Desenvolvimento de API para locadoras de veículos

Luana Schelb Teixeira[1]

[1]Ciência da Computação – Pontifícia Universidade Católica de Minas Gerais (PUC-MG)
Poços de Caldas – MG – Brazil

lua.schelb@gmail.com

1. Introdução

O projeto desenvolvido tem o propósito de fornecer uma API funcional para locadoras de veículos. Esse desenvolvimento é importante, pois código livre criados como trabalhos práticos podem servir de aprendizado ou de base para implementação de um sistema no mundo real. O sistema permite ao usuário cadastrar as informações necessárias para uma locadora de veículos como criar e gerenciar suas reservas, clientes, carros, funcionários entre outros pontos. A aplicação foi desenvolvida na linguagem C# com a framework ASP.NET Core[Masse 2011].

Foi gravado um Pitch demonstrativo explicando o sistema e suas funcionalidades. O vídeo está hospedado na plataforma Youtube no link a seguir https://www.youtube.com/watch?v=ZZQRuGNeYvY

2. Requisitos Funcionais

O sistema fornece o gerenciamento, ou seja, listagem, criação, edição, atualização e deleção, para cada entidade da aplicação gerando os casos de uso:

- CU-01: O Sistema deve permitir o gerenciamento de Veículos
- CU-02: O Sistema deve permitir o gerenciamento de Clientes
- CU-03: O Sistema deve permitir o gerenciamento de Funcionários
- CU-04: O Sistema deve permitir o gerenciamento de Reservas
- CU-05: O Sistema deve permitir o gerenciamento de Manutenções
- CU-06: O Sistema deve permitir o gerenciamento de Mecânicos
- CU-07: O Sistema deve permitir o gerenciamento de Ocorrências
- CU-08: O Sistema deve permitir o gerenciamento de Locadoras
- CU-09: O Sistema deve permitir o gerenciamento de Feedbacks

3. Arquitetura do Sistema

No desenvolvimento da aplicação decidimos utilizar o padrão de arquitetura de sistema MVC (Model View Controller)[Reenskaug 2001]. Este padrão consiste em separar a aplicação em três camadas: uma cama de interação com o usuário (View), uma camada de manipulação dos dados (Model) e uma camada de controle (Controller). A escolha dessa arquitetura foi motivada pela sua capacidade de separar as responsabilidades e com isso melhorar a modularidade da aplicação.

4. Implementação Técnica

A aplicação foi desenvolvida utilizando as tecnologias C# 8.0, ASP.NET Core, Entity Framework, SQL Express, Swagger,

Apesar de termos adotado o padrão MVC na aplicação, fizemos a escolha de design de não implementar as views devido ao escopo do desenvolvimento. Porém, caso a aplicação fosse utilizada no mundo real seriam criadas views para permitir aos usuários interagirem com o sistema.

5. Descrição do Banco de Dados

Para mapear as classes referenciadas nos casos de uso criamos as tabelas abaixo no SQL SERVER e definimos os relacionamentos. Todas as chaves estrangeiras estão com o comportamento de deleção padrão em cascata.

Utilizando a ferramenta SQL Express podemos gerar um diagrama do banco de dados para exemplificar a estrutura e relações entre tabelas. Geramos na figura abaixo a representação do banco de dados criado

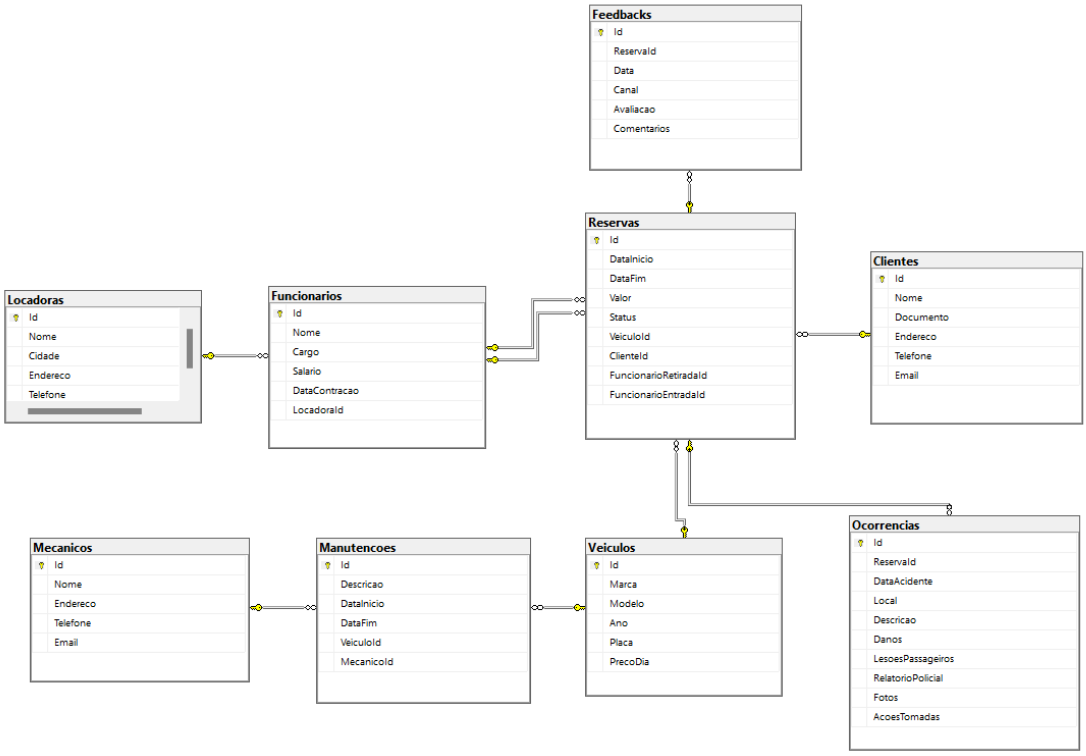

Figura 1. Diagrama do Banco de Dados gerado pelo SQL Express

Consultas SQL Relevantes

1. Selecionar todos os veículos disponíveis para reserva:

```
SELECT * FROM Veiculos WHERE Id NOT IN
(SELECT VeiculoId FROM Reservas WHERE Status = 'Ativa');
```

2. Obter todas as reservas feitas por um cliente específico:

```
SELECT * FROM Reservas WHERE ClienteId = [ID_do_Cliente];
```

3. Calcular o total de gastos de um cliente em reservas:

```
SELECT SUM(Valor) AS Total_Gasto
FROM Reservas WHERE ClienteId = [ID_do_Cliente];
```

Essas consultas exemplificam como os dados podem ser manipulados no contexto do sistema de locadora de veículos, proporcionando informações úteis para gerenciamento e análise.

6. Testes e Validação

Os testes manuais foram feitos utilizando o Swagger, para cada requisição realizamos o teste com dados válidos para confirmar o funcionamento dos fluxos principais. Para automatizar e montar um relatório dos testes utilizamos a ferramenta Postman para utilizar cada rota e verificar o cenário positivo. Na figura abaixo está exibido o resultado do último teste em que foram testadas as 45 rotas com um total de 45 sucessos totalizando 100% sucesso.

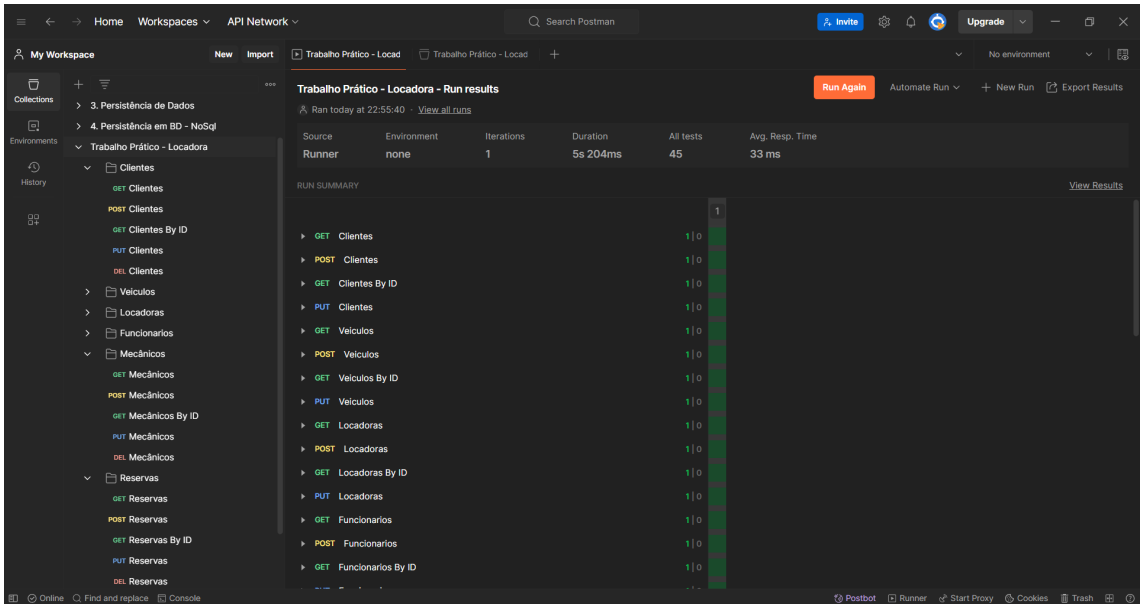

Figura 2. Execução dos testes positivos automatizados

7. Resultados e Conclusões

O sistema se comportou como esperado sendo capaz do gerenciamento e persistência dos dados. Os requisitos elaborados foram totalmente satisfeitos ao final do desenvolvimento e aplicação não demonstrou problemas como lentidão, erros de compilação ou exceções não tratadas. Dado isso é possível concluir que o sistema se demonstrou capaz de atender as necessidades demandadas por uma locadora de veículos.

8. Considerações Finais

A aplicação final ficou bastante completa e conseguiu, apesar de ser simples, cobrir uma grande quantidade de casos de uso. Poderia ser implementado soluções para deleção lógica, evitando perda de dados em caso de deleção, poderíamos implementar outras funcionalidades como requisições para autenticação de clientes, funcionários e administradores por exemplo.

Referências

Masse, M. (2011). *REST API Design Rulebook*. O'Reilly Media, Sebastopol, CA, 1 edition.

Reenskaug, T. (2001). Model-view-controller: A pattern for building user interfaces. *Expert Object-Oriented Programming*.

9. Apêndices

9.1. Diagrama UML de classes

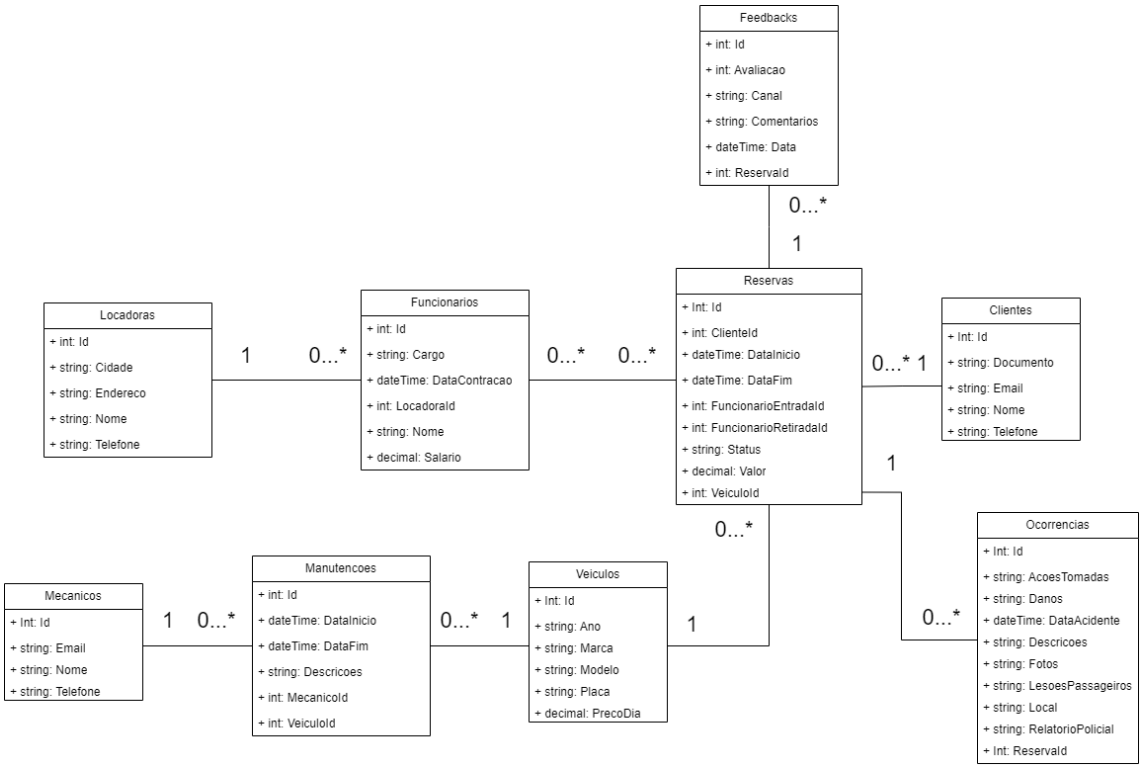

Figura 3. Diagrama UML