Relatório do Trabalho Prático 1
Tecnologias para Análise e Desenvolvimento de Sistemas

Lêda L. B. do Val[1]

[1]Pontifícia Universidade Católica de Minas Gerais (PUC Minas)

{ledadoval}2002@gmail.com

Abstract. *This project is a vehicle rental system developed in C'#' using Entity Framework for object-relational mapping and SQL Express as the database. The system provides a comprehensive solution for efficient and intuitive management of vehicles, customers, and reservations. Its implementation aims to streamline the vehicle rental process, providing a seamless experience for both customers and rental agency staff.*

Resumo. *Esse projeto se trata de um sistema de locadora de veículos desenvolvido em C'#' com o uso do Entity Framework para mapeamento objeto-relacional e o SQL Express como banco de dados. O sistema oferece uma solução abrangente para o gerenciamento eficiente e intuitivo de veículos, clientes e reservas. Sua implementação visa facilitar o processo de aluguel de veículos, proporcionando uma experiência fluida tanto para os clientes quanto para os funcionários da locadora.*

1. Introdução

As empresas de locação de veículo estão presentes na nossa sociedade e desempenham um papel crucial na mobilidade urbana atual. Com a crescente demanda de locação de veiculos, o desenvolvimento de sistemas eficientes para gestão desse negócio tornou-se primordial. Em resposta a essa demanda, surge o projeto LocaVeículas como uma solução abrangente projetada para otimizar o processo de aluguel de veículos.

Este projeto, desenvolvido usando C'#' e aproveitando tecnologias como Entity Framework e SQL Express, visa revolucionar a maneira como os serviços de aluguel de veículos são gerenciados. Ao fornecer ferramentas intuitivas para gerenciar veículos, clientes e reservas, o LocaVeículas busca aprimorar a experiência geral de aluguel tanto para os clientes quanto para o pessoal da agência de aluguel.

Neste trabalho, exploramos as características, implementação e benefícios potenciais do sistema LocaVeículas, investigando suas capacidades em otimizar operações e melhorar a satisfação do cliente dentro da indústria de aluguel de veículos.

1.1. Problema

Nesse cenário de gestão de uma locadora de automóveis, detectamos um problema em comum: gestão manual e desorganizada das operações da empresa. Tarefas como registro de locações, controle de estoque de veículos e acompanhamento de clientes eram realizadas de forma manual, o que resultava em erros, inconsistências nos dados e lentidão nos processos. Além disso, a falta de uma visão integrada e análise de dados dificultava a identificação de tendências de mercado e oportunidades de otimização.

1.2. Objetivos

O objetivo do LocaVeiculos é desenvolver uma plataforma eficiente para o gerenciamento completo das operações relacionadas à locação de veículos. Isso inclui o cadastro de veículos disponíveis para locação, registro de clientes, controle de reservas e acompanhamento do status dos veículos. Nosso foco é oferecer uma solução completa e integrada para a gestão de uma locadora de veículos, contribuindo para a eficiência operacional e o sucesso do negócio.

1.3. Justificativas

O LocaVeiculos surge através da necessidade de modernizar e otimizar os processos operacionais de empresas de locação de veículos. A implementação desse sistema visa automatizar tarefas manuais, melhorar a eficiência operacional, aumentar a precisão e confiabilidade dos dados, proporcionar uma experiência mais satisfatória para os clientes e fornecer insights valiosos para tomada de decisões estratégicas. Essas melhorias são essenciais para manter a competitividade no mercado e garantir o sucesso do negócio a longo prazo.

1.4. Público-alvo

O público-alvo do LocaVeiculos inclui tanto a equipe interna da empresa quanto os clientes que utilizam os serviços da locadora.

Equipe Interna:

- Gerentes e supervisores responsáveis pela gestão da locadora.
- Funcionários responsáveis pelo atendimento aos clientes, reservas de veículos e manutenção da frota.
- Equipe financeira encarregada do controle de pagamentos e faturamento.
- Administradores de sistema responsáveis pela manutenção e atualização do sistema.

Clientes da Locadora:

- Indivíduos que desejam alugar veículos para viagens, trabalho ou necessidades pessoais.
- Empresas que necessitam de veículos para fins comerciais ou transporte de funcionários.
- Turistas que visitam a região e precisam de meios de transporte temporários.

2. Requisitos

2.1. Requisitos Funcionais

Gerenciamento de Veículos:

- Cadastro de novos veículos no sistema, incluindo informações como modelo, marca, ano, placa, situação, tipo de combustível e tipo do veículo.
- Atualização e remoção de informações de veículos existentes.
- Visualização e busca rápida de veículos disponíveis para locação.

Gerenciamento de Clientes:

- Registro de novos clientes, incluindo detalhes como nome, telefone, endereço, email, se já cliente e quantos veiculos ele já alugou.
- Edição e exclusão de informações de clientes.
- Possibilidade de manter registros de clientes frequentes para oferecer descontos ou promoções especiais, tendo em vista que registramos a frequência de locação de carros.

Reservas:

- Funcionalidade para criar novas reservas, especificando o veículo desejado, o período de locação, se irá estender o período de locação e o valor final.
- Visualização e gerenciamento de reservas existentes, incluindo a capacidade de editar ou cancelar reservas.

2.2. Requisitos Não Funcionais

Desempenho:

- O sistema deve ser responsivo e rápido, mesmo com um grande volume de dados e usuários simultâneos.
- O tempo de resposta para consultas e operações do sistema deve ser mínimo, garantindo uma experiência eficiente para os usuários.

Escalabilidade

- Capacidade de expansão do sistema para lidar com um aumento na demanda de usuários e crescimento da frota de veículos ao longo do tempo.
- Arquitetura flexível que permite adicionar novos recursos e funcionalidades conforme necessário.

Confiabilidade

- Garantia de disponibilidade do sistema durante o horário comercial, minimizando o tempo de inatividade não planejado.

3. Arquitetura do Sistema

3.1. Descrição da Arquitetura

A arquitetura do sistema foi desenvolvida seguindo o padrão MVC (Model-View-Controller) para garantir uma separação clara de responsabilidades entre os diferentes componentes.

Essa abordagem garante uma divisão clara de responsabilidades e facilita a manutenção e escalabilidade do sistema.

3.2. Separação interna do projeto

A divisão de responsabilidades entre as camadas do sistema fi realizada da seguinte maneira:

- A camada de **Models** é responsável pela manipulação e persistência dos dados, incluindo a definição das entidades do banco de dados, mapeadas para classes C'#' usando o Entity Framework.

- A camada de **Context** é substituída por respostas JSON retornadas pelos endpoints da API. Essas respostas são serializações dos objetos retornados pelo modelo para representar os dados de forma estruturada.
- A camada de **Controller** é responsável por receber as requisições HTTP dos clientes, roteá-las para as ações apropriadas e retornar as respostas correspondentes. As controllers contêm a lógica de aplicação para processar as requisições e atualizar o modelo conforme necessário.

4. Implementação Técnica

4.1. Tecnologias Utilizadas

A implementação técnica do sistema foi realizada utilizando as seguintes tecnologias:

- **C'#' 8.0:** Linguagem de programação utilizada para o desenvolvimento da aplicação backend.
- **ASP.NET Core:** Framework utilizado para criar APIs web robustas e escaláveis.
- **Entity Framework:** ORM (Object-Relational Mapping) utilizado para mapear as entidades do banco de dados relacional para objetos em C'#'.
- **SQL Server:** Banco de dados relacional utilizado para armazenar e persistir os dados sobre as reservas de aluguel de veículos..
- **Swagger:** Ferramenta utilizada para documentar e testar APIs de forma interativa

4.2. Principais Classes e Responsabilidades

As principais classes do sistema e suas responsabilidades são:

- **ApplicationContext:** Classe responsável por configurar e fornecer acesso ao contexto do banco de dados utilizando o Entity Framework.
- **ClienteController:** Classe responsável por definir os endpoints da API relacionados aos clientes, incluindo operações CRUD (Create, Read, Update, Delete).
- **ReservaController:** Classe responsável por definir os endpoints da API relacionados às reservas de veículos.
- **VeiculoController:** Classe responsável por definir os endpoints da API relacionados aos veículos disponíveis para locação.
- **Outras classes de modelo:** Classes que representam as entidades do banco de dados, como Cliente, Reserva, Veiculo, entre outras. Essas classes contêm propriedades que correspondem aos campos das tabelas do banco de dados e métodos para manipular esses dados.

4.3. Escolhas de Design e Padrões de Desenvolvimento

Durante o desenvolvimento do sistema, foram adotadas as seguintes escolhas de design e padrões de desenvolvimento:

- **Padrão MVC:** O sistema foi estruturado seguindo o padrão MVC (Model-View-Controller) para garantir uma separação clara de responsabilidades entre os diferentes componentes da aplicação.
- **Uso do Entity Framework:** O Entity Framework foi escolhido para facilitar o acesso e manipulação dos dados do banco de dados, utilizando abordagens de Code First ou Database First, dependendo das necessidades do projeto.

- **API RESTful:** A API foi projetada seguindo os princípios RESTful, utilizando verbos HTTP para operações CRUD e URLs amigáveis para representar recursos.
- **Documentação com Swagger:** A documentação da API foi feita utilizando o Swagger, que permite gerar automaticamente uma documentação interativa com descrições detalhadas de cada endpoint, parâmetros necessários e exemplos de respostas.

Essas escolhas de design e padrões de desenvolvimento foram adotadas para garantir a robustez, escalabilidade e manutenibilidade do sistema.

5. Descrição do Banco de Dados

5.1. Modelo de Dados

O modelo de dados utilizado no sistema é composto por várias entidades relacionadas entre si. Abaixo está a descrição das principais tabelas do banco de dados:

- **Tabela Cliente:** Armazena informações sobre os clientes da locadora de veículos, como nome, telefone, endereço, email, se é um novo cliente, quantos veículos já alugou e o IDCliente.
- **Tabela Veiculo:** Contém dados sobre os veículos disponíveis para locação, incluindo marca, modelo, placa, ano, tipo de combustível, situação do veículo e IDVeiculo.
- **Tabela Reserva:** Registra as reservas de veículos feitas pelos clientes, incluindo data de início e fim da reserva, veículo reservado, cliente associado, valor total e se a reserva irá estender ou não.

5.2. Relacionamentos e Restrições de Integridade

Os relacionamentos entre as tabelas são definidos por chaves estrangeiras que garantem a integridade referencial. Algumas restrições de integridade comuns incluem:

- **Chave Primária:** Cada tabela possui uma chave primária única que identifica exclusivamente cada registro na tabela.
- **Chave Estrangeira:** As chaves estrangeiras são utilizadas para estabelecer relacionamentos entre as tabelas. Por exemplo, a tabela Reserva pode ter uma chave estrangeira que referencia o ID do cliente que fez a reserva e outra que referencia o ID do veículo reservado.

5.3. Exemplos de Consultas SQL

Abaixo estão alguns exemplos de consultas SQL relevantes para o sistema:

- **Consultar todos os clientes:**
```
SELECT QuantLocados FROM Cliente
WHERE Nome = 'Richard Montoya;
```

- **Consultar todos os veículos disponíveis para locação:**
```
SELECT * FROM Veiculo
WHERE SituacaoVeiculo = 'Disponível';
```

- **Consultar todas as reservas feitas por um cliente específico:**
```
SELECT * FROM Reserva WHERE IdCliente = 1981;
```

6. Documentação EndPoints

6.1. API Cliente

Esta API fornece endpoints para realizar operações CRUD (Create, Read, Update, Delete) em relação aos clientes no LocaVeiculo.

6.1.1. GET: api/Cliente

Retorna uma lista de todos os clientes cadastrados.

- **Método HTTP:** `GET`
- **Endpoint:** `GET /api/Cliente`
- **Descrição:** Retorna uma lista de todos os clientes cadastrados no sistema.
- **Resposta:** Uma lista de objetos JSON representando os clientes, ou um array vazio se não houver clientes cadastrados.
- **Código de Resposta:**
  - 200 OK: Requisição bem-sucedida.
  - 404 Not Found: Não foram encontrados clientes cadastrados.

6.1.2. GET: api/Cliente/{id}

Retorna os detalhes de um cliente específico com o ID fornecido.

- **Método HTTP:** `GET`
- **Endpoint:** `GET /api/Cliente/{id}`
- **Descrição:** Retorna os detalhes de um cliente específico com o ID fornecido.
- **Parâmetros:** `id` - O ID do cliente a ser recuperado.
- **Resposta:** Um objeto JSON representando o cliente com o ID fornecido.
- **Código de Resposta:**
  - 200 OK: Requisição bem-sucedida.
  - 404 Not Found: Cliente não encontrado.

6.1.3. PUT: api/Cliente/{id}

Atualiza os detalhes de um cliente existente com o ID fornecido.

- **Método HTTP:** `PUT`
- **Endpoint:** `PUT /api/Cliente/{id}`
- **Descrição:** Atualiza os detalhes de um cliente existente com o ID fornecido.
- **Parâmetros:** `id` - O ID do cliente a ser atualizado.
- **Corpo da Requisição:** Um objeto JSON contendo os novos detalhes do cliente.
- **Resposta:** Nenhuma resposta é retornada no corpo da resposta.
- **Código de Resposta:**
  - 204 No Content: Cliente atualizado com sucesso.
  - 400 Bad Request: O ID do cliente no corpo da requisição não corresponde ao ID fornecido nos parâmetros da URL.
  - 404 Not Found: Cliente não encontrado.

6.1.4. POST: api/Cliente

Cria um novo cliente com os detalhes fornecidos.

- **Método HTTP:** `POST`
- **Endpoint:** `POST /api/Cliente`
- **Descrição:** Cria um novo cliente com os detalhes fornecidos.
- **Corpo da Requisição:** Um objeto JSON contendo os detalhes do novo cliente a ser criado.
- **Resposta:** Um objeto JSON representando o novo cliente criado.
- **Código de Resposta:**
  - 201 Created: Cliente criado com sucesso.
  - 409 Conflict: Já existe um cliente com o mesmo email ou telefone.

6.1.5. DELETE: api/Cliente/{id}

Remove um cliente existente com o ID fornecido.

- **Método HTTP:** `DELETE`
- **Endpoint:** `DELETE /api/Cliente/{id}`
- **Descrição:** Remove um cliente existente com o ID fornecido.
- **Parâmetros:** `id` - O ID do cliente a ser removido.
- **Resposta:** Nenhuma resposta é retornada no corpo da resposta.
- **Código de Resposta:**
  - 204 No Content: Cliente removido com sucesso.
  - 404 Not Found: Cliente não encontrado.

6.2. API Reserva

Esta API fornece endpoints para realizar operações CRUD (Create, Read, Update, Delete) em relação as reservas do sistema.

6.2.1. GET: api/Reserva

Retorna uma lista de todas as reservas cadastradas.

- **Método HTTP:** `GET`
- **Endpoint:** `GET /api/Reserva`
- **Descrição:** Retorna uma lista de todas as reservas cadastradas no sistema.
- **Resposta:** Uma lista de objetos JSON representando as reservas, ou um array vazio se não houver reservas cadastradas.
- **Código de Resposta:**
  - 200 OK: Requisição bem-sucedida.
  - 404 Not Found: Não foram encontradas reservas cadastradas.

6.2.2. GET: api/Reserva/{id}

Retorna os detalhes de uma reserva específica com o ID fornecido.

- **Método HTTP:** `GET`
- **Endpoint:** `GET /api/Reserva/{id}`
- **Descrição:** Retorna os detalhes de uma reserva específica com o ID fornecido.
- **Parâmetros:** `id` - O ID da reserva a ser recuperada.
- **Resposta:** Um objeto JSON representando a reserva com o ID fornecido.
- **Código de Resposta:**
  - 200 OK: Requisição bem-sucedida.
  - 404 Not Found: Reserva não encontrada.

6.2.3. PUT: api/Reserva/{id}

Atualiza os detalhes de uma reserva existente com o ID fornecido.

- **Método HTTP:** `PUT`
- **Endpoint:** `PUT /api/Reserva/{id}`
- **Descrição:** Atualiza os detalhes de uma reserva existente com o ID fornecido.
- **Parâmetros:** `id` - O ID da reserva a ser atualizada.
- **Corpo da Requisição:** Um objeto JSON contendo os novos detalhes da reserva.
- **Resposta:** Nenhuma resposta é retornada no corpo da resposta.
- **Código de Resposta:**
  - 204 No Content: Reserva atualizada com sucesso.
  - 400 Bad Request: O ID da reserva no corpo da requisição não corresponde ao ID fornecido nos parâmetros da URL.
  - 404 Not Found: Reserva não encontrada.

6.2.4. POST: api/Reserva

Cria uma nova reserva com os detalhes fornecidos.

- **Método HTTP:** `POST`
- **Endpoint:** `POST /api/Reserva`
- **Descrição:** Cria uma nova reserva com os detalhes fornecidos.
- **Corpo da Requisição:** Um objeto JSON contendo os detalhes da nova reserva a ser criada.
- **Resposta:** Um objeto JSON representando a nova reserva criada.
- **Código de Resposta:**
  - 201 Created: Reserva criada com sucesso.
  - 409 Conflict: Já existe uma reserva com o mesmo ID.

6.2.5. DELETE: api/Reserva/{id}

Remove uma reserva existente com o ID fornecido.

- **Método HTTP:** `DELETE`

- **Endpoint:** `DELETE /api/Reserva/{id}`
- **Descrição:** Remove uma reserva existente com o ID fornecido.
- **Parâmetros:** `id` - O ID da reserva a ser removida.
- **Resposta:** Nenhuma resposta é retornada no corpo da resposta.
- **Código de Resposta:**
  - 204 No Content: Reserva removida com sucesso.
  - 404 Not Found: Reserva não encontrada.

6.3. API Veículo

Esta API fornece endpoints para realizar operações CRUD (Create, Read, Update, Delete) sobre os veículos no sistema da LocaVeiculos.

6.3.1. GET: api/Veiculo

Retorna uma lista de todos os veículos cadastrados.

- **Método HTTP:** `GET`
- **Endpoint:** `GET /api/Veiculo`
- **Descrição:** Retorna uma lista de todos os veículos cadastrados no sistema.
- **Resposta:** Uma lista de objetos JSON representando os veículos, ou um array vazio se não houver veículos cadastrados.
- **Código de Resposta:**
  - 200 OK: Requisição bem-sucedida.
  - 404 Not Found: Não foram encontrados veículos cadastrados.

6.3.2. GET: api/Veiculo/{id}

Retorna os detalhes de um veículo específico com o ID fornecido.

- **Método HTTP:** `GET`
- **Endpoint:** `GET /api/Veiculo/{id}`
- **Descrição:** Retorna os detalhes de um veículo específico com o ID fornecido.
- **Parâmetros:** `id` - O ID do veículo a ser recuperado.
- **Resposta:** Um objeto JSON representando o veículo com o ID fornecido.
- **Código de Resposta:**
  - 200 OK: Requisição bem-sucedida.
  - 404 Not Found: Veículo não encontrado.

6.3.3. PUT: api/Veiculo/{id}

Atualiza os detalhes de um veículo existente com o ID fornecido.

- **Método HTTP:** `PUT`
- **Endpoint:** `PUT /api/Veiculo/{id}`
- **Descrição:** Atualiza os detalhes de um veículo existente com o ID fornecido.
- **Parâmetros:** `id` - O ID do veículo a ser atualizado.
- **Corpo da Requisição:** Um objeto JSON contendo os novos detalhes do veículo.
- **Resposta:** Nenhuma resposta é retornada no corpo da resposta.

- **Código de Resposta:**
  - 204 No Content: Veículo atualizado com sucesso.
  - 400 Bad Request: O ID do veículo no corpo da requisição não corresponde ao ID fornecido nos parâmetros da URL.
  - 404 Not Found: Veículo não encontrado.

6.3.4. POST: api/Veiculo

Cria um novo veículo com os detalhes fornecidos.

- **Método HTTP:** `POST`
- **Endpoint:** `POST /api/Veiculo`
- **Descrição:** Cria um novo veículo com os detalhes fornecidos.
- **Corpo da Requisição:** Um objeto JSON contendo os detalhes do novo veículo a ser criado.
- **Resposta:** Um objeto JSON representando o novo veículo criado.
- **Código de Resposta:**
  - 201 Created: Veículo criado com sucesso.
  - 409 Conflict: Já existe um veículo com a mesma placa.

6.3.5. DELETE: api/Veiculo/{id}

Remove um veículo existente com o ID fornecido.

- **Método HTTP:** `DELETE`
- **Endpoint:** `DELETE /api/Veiculo/{id}`
- **Descrição:** Remove um veículo existente com o ID fornecido.
- **Parâmetros:** `id` - O ID do veículo a ser removido.
- **Resposta:** Nenhuma resposta é retornada no corpo da resposta.
- **Código de Resposta:**
  - 204 No Content: Veículo removido com sucesso.
  - 404 Not Found: Veículo não encontrado.

7. Resultados e Conclusões

Durante o processo de desenvolvimento do sistema LocaVeículos, analisamos todos os resultados visando fornecer uma experiência agradável para todos os usuários seguindo os requisitos e especificações do projeto.

- **Automação de Processos:** O sistema permite a automação de diversos processos manuais, como registro de clientes, gestão de reservas e controle de veículos disponíveis.
- **Melhoria da Precisão dos Dados:** Com a centralização de todas as informações em um único sistema e a implementação de validações de dados, houve uma melhoria significativa na precisão e integridade dos dados.

7.1. Avaliação da Conformidade dos Requisitos

Para a análise se todos os requisitos foram atendidos realizamos testes para observar as funcionalidades implementadas e tivemos os seguintes resultados:

- Todos os requisitos funcionais identificados foram devidamente implementados no sistema, o que indica que todas as principais funcionalidades esperadas estão disponíveis para os usuários.
- Os requisitos não funcionais, como desempenho, segurança e usabilidade, foram atendidos de acordo com as expectativas.
- O sistema foi capaz de lidar com cenários variados e situações de uso realista, demonstrando sua robustez e capacidade de suportar as necessidades da locadora de veículos.

7.2. Conclusões sobre a Eficácia do Sistema

A partir dos resultados obtidos e da avaliação da conformidade dos requisitos, concluimos que o LocaVeículos se mostra uma ferramenta eficaz para atender às necessidades da locadora de veículos. Dentre os fatores que nos levaram a essa conclusão, temos

- A automação de processos e a redução do tempo gasto em tarefas administrativas, facilitando e diminuindo erros.
- A precisão e integridade dos dados foram aprimoradas, proporcionando uma base sólida para a tomada de decisões informadas e estratégicas.
- A conformidade dos requisitos garantem sua capacidade de suportar as demandas das locadoras de veículo.

8. Wireframe

Para a construção do FrontEnd do sistema foi esquematizado wiraframes que auxiliaram na composição do site do LocaVeiculos. Utilizando o NodeJS, fizemos a conexão com o banco de dados que foi esquematizado e seguindo as rotas dos endpoints, deixamos o sistema funcional. Para a visualização do WireFrame através do link do Figma: link.

9. Considerações Finais

O sistema LocaVeículos, que foi desenvolvido para gerenciar locadoras de veículos, representa um avanço significativo na otimização de processos e na experiência do cliente. Utilizando tecnologias modernas, ele oferece uma solução robusta e escalável. Embora enfrentando desafios durante o desenvolvimento, como integração de componentes e garantia de desempenho, o LocaVeículos superou obstáculos e entregou um sistema eficiente.

Os benefícios do LocaVeículos são notáveis, automatizando processos manuais, melhorando a precisão dos dados e proporcionando agilidade no atendimento. Além disso, sua capacidade analítica oferece suporte à tomada de decisões estratégicas. Olhando para o futuro, vemos potencial para expansão e aprimoramento com a integração de novas tecnologias e feedback contínuo dos usuários.

Em suma, o LocaVeículos representa um passo significativo na evolução da indústria de aluguel de veículos, proporcionando benefícios tanto para empresas quanto para clientes, e contribuindo para um setor mais eficiente e centrado no cliente.

10. Referências

1. Template SBC Conferences: https://www.overleaf.com/latex/templates/sbc-conferences-template/blbxwjwzdngr
2. Documentação do ASP.NET Core: https://learn.microsoft.com/pt-br/aspnet/core/?view=aspnetcore-8.0