliografia

Referências

- [1] Swagger. *Como é o funcionamento do Swagger?*. Disponível em: <https://grld.io/2022/04/15/swagger/>.
- [2] ORM. *Conceitos básicos*. Disponível em: <https://www.devmedia.com.br/orm-object-relational-mapper/19056>.
- [3] Astah. *Astah UML*. Disponível em: <https://astah.net/pt/>.
- [4] SQL. *SQL Express - Suporte*. Disponível em: <https://learn.microsoft.com/pt-br/answers/>.
- [5] Astah. *Astah UML*. Disponível em: <https://astah.net/pt/>.