FIAP

Documentação do Sistema Oficina Virtual

DOMAIN DRIVEN DESIGN USING JAVA

Nome: Ana Carolina Reis Santana **RM:** 556219

Nome: Celina Alcantara **RM:** 558090

Nome: Leticia Zago **RM:** 558090

Índice

2. Objetivo e Escopo do Projeto

2.1 Objetivo Geral

O objetivo do projeto é desenvolver uma solução de oficina virtual que ofereça uma experiência completa para os usuários, permitindo a realização de auto diagnósticos de veículos de forma eficiente e acessível, com o suporte de um assistente virtual inteligente. A solução visa proporcionar uma ferramenta integrada para segurados da Porto Seguro e usuários não segurados, facilitando o diagnóstico de problemas nos veículos e sugerindo soluções de manutenção.

2.2 Escopo da Solução

A solução desenvolvida abrange um sistema web e uma aplicação para auto diagnóstico de veículos, onde os usuários podem interagir com um assistente virtual, alimentado por inteligência artificial, para identificar possíveis falhas mecânicas ou elétricas nos veículos. O sistema possui uma área dedicada a segurados da Porto Seguro, que possuem benefícios adicionais, e outra área para usuários não segurados, que também têm acesso ao auto diagnóstico.

Além disso, o sistema inclui uma parte de backend em Java, que gerencia o relacionamento entre clientes e veículos, aplicando regras de negócio robustas para garantir que cada veículo esteja vinculado a um cliente válido. O banco de dados é alimentado com as informações geradas pelo sistema, permitindo a análise de dados e a geração de insights que podem ser utilizados para aprimorar ainda mais a experiência dos usuários.

2.3 Diferenciais do Projeto

- **Auto Diagnóstico com Inteligência Artificial:** Um assistente virtual inteligente ajuda os usuários a identificar problemas nos veículos e sugerir soluções, proporcionando uma experiência de auto diagnóstico personalizada e prática.
- **Área Dedicada para Segurados Porto e Usuários Não Segurados:** O sistema oferece funcionalidades específicas para segurados, permitindo acesso a benefícios adicionais, enquanto mantém uma experiência de alta qualidade para os usuários não segurados.
- **Gestão Completa de Clientes e Veículos:** Utilizando regras de negócio no backend desenvolvido em Java, a solução garante que os veículos sejam corretamente vinculados aos clientes, evitando duplicidades e assegurando a integridade dos dados.
- **Análises e Insights para Melhorar a Experiência do Usuário:** O backend permite a coleta e análise de dados gerados pelo sistema, possibilitando o desenvolvimento de insights para otimizar o serviço e proporcionar uma experiência mais eficaz para os usuários.

3.Principais Funcionalidades da Solução

3.1 Sistema de Auto Diagnóstico com Assistente Virtual

A funcionalidade central da solução é o sistema de auto diagnóstico de veículos, que é impulsionado por um assistente virtual inteligente. Este assistente utiliza algoritmos de inteligência artificial para analisar problemas relatados pelos usuários e fornecer diagnósticos precisos e recomendações de manutenção. O sistema é projetado para ser intuitivo e interativo, permitindo que os usuários descrevam os sintomas de seus veículos e recebam orientações detalhadas sobre possíveis causas e soluções.

3.2 Área para Segurados Porto

Para os usuários que são segurados Porto, a plataforma oferece uma área exclusiva com funcionalidades adicionais. Os segurados têm acesso a um conjunto de ferramentas e recursos específicos, como o rastreamento de sinistros, acompanhamento do status da apólice e acesso a ofertas e serviços personalizados. Essa área é projetada para integrarse perfeitamente com os processos da seguradora, proporcionando um suporte mais eficiente e personalizado aos clientes.

3.3 Área para Usuários Não Segurados

A solução também contempla uma área dedicada para usuários que não possuem seguro com a Porto. Nesta seção, os usuários podem acessar funcionalidades básicas de diagnóstico e manutenção, sem os recursos adicionais disponíveis para os segurados. A área oferece orientações gerais e recomendações para manutenção preventiva e correção de problemas, proporcionando um suporte essencial para a gestão e cuidado dos veículos.

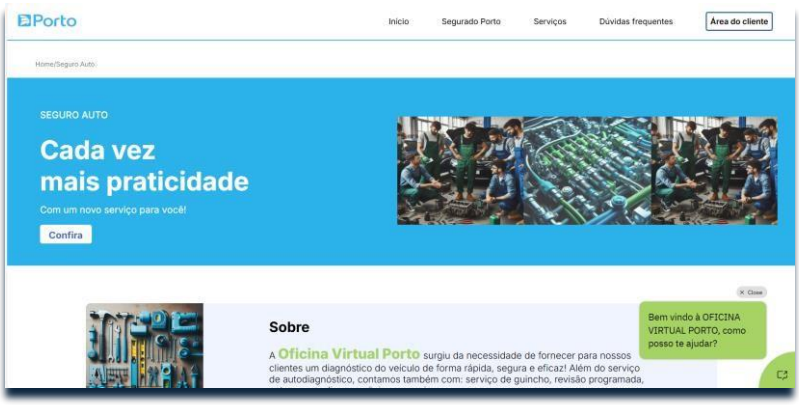
3.4 Vinculação de Veículos a Clientes (Java - Regras de Negócios)

No sistema desenvolvido em Java, foram implementadas regras de negócios específicas para garantir a correta vinculação de veículos aos clientes. Isso inclui a validação de que um veículo só pode ser cadastrado se estiver associado a um cliente existente e a prevenção de duplicidade de veículos para um mesmo cliente. Essas regras são gerenciadas por meio das camadas MVC e DAO, garantindo a integridade dos dados e a consistência das operações relacionadas a veículos e clientes.

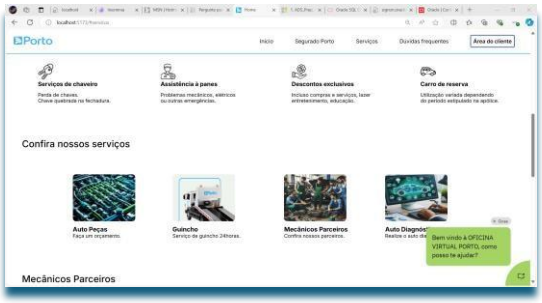
3.5 Backend para Análises e Geração de Insights

O backend da solução é responsável por alimentar o banco de dados com informações detalhadas sobre os diagnósticos e manutenções realizadas. Esse backend permite realizar análises abrangentes e gerar insights valiosos sobre o desempenho dos veículos e a eficácia das intervenções de manutenção. Com base nesses dados, a plataforma pode desenvolver relatórios e recomendações para melhorar a experiência do usuário e otimizar os processos de manutenção e suporte.

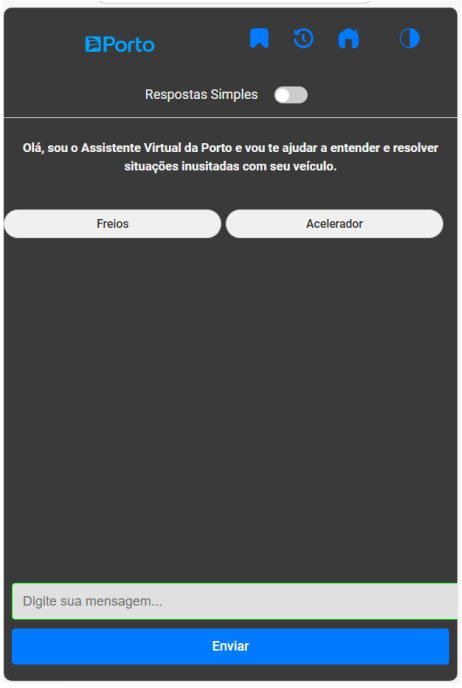
4.Protótipo



Página Inicial

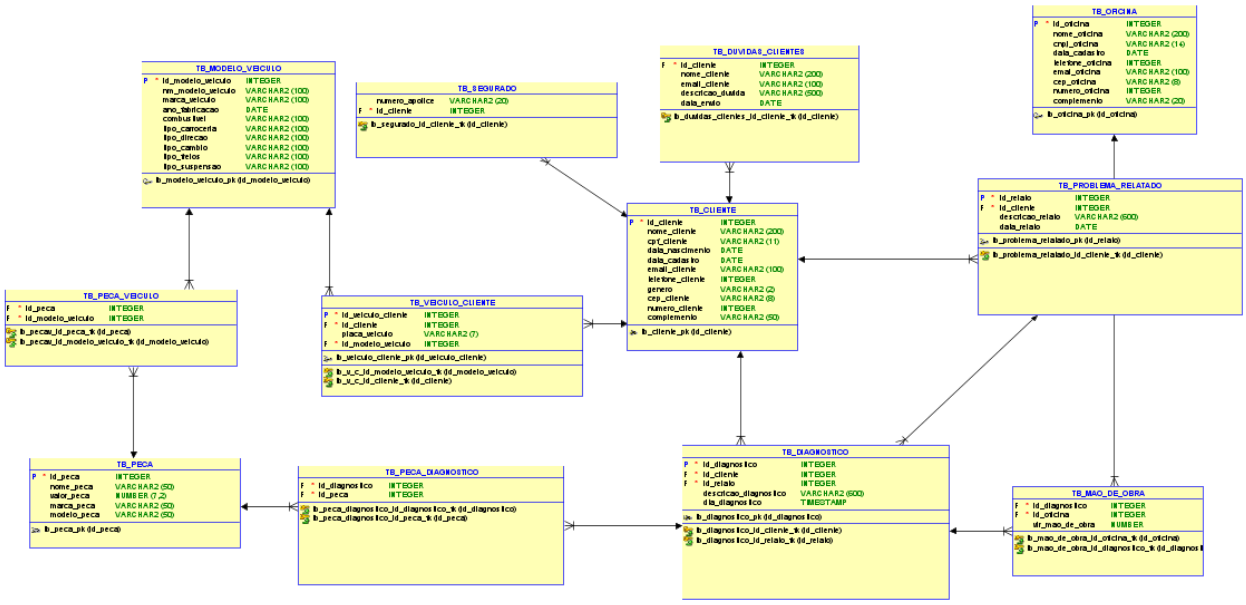


Area de Serviços Disponíveis

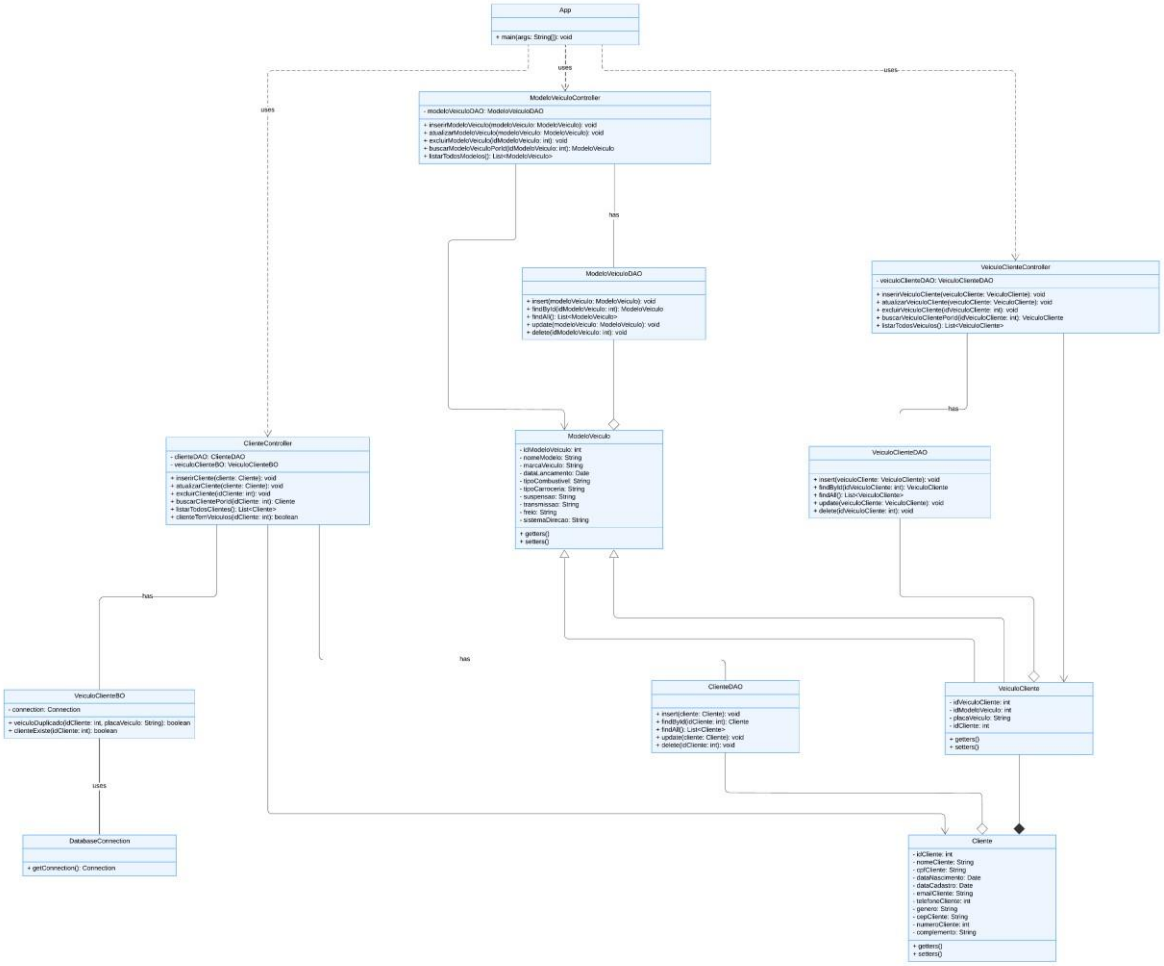


Assistente Virtual – Auto Diagnostico

5.Modelagem Banco de Dados



4.Diagrama de Classes



Procedimentos para Rodar a Aplicação

Regras de Negócios Implementadas

Vinculação de Veículos a Clientes:

Uma das principais regras de negócios é a vinculação correta de veículos aos clientes. Para garantir que um veículo só possa ser cadastrado se estiver associado a um cliente válido, foi implementada uma validação na camada de negócios (VeiculoClienteBO). Esta camada verifica se o id_cliente fornecido existe antes de permitir o cadastro ou a atualização de um veículo. Além disso, a lógica de negócios impede a duplicação de veículos para um mesmo cliente, garantindo a consistência dos dados.

Aqui estou considerando apenas os **endpoints** de Veiculos.

Estrutura do Código

A API é estruturada em várias camadas, cada uma com responsabilidades específicas:

1. **Modelo (Model)**: Define as entidades do sistema, como VeiculoCliente.
2. **Controle (Controller)**: Lida com a lógica de negócios e interação com a camada de acesso a dados.
3. **Business Object (BO)**: Contém regras de negócio e validações antes de chamar o controlador.
4. **Data Access Object (DAO)**: Realiza operações de banco de dados, como inserções, atualizações, exclusões e consultas.
5. **Recurso (Resource)**: Expõe os endpoints da API usando JAX-RS para manipulação de dados.

Endpoints da API

Método	Endpoint	Descrição	Exemplo de Requisição	Corpo da Requisição	Exemplo de Resposta
GET	`/veiculo`	Lista todos os veículos	`GET /veiculo`	N/A	200 OK, [{ "idVeiculoCliente": 1, "placaVeiculo": "XYZ1234", ... }, {...}]
GET	`/veiculo/{id}`	Busca um veículo por ID	`GET /veiculo/1`	N/A	200 OK, { "idVeiculoCliente": 1, "placaVeiculo": "XYZ1234", "idCliente": 1, "idModeloVeiculo": 2 }
POST	`/veiculo`	Cadastra um novo veículo	`POST /veiculo`	{ "idCliente": 1, "placaVeiculo": "XYZ1234", "idModeloVeiculo": 2 }	201 Created
PUT	`/veiculo/{id}`	Atualiza um veículo existente	`PUT /veiculo/1`	{ "idCliente": 1, "placaVeiculo": "XYZ5678", "idModeloVeiculo": 2 }	200 OK
DELETE	`/veiculo/{id}`	Deleta um veículo pelo ID	`DELETE /veiculo/1`	N/A	200 OK

Detalhes dos Endpoints

1. Listar Veículos

- **Endpoint**: /veiculo
- **Método**: GET
- **Descrição**: Retorna uma lista de todos os veículos cadastrados no sistema.

Exemplo de Requisição:

Exemplo de Resposta:

```json
[
    { "idVeiculoCliente": 1, "placaVeiculo": "XYZ1234", "idCliente": 1, "idModeloVeiculo":
    { "idVeiculoCliente": 2, "placaVeiculo": "ABC5678", "idCliente": 2, "idModeloVeiculo":
]
```

2. Buscar Veículo por ID

- **Endpoint**: /veiculo/{id}
- **Método**: GET
- **Descrição**: Retorna os detalhes de um veículo específico, identificado pelo seu ID.

Exemplo de Requisição:

Exemplo de Resposta:

```json
{
    "idVeiculoCliente": 1,
    "placaVeiculo": "XYZ1234",
    "idCliente": 1,
    "idModeloVeiculo": 2
}
```

Resposta de Erro (Veículo não encontrado):

```json
{
    "error": "Veículo não encontrado"
}
```

3. Cadastrar Novo Veículo

- **Endpoint**: /veiculo
- **Método**: POST
- **Descrição**: Cadastra um novo veículo no sistema. A requisição deve incluir as informações do veículo.

Exemplo de Requisição:

```bash
curl -X POST http://localhost:8080/veiculo \
-H "Content-Type: application/json" \
-d '{"idCliente": 1, "placaVeiculo": "XYZ1234", "idModeloVeiculo": 2}'
```

Exemplo de Resposta:

```json
{
    "message": "Veículo cadastrado com sucesso."
}
```

Resposta de Erro (Cliente não existe):

```json
{
    "error": "Cliente não existe."
}
```

4. Atualizar Veículo Existente

- **Endpoint**: /veiculo/{id}
- **Método**: PUT
- **Descrição**: Atualiza as informações de um veículo existente. O ID deve ser passado na URL, e o corpo deve conter os novos dados.

Exemplo de Requisição:

```bash
curl -X PUT http://localhost:8080/veiculo/1 \
-H "Content-Type: application/json" \
-d '{"idCliente": 1, "placaVeiculo": "XYZ5678", "idModeloVeiculo": 2}'
```

Exemplo de Resposta:

```json
{
    "message": "Veículo atualizado com sucesso."
}
```

Resposta de Erro (Veículo não encontrado):

```json
{
    "error": "Veículo não encontrado."
}
```

5. Deletar Veículo

- **Endpoint**: /veiculo/{id}
- **Método**: DELETE
- **Descrição**: Remove um veículo do sistema, usando seu ID.

Exemplo de Requisição:

```bash
curl -X DELETE http://localhost:8080/veiculo/1
```

Exemplo de Resposta:

```json
{
    "message": "Veículo deletado com sucesso."
}
```

Resposta de Erro (Veículo não encontrado):

```json
{
    "error": "Veículo não encontrado."
}
```