



Faculdade de Informática e Administração Paulista

**Giovanna Revito Roz - RM558981**

**Kaian Gustavo de Oliveira Nascimento - RM558986**

**Lucas Kenji Kikuchi - RM554424**

**Domain Driven Design**

**PortoAutoTech**

**Sprint 2**

# INTEGRANTES

<b>RM</b> <b>(SOMENTE NÚMEROS)</b>	<b>NOME COMPLEMENTO</b> <b>(SEM ABREVIAR)</b>
558981	Giovanna Revito Roz
558986	Kaian Gustavo de Oliveira Nascimento
554424	Lucas Kenji Kikuchi



**SUMÁRIO**

1.Descrição do Projeto.....	5
2.Diagrama de Classes UML.....	7

## 1 – Descrição

A plataforma PortoAutoTech visa a implementação de uma interface para auxiliar clientes decorrentes e novos clientes da Porto Seguro, especificamente aqueles que possuem pouco ou nenhum conhecimento sobre problemas relacionados a carros. Um sistema tecnológico e inovador, que resolve de maneira mais fácil. Com mais precisão. Mais rapidez. Mais facilidade. O objetivo é que, até o fim do projeto, o sistema incorpore uma interface que permita ao cliente ter:

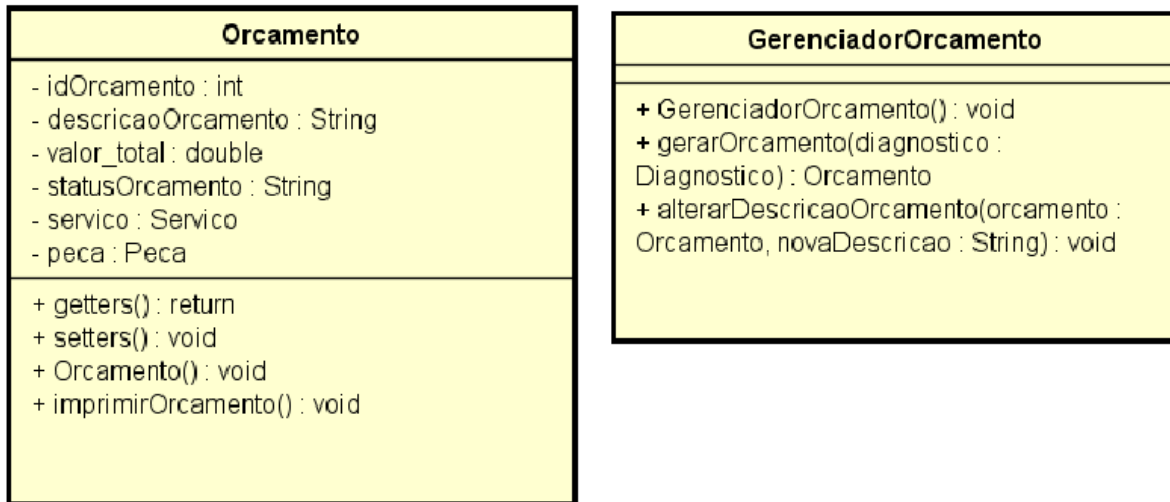
- um autodiagnóstico do problema apresentado pelo veículo através de uma I.A treinada através do Machine Learning em Python, incorporando no Chatbot;
- a possibilidade de se cadastrar com suas informações pessoais (Nome, telefone, e-mail etc.) e informações de seu veículo (modelo, marca, ano...) através de nosso aplicativo / website;
- a possibilidade de agendar um serviço através do Chatbot, informando o tipo de serviço, a oficina e a data;
- um pré orçamento, com um cálculo baseado no valor do serviço + valor das peças, e a estimativa de prazo de término do serviço definido com antecedência;
- a localização de oficinas mais próximas, informando a disponibilidade de peças e agendamento de determinados serviços com a integração com um banco de dados da oficina;
- mapeamento do carro, informando através de uma notificação automática quando uma manutenção preventiva deve ser feita, baseado na última vez que o cliente realizou um serviço no carro;
- notificação automática da disponibilidade de uma determinada peça, que será acionada quando a peça for registrada como disponível no banco de dados.
- interface que informa os pontos mais próximos de carregamento para carros elétricos;

- progresso da manutenção, visualizada através de uma barra de progresso, indicando cada mudança importante sobre o status da manutenção, conforme o serviço é feito (manualmente alterado pelo mecânico);
- reconhecimento de imagem através da I.A, que será responsável por analisar a imagem do problema enviado pelo usuário, retornando à solução mais plausível;
- comunicação através de voz (speech-to-text) com o Chatbot, permitindo descrever o problema oralmente ou enviar ruídos emitidos pelo carro, que serão analisados pela I.A;
- ligação em chamada com mecânico através do Chatbot, explicando o progresso da manutenção.

Através das funcionalidades descritas, o sistema será capaz de auxiliar pessoas que enfrentam dificuldades no entendimento dos problemas do veículo, além de fornecer diferenciais que auxiliarão em necessidades dos clientes que poucas empresas oferecem.

O sistema em Java, no caso, será responsável pela integração com o banco de dados, recebendo informações como dados de usuários, veículos, peças, agendamentos, diagnósticos e orçamentos.

## 2 – Diagrama de Classes UML



- **Orcamento**: classe Model que armazena informações sobre um orçamento (id, descrição, valor total, status, serviço e peça). Possui métodos getters e setters, construtor vazio e um método para imprimir os dados do orçamento;
- **GerenciadorOrcamento**: gerenciador da classe Orcamento, responsável por gerar um novo orçamento com base no diagnóstico;

Funcionario	GerenciadorFuncionario
<ul style="list-style-type: none"> <li>- matriculaFuncionario : int</li> <li>- nomeFuncionario : String</li> <li>- cargo : Cargo</li> <li>- centroAutomotivo : CentroAutomotivo</li> <li>- disponibilidadeFuncionario : boolean</li> <li>- horarioTrabalho : String</li> </ul>	<ul style="list-style-type: none"> <li>- funcionarios : ArrayList&lt;Funcionario&gt;</li> </ul>
<ul style="list-style-type: none"> <li>+ getters() : return</li> <li>+ setters() : void</li> <li>+ Funcionario(matriculaFuncionario : int, nomeFuncionario : String, cargo : Cargo, centroAutomotivo : CentroAutomotivo, disponibilidadeFuncionario : boolean, horarioTrabalho : String) : void</li> <li>+ imprimirFuncionario() : void</li> </ul>	<ul style="list-style-type: none"> <li>+ GerenciadorFuncionario() : void</li> <li>+ alterarCargo(funcionario : Funcionario, novoCargo : Cargo) : void</li> <li>+ alterarCentroAutomotivo(funcionario : Funcionario, novoCentroAutomotivo : CentroAutomotivo) : void</li> <li>+ alterarDisponibilidade(funcionario : Funcionario) : void</li> <li>+ alterarHorarioTrabalho(funcionario : Funcionario, horarioTrabalhoNovo : String) : void</li> <li>+ adicionarFuncionario(funcionario : Funcionario) : void</li> <li>+ removerFuncionario(funcionario : Funcionario) : void</li> <li>+ listarFuncionarios() : void</li> <li>+ retornaListaFuncionarios() : ArrayList&lt;Funcionario&gt;</li> </ul>

- **Funcionario**: classe Model que armazena informações sobre um funcionário (matrícula, nome, cargo, centro automotivo, disponibilidade e horário de trabalho). Possui métodos getters e setters, construtor parametrizado e um método para imprimir os dados do funcionário;
- **GerenciadorFuncionario**: gerenciador de Funcionario, possui um ArrayList de funcionários como atributo. Pode alterar o cargo, centro automotivo, disponibilidade e horário de trabalho do funcionário. Também pode adicionar e remover o funcionário da lista de funcionários, além de imprimir e retornar a lista.



Usuario	GerenciadorUsuario
<ul style="list-style-type: none"> <li>- cpfUsuario : String</li> <li>- nomeUsuario : String</li> <li>- senha : String</li> <li>- email : String</li> <li>- telefone : String</li> <li>- veiculos : ArrayList&lt;Veiculo&gt;</li> </ul>	<ul style="list-style-type: none"> <li>- usuarios : ArrayList&lt;Usuario&gt;</li> <li>- scanner : Scanner</li> <li>- usuarioLogado : Usuario</li> </ul>
<ul style="list-style-type: none"> <li>+ getters() : return</li> <li>+ setters() : void</li> <li>+ Usuario(cpfUsuario : String, nomeUsuario : String, senha : String, email : String, telefone : String) : void</li> <li>+ imprimirUsuario() : void</li> </ul>	<ul style="list-style-type: none"> <li>+ GerenciadorUsuario() : void</li> <li>+ cadastrar() : Usuario</li> <li>+ login() : void</li> <li>+ logout() : void</li> <li>+ adicionarVeiculoAoUsuario(veiculo : Veiculo, usuario : Usuario) : void</li> <li>+ removerVeiculoDoUsuario(veiculo : Veiculo, usuario : Usuario) : void</li> <li>+ imprimirUsuarios() : void</li> <li>+ retornaListaUsuarios() : ArrayList&lt;Usuario&gt;</li> <li>+ listarVeiculosDoUsuario(usuario : Usuario) : void</li> <li>+ getUsuarioLogado() : Usuario</li> <li>+ setUsuarioLogado() : void</li> </ul>

- **Usuario**: classe model que armazena informações sobre um usuário (CPF, nome, senha, e-mail, telefone e uma lista de veículos. Possui métodos getters e setters, um construtor parametrizado e um método para imprimir os dados do usuário;
- **GerenciadorUsuario**: gerenciador de Usuario, possui um ArrayList de usuários, o Scanner e um usuarioLogado com getters e setters.
  - cadastra um usuário, pedindo cpf, nome, senha, telefone e e-mail através de inputs.
  - loga o usuário, solicitando e-mail e senha. Caso os dados fornecidos coincidam com algum usuário cadastrado, autoriza o acesso. O atributo usuarioLogado é atualizado com os dados do usuário que acabou de entrar no sistema;
  - quando o usuário faz logout, o usuarioLogado é definido como null, permitindo um novo login;
  - adiciona e remove veículos da lista do usuário logado, além imprimir e retornar a lista de usuários, e imprimir a lista de veículos do usuário;

CentroAutomotivo	GerenciadorCentro
- idCentro : int - nomeCentro : String - enderecoCentro : String - telefoneCentro : String - horarioFuncionamento : String	- centrosAutomotivos : ArrayList<CentroAutomotivo>
+ getters() : return + setters() : void + CentroAutomotivo(idCentro : int, nomeCentro : String, enderecoCentro : String, telefoneCentro : String, horarioFuncionamento : String) : void + imprimirCentro() : void	+ GerenciadorCentro() : void + adicionarUnidade(centroAutomotivo : CentroAutomotivo) : void + removerUnidade(centroAutomotivo : CentroAutomotivo) : void + listarUnidades() : void + retornaListaCentros() : ArrayList<CentroAutomotivo> + retornaCentro(nomeCentro : String) : CentroAutomotivo

- **CentroAutomotivo**: classe Model que armazena informações sobre o centro automotivo (id, nome, endereço, telefone e horário de funcionamento). Possui métodos getters e setters, construtor parametrizado e um método para imprimir os dados do centro;
- **GerenciadorCentro**: gerenciador de CentroAutomotivo, possui um ArrayList de CentroAutomotivo como atributo. Pode adicionar e remover centros na lista, imprimir e retornar a lista, além de retornar um objeto CentroAutomotivo baseado em seu nome;

Veiculo	GerenciadorVeiculo
<ul style="list-style-type: none"><li>- marca : String</li><li>- modelo : String</li><li>- placa : String</li><li>- ano : int</li><li>- quilometragem : double</li></ul>	<ul style="list-style-type: none"><li>- veiculosTotais : ArrayList&lt;Veiculo&gt;</li><li>- scanner : Scanner</li></ul>
<ul style="list-style-type: none"><li>+ getters() : return</li><li>+ setters() : void</li><li>+ Veiculo() : void</li><li>+ Veiculo(marca : String, modelo : String, placa : String, ano : int, quilometragem : double) : void</li><li>+ imprimirVeiculo() : void</li></ul>	<ul style="list-style-type: none"><li>+ GerenciadorVeiculo() : void</li><li>+ criarVeiculo() : Veiculo</li><li>+ removerVeiculoDosCadastrados(veiculo : Veiculo) : void</li><li>+ listarVeiculosCadastrados() : void</li><li>+ buscarVeiculosPorMarca(marca : String) : ArrayList&lt;Veiculo&gt;</li><li>+ retornaListaVeiculos() : ArrayList&lt;Veiculo&gt;</li></ul>

- **Veiculo**: classe Model que armazena informações sobre um veículo (marca, modelo, placa, ano e quilometragem). Possui métodos getters e setters, construtor parametrizado e um método para imprimir os dados do veículo;
- **GerenciadorVeiculo**: gerenciador de Veiculo, possui um ArrayList de Veiculo e um Scanner como atributos. Pode criar um veículo, pedindo marca, modelo, placa, ano e quilometragem ao usuário através de inputs.

Peca	GerenciadorPecas
<ul style="list-style-type: none"><li>- idPeca : int</li><li>- disponibilidadePeca : boolean</li><li>- nomePeca : String</li><li>- precoPeca : double</li></ul>	<ul style="list-style-type: none"><li>- pecas : ArrayList&lt;Peca&gt;</li></ul>
<ul style="list-style-type: none"><li>+ getters() : return</li><li>+ setters() : void</li><li>+ Peca(idPeca : int, disponibilidadePeca : boolean, nomePeca : String, precoPeca : double) : void</li><li>+ imprimirPeca() : void</li></ul>	<ul style="list-style-type: none"><li>+ GerenciadorPecas() : void</li><li>+ alterarDisponibilidadePeca(peca : Peca) : void</li><li>+ adicionarPeca(peca : Peca) : void</li><li>+ removerPeca(peca : Peca) : void</li><li>+ listarPecas() : void</li><li>+ retornaListaPecas() : ArrayList&lt;Peca&gt;</li><li>+ consultarDisponibilidadePeca(pecaSolicitada : Peca) : void</li></ul>

- **Peca**: classe Model que armazena informações de uma peça (id, disponibilidade, nome e preço). Possui métodos getters e setters, construtor parametrizado e um método para imprimir os dados da peça;
- **GerenciadorPecas**: gerenciador de Peca, possui um ArrayList de Peca como atributo. Ele é responsável por alterar e consultar a disponibilidade da peça, adicionar ou remover uma peça da lista de peças e imprimir e retornar a lista;

Service	GerenciadorService
<ul style="list-style-type: none"><li>- idService : int</li><li>- tipoService : String</li><li>- descricaoService : String</li><li>- precoService : double</li><li>- duracaoService : int</li><li>- peca : Peca</li><li>- responsavel : Funcionario</li></ul>	<ul style="list-style-type: none"><li>- servicos : ArrayList&lt;Service&gt;</li></ul>
<ul style="list-style-type: none"><li>+ getters() : return</li><li>+ setters() : void</li><li>+ Service(idService : int, tipoService : String, descricaoService : String, precoService : double, duracaoService : int, peca : Peca) : void</li><li>+ imprimirService() : void</li></ul>	<ul style="list-style-type: none"><li>+ GerenciadorService() : void</li><li>+ adicionarService(servico : Service) : void</li><li>+ removerService(servico : Service) : void</li><li>+ adicionarResponsavel(funcionario : Funcionario, servico : Service) : void</li><li>+ listarServicos() : void</li><li>+ retornaListaServicos() : ArrayList&lt;Service&gt;</li><li>+ retornaService(descricao : String) : Service</li></ul>

- **Service**: classe Model que armazena informações de um serviço (id, tipo, descrição, preço, duração, peça e responsável). Possui métodos getters e setters, construtor parametrizado e um método que imprime os dados do serviço;
- **GerenciadorService**: gerenciador de Service, possui um ArrayList de Service como atributo. Ele é responsável por adicionar e remover um serviço da lista, pode adicionar um responsável (classe Funcionario) a um serviço, pode imprimir e retornar a lista de serviços, além de retornar um objeto Service baseado na descrição;

Cargo	GerenciadorCargo
<ul style="list-style-type: none"><li>- idCargo : int</li><li>- nomeCargo : String</li><li>- descricaoCargo : String</li><li>- areaCargo : String</li><li>- salarioCargo : double</li></ul>	<ul style="list-style-type: none"><li>- cargos : ArrayList&lt;Cargo&gt;</li></ul>
<ul style="list-style-type: none"><li>+ getters() : return</li><li>+ setters() : void</li><li>+ Cargo(idCargo : int, nomeCargo : String, descricaoCargo : String, areaCargo : String, salarioCargo : double) : void</li><li>+ imprimirCargo() : void</li></ul>	<ul style="list-style-type: none"><li>+ GerenciadorCargo() : void</li><li>+ adicionarCargo(cargo : Cargo) : void</li><li>+ removerCargo(cargo : Cargo) : void</li><li>+ listarCargos() : void</li><li>+ retornaListaCargos() : ArrayList&lt;Cargo&gt;</li><li>+ retornaCargo(nomeCargo : String) : Cargo</li></ul>

- **Cargo**: classe Model que armazena informações de um cargo (id, nome, descrição, área e salário). Possui métodos getters e setters, construtor parametrizado e um método para imprimir os dados do cargo;
- **GerenciadorCargo**: gerenciador de Cargo, possui um ArrayList de Cargo como atributo. Ele é responsável por adicionar e remover um cargo da lista, imprimir e retornar a lista de cargos, além de retornar um objeto Cargo baseado em seu nome;

Diagnostico	GerenciadorDiagnostico
- idDiagnostico : int - usuario : Usuario - veiculo : Veiculo - descricaoSintomas : String - solucao : Servico - orcamento : Orcamento - categoria : String - status : String  + getters() : return + setters() : void + Diagnostico(idDiagnostico : int, usuario : Usuario, veiculo : Veiculo, descricaoSintomas : String, solucao : Servico, categoria : String, status : String) : void + imprimirDiagnostico() : void	- diagnosticos : ArrayList<Diagnostico>  + GerenciadorDiagnostico() : void + adicionarDiagnostico(diagnostico : Diagnostico) : void + removerDiagnostico(diagnostico : Diagnostico) : void + retornaListaDiagnosticos() : ArrayList<Diagnostico> + listarDiagnosticos() : void

- **Diagnostico**: classe Model que armazena informações de um diagnóstico (id, usuário, veículo, descrição dos sintomas, solução, orçamento, categoria e status). Possui métodos getters e setters, construtor parametrizado e um método para imprimir os dados do diagnóstico;
- **GerenciadorDiagnostico**: gerenciador de Diagnostico, possui um ArrayList de Diagnostico como atributo. Ele é responsável por adicionar e remover diagnósticos da lista, além de imprimir e retornar a lista de diagnósticos.

Agendamento	GerenciadorAgendamento
- idAgendamento : String - data : String - hora : String - descricao : String - centro : CentroAutomotivo - servico : Servico - usuario : Usuario - veiculo : Veiculo	- agendamentos : ArrayList<Agendamento> - scanner : Scanner
+ getters() : return + setters() : void + Agendamento() : void + Agendamento(data : String, hora : String, descricao : String, centro : CentroAutomotivo, servico : Servico, usuario : Usuario, veiculo : Veiculo) : void + imprimirAgendamento() : void	+ GerenciadorAgendamento() : void + realizarAgendamento(gs : GerenciadorServico, gu : GerenciadorUsuario, usuario : Usuario, gc : GerenciadorCentro) : Agendamento + agendarComDiagnostico(diagnostico : Diagnostico, gu : GerenciadorUsuario, gc : GerenciadorCentro) : Agendamento + adicionarAgendamento(agendamento : Agendamento) : void + removerAgendamento(agendamento : Agendamento) : void + retornaListaAgendamentos() : ArrayList<Agendamento> + imprimirAgendamentos() : void

- **Agendamento**: classe Model que armazena informações de um agendamento (id, data, hora, descrição, centro automotivo, serviço, usuário e veículo). Possui métodos getters e setters, construtor vazio, construtor parametrizado e um método para imprimir os dados do agendamento;
- **GerenciadorAgendamento**: gerenciador de Agendamento, possui um ArrayList de Agendamento e um Scanner como atributos. Ele é responsável por criar um Agendamento, pedindo ao usuário a data, o horário, o serviço, o centro automotivo e o veículo através de inputs. O usuário também pode fazer um agendamento utilizando o diagnóstico obtido, precisando apenas informar a data, o horário e o centro automotivo. O gerenciador também pode adicionar e remover um agendamento da lista, além de imprimir e retornar a lista de agendamentos;