FIAP

Faculdade de Informática e Administração Paulista

Gabriel Augusto Fernandes - RM: 98986

Kauê Fernandes Braz - RM: 97768

Matheus Dantas de Sousa - RM: 98406

Matheus Giusto Lopes - RM: 99969

Thomas Nícolas de Melo Mendonça - RM: 99832

Solution Developers – ResQ AI

Trabalho desenvolvido na disciplina de Domain Driven Design, como exigência para obtenção de nota do Semestre em questão, sob orientação do Professor Fernando Luiz de Almeida.

SÃO PAULO

2º SEMESTRE DE 2023

Sumário

Descrição Projeto

A nossa proposta de solução visa a automatização do processo operacional humano por meio da utilização de bots, com o intuito de aprimorar a eficiência e precisão do atendimento ao cliente. Para tal, utilizamos tecnologias de reconhecimento de fala, por meio de processamento de linguagem natural, e de reconhecimento de imagem, por meio de machine learning.

O bot de reconhecimento de fala é capaz de compreender o que o cliente está dizendo, por meio do uso de técnicas de processamento de linguagem natural, e extrair informações relevantes sobre o sinistro. O bot é treinado para reconhecer os termos e frases mais comuns utilizados no contexto do sinistro e pode fazer perguntas adicionais para esclarecer informações ambíguas. Com isso, é possível garantir que o bot entenda as necessidades do cliente de maneira mais precisa e eficiente.

Por sua vez, o reconhecimento de imagem por meio de machine learning permite que a IA realize a análise fotos ou vídeos do sinistro e identifique padrões que possam indicar a necessidade de um guincho. A IA é treinada continuamente com novos dados e feedbacks, o que possibilita a melhoria contínua da sua precisão ao longo do tempo.

A automação proposta tem como objetivo assegurar a correta interpretação das informações, reduzindo possíveis erros e aumentando a eficiência do processo. Além disso, a solução apresentada contribui para a diminuição do tempo de espera do cliente e para a melhoria da qualidade do serviço prestado, resultando em economia de tempo e dinheiro, redução de riscos e, consequentemente, maior satisfação dos clientes.

Funcionalidades:

Camada VO:

Guincho (Guincho.java):

- Representa uma entidade para guinchos.
- Contém atributos como id_guincho, modelo_guincho, altura_guincho_cm, largura_guincho_cm, etc.
- Atua como um objeto de valor no contexto da aplicação.

Apolice (Apolice.java):

- Representa uma entidade para apólices.
- Contém atributos como id_apolice, numero_apolice, tipo_apolice, e valor_apolice.
- Atua como um objeto de valor representando informações sobre apólices.

Prestador (Prestador.java):

- Representa uma entidade para prestadores.
- Contém atributos como id_prestador, nome_prestador, e cnpj_prestador.
- Atua como um objeto de valor representando informações sobre prestadores.

Segurado (Segurado.java):

- Representa uma entidade para segurados.
- Contém atributos como id_segurado, nome_segurado, cpfCnpj, email_segurado, etc.
- Atua como um objeto de valor representando informações sobre segurados.

Sinistro (Sinistro.java):

- Representa uma entidade para sinistros.
- Contém atributos como id_sinistro, descricao_sinistro, data_ocorrencia, etc.
- Atua como um objeto de valor representando informações sobre sinistros.

Veiculo (Veiculo.java):

- Representa uma entidade para veículos.
- Contém atributos como id_veiculo, marca_veiculo, modelo_veiculo, placa_veiculo, etc.
- Atua como um objeto de valor representando informações sobre veículos.

Camada BO:

GuinchoBO (GuinchoBO.java):

- Objeto de negócio para a entidade Guincho.
- Implementa regras de negócio, especificamente validando a unicidade da placa modal do guincho.

ApoliceBO (ApoliceBO.java):

- Objeto de negócio para a entidade Apolice.
- Implementa regras de negócio, especialmente validando a unicidade do número da apólice.

PrestadorBO (PrestadorBO.java):

- Objeto de negócio para a entidade Prestador.
- Implementa regras de negócio, como a validação de unicidade do CNPJ.

SeguradoBO (SeguradoBO.java):

- Objeto de negócio para a entidade Segurado.
- Implementa regras de negócio, particularmente validando a unicidade do CPF/CNPJ.

SinistroBO (SinistroBO.java):

- Objeto de negócio para a entidade Sinistro.
- Implementa regras de negócio específicas relacionadas a sinistros.

VeiculoBO (VeiculoBO.java):

- Objeto de negócio para a entidade Veiculo.
- Implementa regras de negócio, como a validação de unicidade da placa de veículo.

Camada DAO:

GuinchoRepository (GuinchoRepository.java):

- Repositório Spring Data JPA para a entidade Guincho.

ApoliceRepository (ApoliceRepository.java):

- Repositório Spring Data JPA para a entidade Apolice.

PrestadorRepository (PrestadorRepository.java):

- Repositório Spring Data JPA para a entidade Prestador.

SeguradoRepository (SeguradoRepository.java):

- Repositório Spring Data JPA para a entidade Segurado.

SinistroRepository (SinistroRepository.java):

- Repositório Spring Data JPA para a entidade Sinistro.

VeiculoRepository (VeiculoRepository.java):

- Repositório Spring Data JPA para a entidade Veiculo.

Camada Controller:

GuinchoController (GuinchoController.java):

- Controlador RESTful para lidar com operações CRUD relacionadas a entidades Guincho.

ApoliceController (ApoliceController.java):

- Controlador RESTful para lidar com operações CRUD relacionadas a entidades Apolice.

PrestadorController (PrestadorController.java):

- Controlador RESTful para lidar com operações CRUD relacionadas a entidades Prestador.

SeguradoController (SeguradoController.java):

- Controlador RESTful para lidar com operações CRUD relacionadas a entidades Segurado.

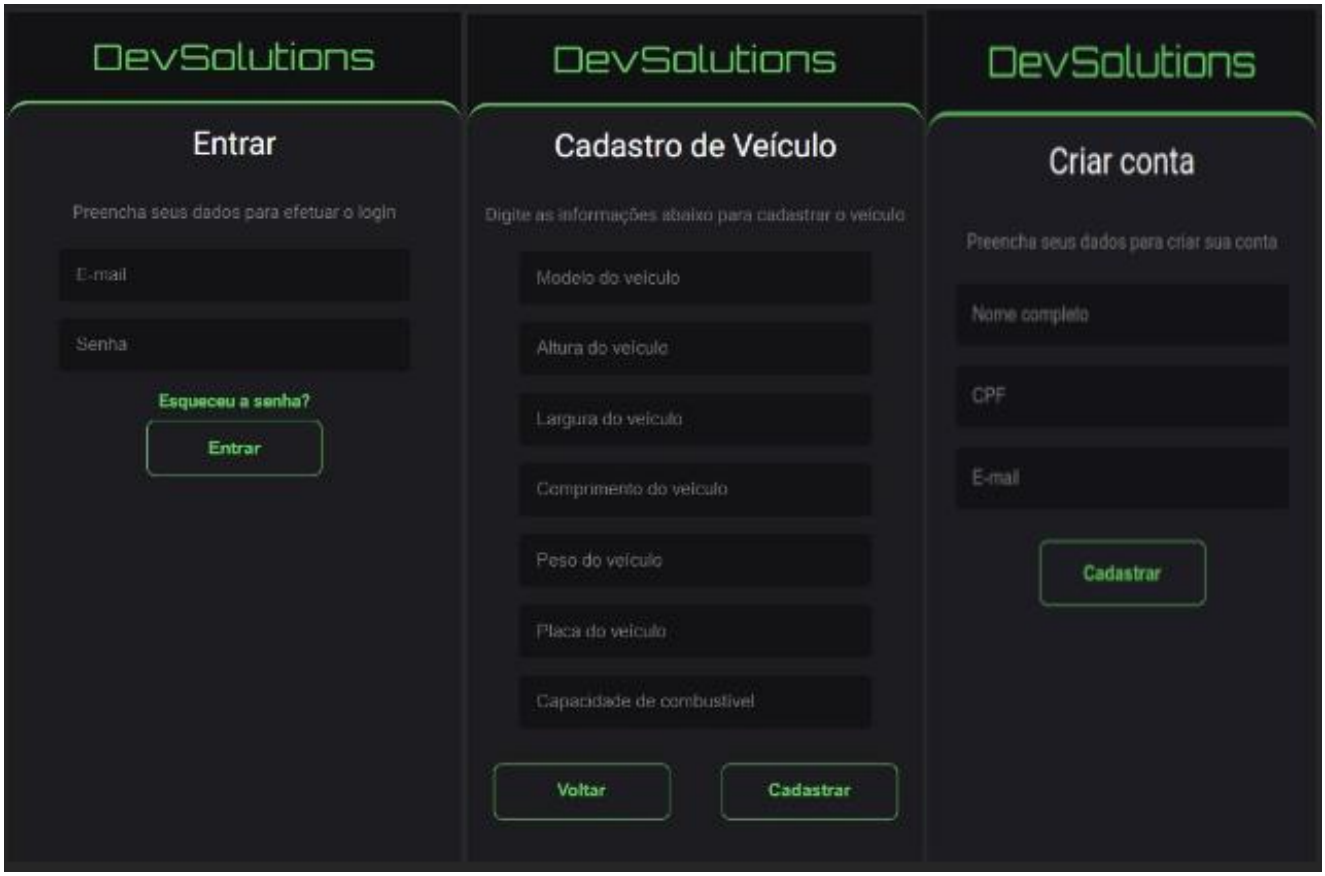
SinistroController (SinistroController.java):

- Controlador RESTful para lidar com operações CRUD relacionadas a entidades Sinistro.

VeiculoController (VeiculoController.java):

- Controlador RESTful para lidar com operações CRUD relacionadas a entidades Veiculo.

Protótipo



Modelo banco de dados

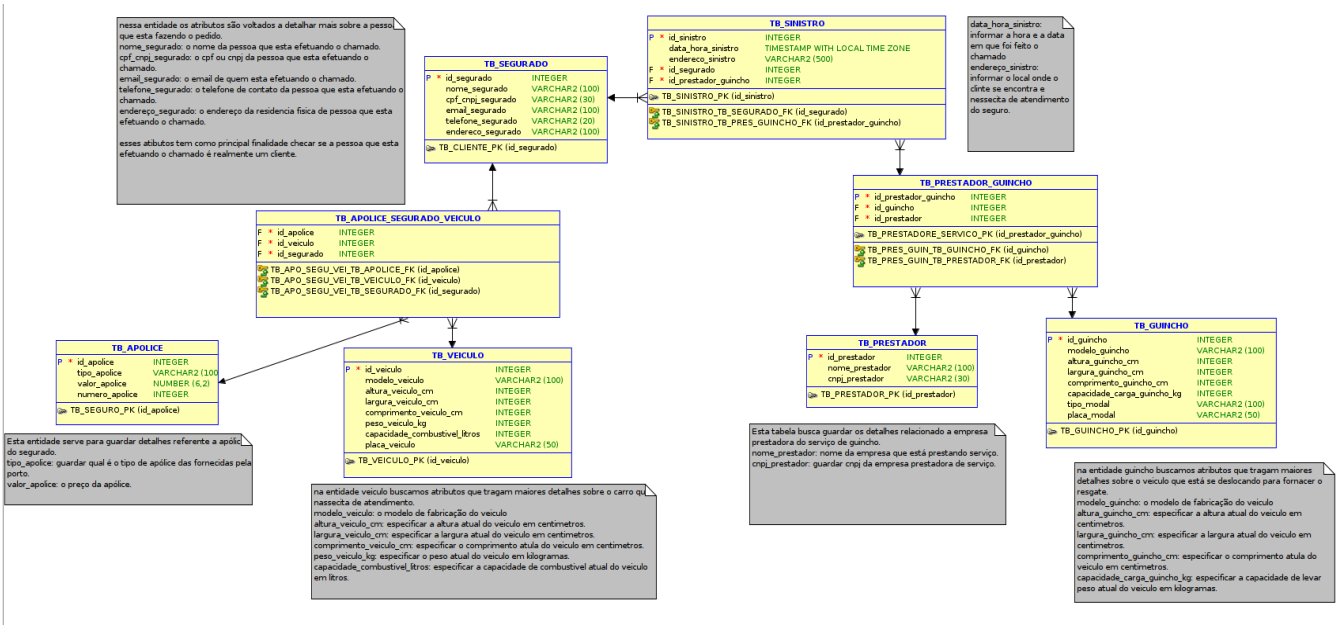
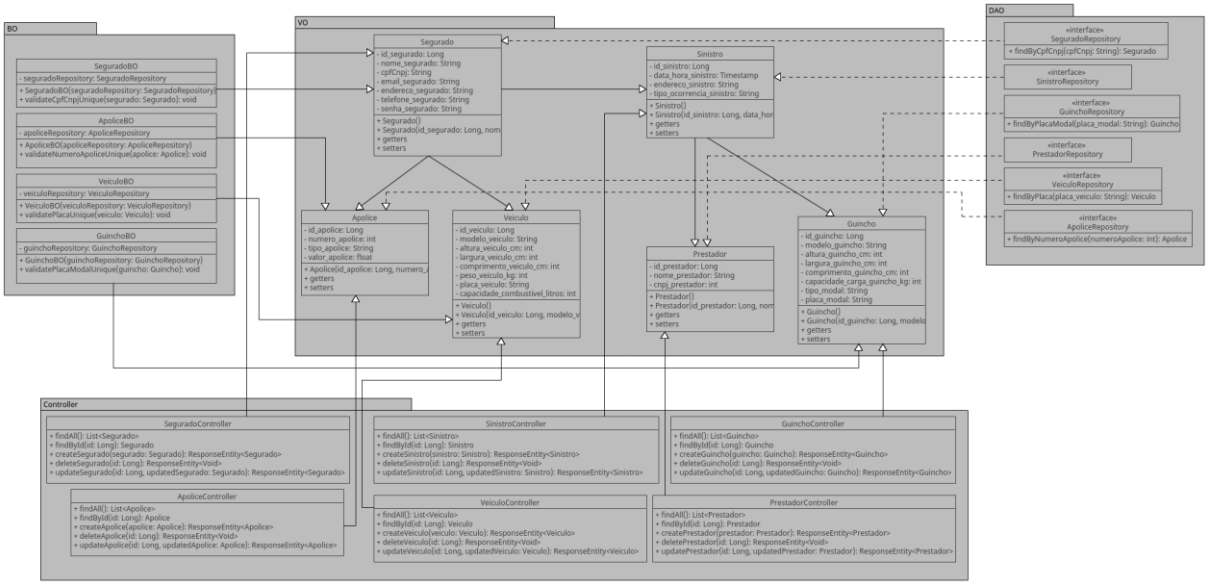


Tabela dos endpoints

URI	verbo HTTP	Status
http://localhost:8080/segurado	GET	200
http://localhost:8080/segurado/{id}		
http://localhost:8080/veiculo		
http://localhost:8080/veiculo/{id}	POST	201
http://localhost:8080/sinistro		
http://localhost:8080/sinistro/{id}		
http://localhost:8080/apolice	PUT	200
http://localhost:8080/apolice/{id}		
http://localhost:8080/guincho		
http://localhost:8080/guincho/{id}	DELETE	204
http://localhost:8080/prestador		
http://localhost:8080/prestador/{id}		

Modelagem UML



Procedimentos para rodar aplicação

Para rodar a aplicação, siga os passos abaixo:

1. Abra o projeto no seu ambiente Eclipse.

2. Localize a classe `PortoApplication.java` no pacote principal do projeto, em `src/main/java/com.example.porto/`.

3. Execute a classe `PortoApplication.java` clicando com o botão direito sobre ela e escolhendo a opção "Run".

4. Aguarde até que a aplicação seja inicializada. Você verá mensagens no console indicando que a aplicação está rodando.

5. Agora que a aplicação está em execução, você pode usar o POSTMAN para testar os endpoints.

6. Abra o POSTMAN e crie requisições para cada endpoint que você deseja testar. Utilize os métodos HTTP apropriados para cada operação:

- GET: Recuperar informações.
- POST: Criar novos recursos.
- PUT: Atualizar recursos existentes.
- DELETE: Excluir recursos.

7. Envie as requisições e observe as respostas do servidor. Verifique se as operações são realizadas corretamente.