

Universidade de Brasília (UnB)  
Departamento de Ciência da Computação

# Tonhão Autopeças

Proposta de Projeto TP1

Grupo 08

Nirva Neves de Macedo, 232009585  
Henrique Morcelles Salum, 232003008  
Dérick Daniel Silva de Andrade, 231003522

Brasília, 21 de outubro de 2024

## Sumário

<b>1</b>	<b>Descrição do Problema</b>	<b>2</b>
1.1	Estrutura do Projeto . . . . .	2
<b>2</b>	<b>Regras de Negócio</b>	<b>3</b>
<b>3</b>	<b>Diagrama de Classes</b>	<b>3</b>
3.1	Classes do Sistema . . . . .	3
3.1.1	TipoDePeça . . . . .	3
3.1.2	Peça . . . . .	4
3.1.3	Estoque . . . . .	5
3.1.4	Veículo . . . . .	5
3.1.5	Serviço . . . . .	6
3.1.6	Pessoa . . . . .	6
3.1.7	Cliente . . . . .	7
3.1.8	Funcionário . . . . .	7
3.2	Classes Auxiliares . . . . .	8
3.2.1	Sessão de Usuário . . . . .	8
3.2.2	idGenerator . . . . .	9
3.2.3	Objetos . . . . .	10
<b>4</b>	<b>Telas</b>	<b>10</b>
4.1	Telas de Início . . . . .	10
4.2	Telas de Cadastro e Criação . . . . .	13
4.3	Telas de Gerenciamento . . . . .	15

# 1 Descrição do Problema

Oficinas mecânicas são essenciais para a manutenção da infraestrutura centrada em carros no Brasil. Infelizmente, são normalmente conhecidas por suas baixas confiabilidades e desorganização, especialmente quando estão afastadas de polos urbanos.

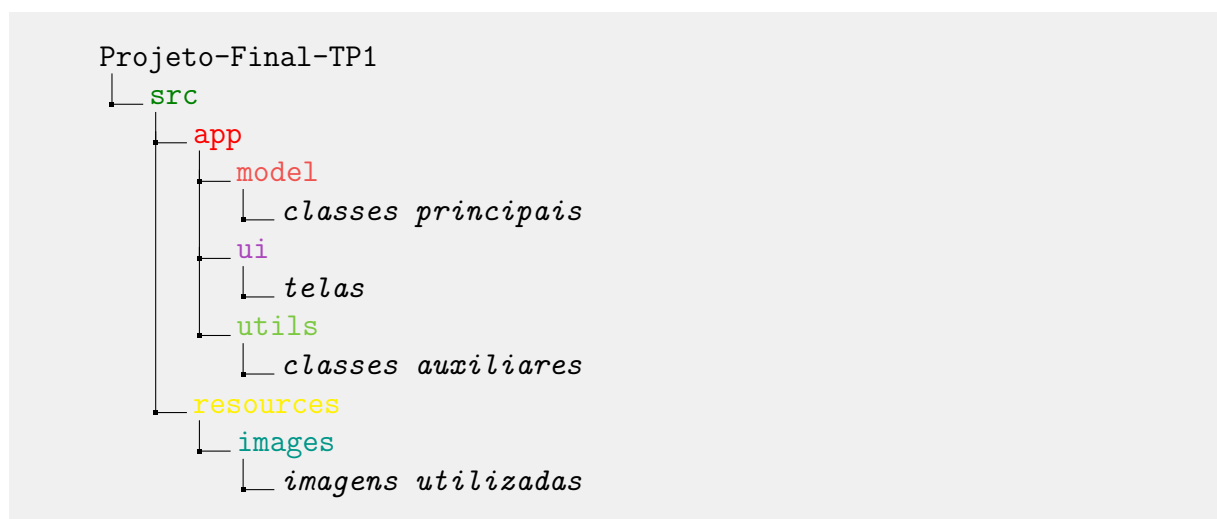
Visando uma solução moderna e simples para essas instituições, propomos um software de gerenciamento baseado em classes, com o objetivo de fornecer uma interface objetiva para gerenciar os seguintes elementos:

- Clientes
- Carros
- Estoque de peças
- Ordens de serviço

O sistema atribui as ordens de serviço ao cliente, carro e funcionário respectivos, provendo status de conclusão, status de pagamento e método de pagamento. Ele também gerencia o estoque ao solicitar novas peças e deduzir peças usadas em um serviço. O sistema automatiza a demanda de peças, verificando a disponibilidade no estoque e alocando-as automaticamente ao serviço solicitado.

## 1.1 Estrutura do Projeto

A parte relevante da estrutura do projeto foi organizada de acordo com a figura abaixo. É importante pontuar, porém, que alguns diretórios foram omitidos; esses são os criados e administrados pelo *NetBeans* ou utilizados para o versionamento pelo *git*.



**Figura 1:** Estrutura de diretórios do projeto.

## 2 Regras de Negócio

- Clientes que pagarem em dinheiro ou Pix recebem 5% de desconto.
- A cada R\$2.000,00 pagos, o cliente ganha um check-up completo do veículo e troca de óleo gratuitos.
- Funcionalidades de cadastrar, alterar e excluir objetos do programa requerem login de administrador.

## 3 Diagrama de Classes

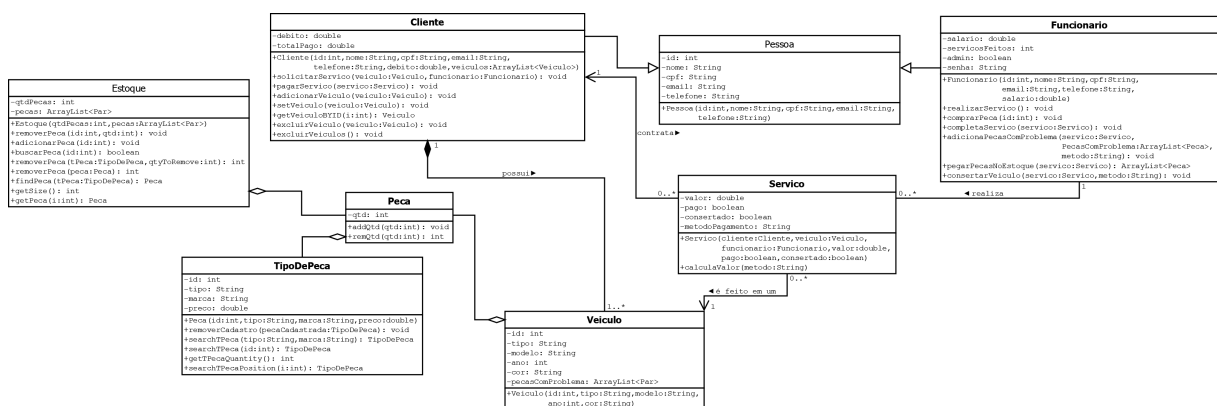


Figura 2: Diagrama de classes do sistema

### 3.1 Classes do Sistema

#### 3.1.1 TipoDePeca

A classe *TipoDePeca* fornece uma abstração para definir os tipos de peças reconhecidos e catalogados pela oficina. Ela possui um sistema de atribuição de códigos a cada novo tipo de peça cadastrado e permite busca por código ou pelos outros atributos: tipo e marca. Além disso, cada tipo de peça possui seu respectivo preço no catálogo. Todos os tipos de peças instanciados no programa são adicionados a uma lista estática da classe. Isso permite que as outras classes de fato usem *TipoDePeca* da forma correta: nenhuma delas (fora a classe que implementa a tela de gerenciamento dos tipos de peça) instancia um novo tipo de peça: na verdade, elas utilizam referências a tipos de peça já existentes. Isso facilita inúmeras operações - ao invés de comparar se diversos atributos de uma peça são os mesmos de outra, por exemplo, podemos simplesmente verificar se é a mesma referência.

Essa classe possui os atributos descritos a seguir:

- **tPecasCadastradas** (*static*): Uma lista estática das peças conhecidas.

- **idGenerator** (*static*): Um atributo utilizado pela própria classe que ajuda a fornecer o serviço de geração de código das peças.
- **freeIds** (*static*): Uma lista que carrega códigos previamente utilizados, mas que foram liberados para uso futuro.
- **id**: O código da peça.
- **tipo**: O tipo da peça.
- **marca**: A marca da peça.
- **preco**: O preço da peça, um *double*.

e os seguintes métodos (além dos getters e setters apropriados), que, com a exceção do construtor, são todos estáticos:

- **TipoDePeca()**: O construtor. Também executa toda a lógica de atribuição de código e reordena a lista.
- **removeCadastro()**: Remove o cadastro de um tipo de peça de acordo com uma referência ao objeto cadastrado.
- **searchTPeca()**: Um método com sobrecarga que permite pesquisar um tipo de peça na lista de acordo com seu código, ou tipo e marca.
- **getTPecaQuantity()**: Um getter especial que retorna a quantidade de peças cadastradas. Utilizado nas interfaces.
- **searchTPecaPosition()**: Um método de busca especial, também utilizado nas interfaces, que fornece uma forma de buscar uma peça pelo seu índice na lista.

### 3.1.2 Peça

A classe *Peca* é uma classe muito simples, que "embrulha" duas informações: o *TipoDePeca*, e a quantidade. Ela fornece uma abstração para representar as peças físicas. Ela possui os seguintes atributos:

- **tipoPeca**: O tipo da peça.
- **qtd**: A quantidade de peças.

além dos seguintes métodos:

- **Peca()**: O construtor padrão.
- **Peca(TipoDePeca tipoPeca, int qtd)**: O construtor que inicializa os atributos.
- **getTipoPeca()**: Retorna o tipo da peça.
- **getQtd()**: Retorna a quantidade de peças.
- **setQtd()**: Define a quantidade de peças.
- **addQtd()**: Adiciona uma quantidade de peças.
- **remQtd()**: Remove uma quantidade de peças.

### 3.1.3 *Estoque*

Estoque é uma classe abstrata pelo motivo de nunca ser instanciada (logo possui apenas membros estáticos), cuja função é autoexplicativa: é a representação do estoque da oficina. Ela possui um único atributo.

- **estoque**: Uma lista de peças.

Seus métodos, que proporcionam uma forma encapsulada de interação com essa lista, são os seguintes:

- **adicionarPeca()**: Um método que adiciona uma quantidade de uma peça ao estoque.
- **removerPeca()**: Um método sobrecarregado que pode remover uma determinada quantidade de uma peça do estoque utilizando um TipoDePeca e uma quantidade ou uma referência a um objeto Peca.
- **findPeca()**: Verifica se a peça está no estoque.
- **getSize()**: Retorna quantos tipos diferentes de peça há no estoque.
- **getPeca()**: Retorna a peça em um determinado índice da lista do estoque.

### 3.1.4 *Veículo*

A classe *Veiculo* modela os dados de um veículo. Uma lista dessas classes é um atributo da classe *Cliente*. Ela armazena as informações de id, ano, tipo, modelo, cor e placa, mas, principalmente, mantém uma lista de *pecasComProblema*, que é vazia até que o funcionário a popule pela tela de gerenciamento de serviços. A classe também oferece métodos para gerenciar os dados do veículo e manipular a lista de peças problemáticas, como adicionar peças com defeito e esvaziar a lista após os reparos. Essa estrutura permite uma organização eficiente dos diagnósticos e serviços realizados pela oficina.

A classe *Veiculo* possui os seguintes atributos:

- **id**: O código do veículo.
- **ano**: O ano do veículo.
- **tipo**: O tipo do veículo.
- **modelo**: O modelo do veículo.
- **cor**: A cor do veículo.
- **placa**: A placa do veículo.
- **pecasComProblema**: Uma lista de peças com defeito.

e os seguintes métodos:

- **Veiculo()**: O construtor padrão.

- **Veiculo(int ano, String tipo, String placa, String modelo, String cor):** O construtor que inicializa os atributos.
- **toString():** Retorna a placa do veículo.
- **esvaziarPecasComProblema():** Esvazia a lista de peças com defeito.

### 3.1.5 Serviço

A classe *Serviços*, por sua vez, é essencial para o sistema. Nela, quase todas as outras classes do projeto são instanciadas para que se realize o que esse sistema se propõe a fazer: permitir que se tenha controle sobre um reparo de um funcionário ao veículo de um cliente.

A classe *Servico* possui os seguintes atributos:

- **id:** O código do serviço.
- **valor:** O valor do serviço.
- **pago:** Um booleano que indica se o serviço foi pago.
- **consertado:** Um booleano que indica se o veículo foi consertado.
- **metodoPagamento:** O método de pagamento.
- **cliente:** O cliente que solicitou o serviço.
- **veiculo:** O veículo que será consertado.
- **funcionario:** O funcionário que realizará o serviço.

e os seguintes métodos:

- **Servico():** O construtor padrão.
- **Servico(Cliente cliente, Veiculo veiculo, Funcionario funcionario):** O construtor que inicializa os atributos.
- **calculaValor():** Calcula o valor do serviço.
- **adicionaPecasComProblema():** Adiciona peças com defeito ao veículo.

### 3.1.6 Pessoa

A classe *Pessoa* é uma classe abstrata que fornece a estrutura básica para representar pessoas no sistema. Ela é a superclasse de *Cliente* e *Funcionario*, que serão explicados doravante. A classe *Pessoa* possui os seguintes atributos:

- **id:** O código da pessoa.
- **nome:** O nome da pessoa.
- **cpf:** O CPF da pessoa.
- **email:** O e-mail da pessoa.
- **telefone:** O telefone da pessoa.

e os seguintes métodos, além dos getters e setters:

- **Pessoa(int id, String nome, String cpf, String email, String telefone):** O construtor que inicializa os atributos com id informado.
- **Pessoa(String nome, String cpf, String email, String telefone):** O construtor que inicializa os atributos.

### 3.1.7 *Cliente*

A classe *Cliente*, que herda a classe *Pessoa*, oferece a estrutura para armazenar os dados de um cliente no contexto de uma oficina mecânica. Ela mantém informações como nome, telefone, endereço e uma lista de veículos associados ao cliente. Os objetos dessa classe são utilizados para criar os serviços da oficina. Os atributos dessa classe são os que seguem, além dos atributos herdados da classe *Pessoa*:

- **debito:** O débito do cliente.
- **totalPago:** O total pago pelo cliente.
- **idGenerator:** atributo utilizado para criar o id do cliente quando ele é criado
- **idsLivres:** Uma lista dos ids livres para serem reutilizados
- **veiculos:** Uma lista de veículos associados ao cliente.

e os seguintes métodos, além dos getters e setters:

- **Cliente(String nome, String cpf, String email, String telefone):** O construtor que inicializa os atributos.
- **excluirVeiculo(Veiculo veiculo):** Exclui um veículo da lista de veículos.
- **excluirVeiculos():** Exclui todos os veículos da lista de veículos.

### 3.1.8 *Funcionário*

A classe *Funcionario*, que herda a classe *Pessoa*, implementa uma estrutura para simular um funcionário no sistema da oficina. Ela fornece os métodos necessários para gerenciar os serviços e peças. Os atributos dessa classe são os que seguem, além dos atributos herdados da classe *Pessoa*:

- **salario:** O salário do funcionário.
- **servicosFeitos:** A quantidade de serviços feitos pelo funcionário.
- **admin:** Um booleano que indica se o funcionário é administrador.
- **senha:** A senha do funcionário, utilizada para acessar o sistema.
- **servicosAtivos:** Um HashMap de serviços ativos, que mapeia o id do serviço ao serviço.
- **idsLivres:** Uma *priority queue* dos ids livres para serem reutilizados.



- **idGenerator**: Um atributo utilizado para criar o id do funcionário quando ele é criado.

e os seguintes métodos, além dos getters e setters:

- **Funcionario(double salario, String senha, int id, String nome, String cpf, String email, String telefone)**: O construtor que inicializa os atributos com id informado.
- **Funcionario(double salario, String senha, String nome, String cpf, String email, String telefone)**: O construtor que inicializa os atributos.
- **Funcionario(String senha, String nome, String cpf, String email, String telefone)**: O construtor que inicializa os atributos sem salário informado.
- **completaServico(Servico servico)**: Completa um serviço.
- **pegarPecasNoEstoque(Servico servico)**: Gerencia as peças necessárias para um serviço, identificando as peças faltantes no estoque e removendo as peças disponíveis.
- **consertarVeiculo(Servico servico, String metodo)**: Calcula o valor de um serviço, esvazia a lista de peças com problema do veículo associado e define o serviço como consertado.

## 3.2 Classes Auxiliares

### 3.2.1 Sessão de Usuário

A classe *SessaoUsuario* é uma classe auxiliar que fornece ao sistema uma lógica de sessão do usuário autenticado no momento. Ela mantém uma referência estática ao funcionário logado, que é criada quando o usuário faz a autenticação pela tela de Login. Os atributos dessa classe são os que seguem:

- **instancia**: Inicializa a única instância do usuário logado.
- **usuarioLogado**: O funcionário logado.

e os métodos são os seguintes:

- **SessaoUsuario()**: O construtor **privado**, para impedir a criação de múltiplas instâncias.
- **getInstancia()**: Retorna a instância do usuário logado.
- **setUsuarioLogado(Funcionario usuarioLogado)**: Define o funcionário logado.
- **getUsuarioLogado()**: Retorna o funcionário logado.

Para essa classe, forçaremos a implementação por aqui, para que ela seja melhor compreendida.

```
1 package app.utils;
2
3 import app.model.Funcionario;
4
5 public class SessaoUsuario {
6     private static SessaoUsuario instancia;
7     private Funcionario usuarioLogado;
8
9     private SessaoUsuario() {}
10
11     public static SessaoUsuario getInstancia() {
12         if (instancia == null) {
13             instancia = new SessaoUsuario();
14         }
15         return instancia;
16     }
17
18     public void setUsuarioLogado(Funcionario usuario) {
19         this.usuarioLogado = usuario;
20     }
21
22     public Funcionario getUsuarioLogado() {
23         return usuarioLogado;
24     }
25 }
```

**Código 1:** Implementação da classe SessaoUsuario

### 3.2.2 *idGenerator*

Essa classe foi utilizada por algumas classes para gerar códigos únicos para os objetos. Ela fornece um método estático que retorna um código único para cada objeto criado. O código é gerado a partir de um contador que é incrementado a cada chamada do método. A classe possui os seguintes atributos estáticos:

- **currentServicoId:** O contador dos ids de serviços.
- **currentVeiculoId:** O contador dos ids de veículos.

e os seguintes métodos:

- **generateServicoID()**: Gera um id único para um serviço.
- **generateVeiculoID()**: Gera um id único para um veículo.

Ambos *synchronized*, isto é, que podem ser acessados apenas por uma thread por vez, para evitar problemas de concorrência.

### 3.2.3 Objetos

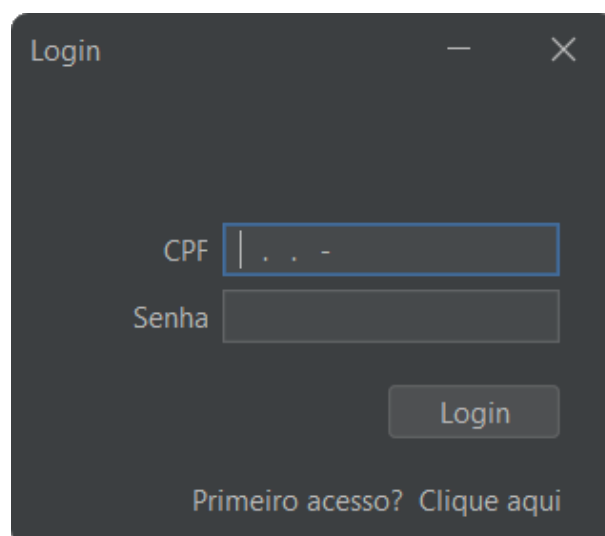
A classe *Objetos* é uma classe auxiliar que mantém *HashMaps* públicos e estáticos de todas as classes do sistema, armazenando os objetos criados de cada classe na interface de forma a simular o funcionamento real de um sistema com persistência durante a execução do programa. Além disso, nessa classe são criados, na execução do programa, os objetos padrão de cada classe, de forma a facilitar a visualização e o teste do sistema. Os atributos dessa classe são os que seguem:

- **clientes**: Um *HashMap* de clientes.
- **funcionarios**: Um *HashMap* de funcionários.
- **veiculos**: Um *HashMap* de veículos.
- **servicos**: Um *HashMap* de serviços.
- **placas**: Uma *ArrayList* de placas de veículos.

seus métodos são demasiados e uma descrição detalhada para cada um deles não é necessária, mas, em resumo, eles são responsáveis por criar os objetos de exemplo no sistema.

## 4 Telas

### 4.1 Telas de Início



**Figura 3:** Tela de login



Figura 4: Tela inicial

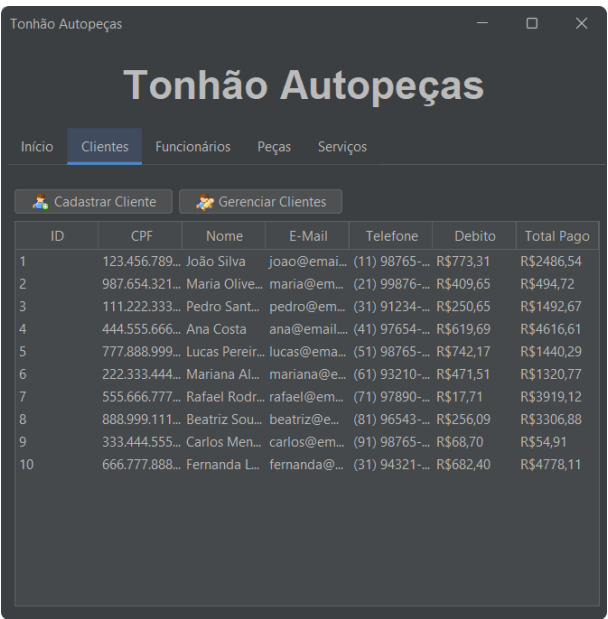


Figura 5: Tela inicial - Seção de clientes

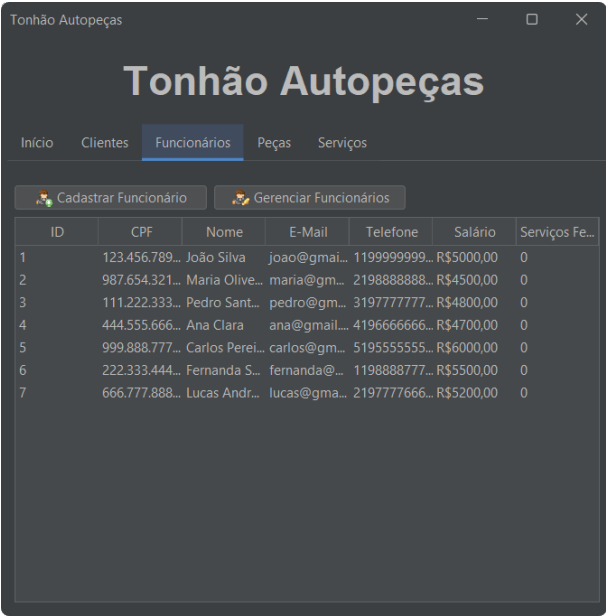


Figura 6: Tela inicial - Se  o de funcion rios

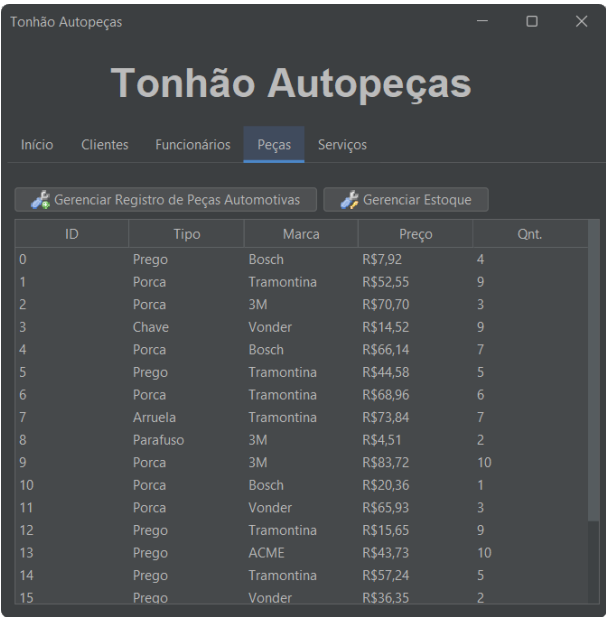
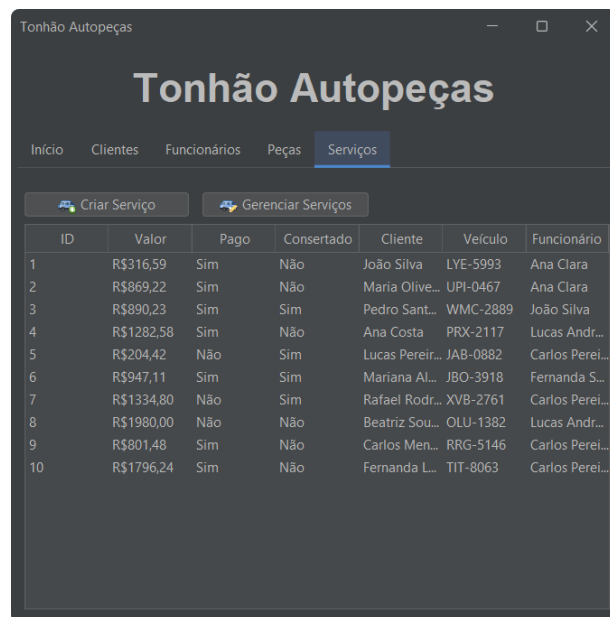


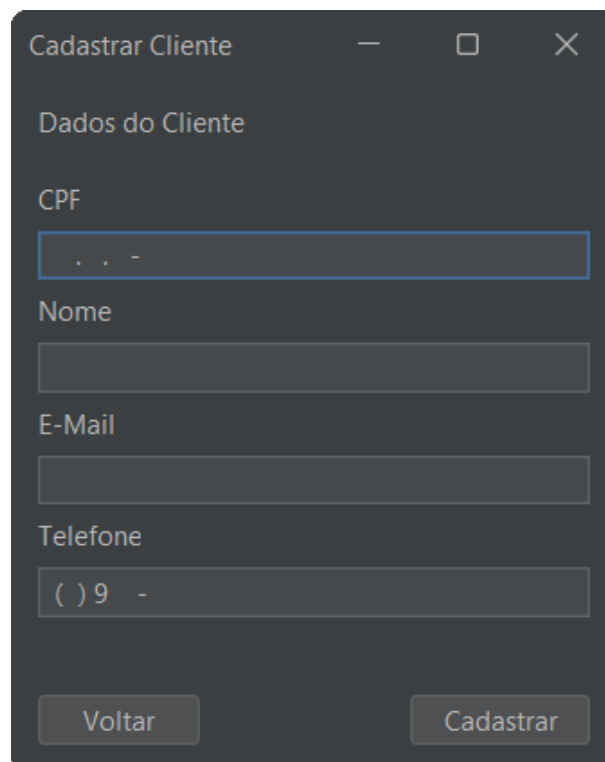
Figura 7: Tela inicial - Se  o de pe as



ID	Valor	Pago	Consertado	Cliente	Veículo	Funcionário
1	R\$316,59	Sim	Não	João Silva	LYE-5993	Ana Clara
2	R\$869,22	Sim	Não	Maria Olive...	UPI-0467	Ana Clara
3	R\$890,23	Sim	Sim	Pedro Sant...	WMC-2889	João Silva
4	R\$1282,58	Sim	Não	Ana Costa	PRX-2117	Lucas Andr...
5	R\$204,42	Não	Sim	Lucas Pereir...	JAB-0882	Carlos Perei...
6	R\$947,11	Sim	Sim	Mariana AL...	JBO-3918	Fernanda S...
7	R\$1334,80	Não	Sim	Rafael Rodr...	XVB-2761	Carlos Perei...
8	R\$1980,00	Não	Não	Beatriz Sou...	OLU-1382	Lucas Andr...
9	R\$801,48	Sim	Não	Carlos Men...	RRG-5146	Carlos Perei...
10	R\$1796,24	Sim	Não	Fernanda L...	TIT-8063	Carlos Perei...

**Figura 8:** Tela inicial - Seção de serviços

## 4.2 Telas de Cadastro e Criação



**Cadastrar Cliente**

Dados do Cliente

CPF

. . -

Nome

E-Mail

Telefone

( ) 9 -

Voltar Cadastrar

**Figura 9:** Tela de Cadastro - Clientes

Cadastro de funcion rio

Dados do Funcion rio

CPF

Nome

E-Mail

Telefone

( ) 9 -

Sal rio (opcional)

Senha

Confirmar senha

Voltar

Cadastrar

Figura 10: Tela de Cadastro - Funcion rios

Gerenciar Clientes

Dados do Cliente

CPF

Pesquisar

ID

Nome

D bito

E-Mail

Total pago

Telefone

( ) 9 -

Ve culos

Excluir

Editar

OK

Cancelar

Dados do Ve culo

ID

Modelo

Tipo

Cor

Ano

Placa

Pe as com problema

Tipo	Marca	Qtd.
------	-------	------

Excluir

Adicionar

Editar

OK

ID	CPF	Nome	E-Mail	Telefone	Debito	Total Pago
1	123.456.789-01	Jo�o Silva	joao@email.com	(11) 98765-4321	R\$772,89	R\$2059,89
2	987.654.321-02	Maria Oliveira	maria@email.com	(21) 99876-5432	R\$573,40	R\$3841,72
3	111.222.333-03	Pedro Santos	pedro@email.com	(31) 91234-5678	R\$589,03	R\$855,99
4	444.555.666-04	Ana Costa	ana@email.com	(41) 97654-3210	R\$405,33	R\$1613,30
5	777.888.999-05	Lucas Pereira	lucas@email.com	(51) 98765-1234	R\$798,72	R\$2446,26

Voltar

Figura 11: Tela de Cadastro - Ve culos

Criar Servi o

Dados do Servi o

CPF do Cliente:

Nome:

CPF do Funcion rio:

Nome:

Ve culos:

Cancelar

Criar Servi o

Figura 12: Tela de Cria  o de Servi os

4.3 Telas de Gerenciamento

Gerenciar Clientes

Dados do Cliente

CPF

Pesquisar

Nome

E-Mail

Telefone

( ) 9 -

ID

D bito

Total pago

Ve culos

Dados do Ve culo

ID

Modelo

Tipo

Cor

Ano

Placa

Pe as com problema

Tipo

Marca

Qtd.

Excluir

Editar

OK

Cancelar

Excluir

Adicionar

Editar

OK

ID	CPF	Nome	E-Mail	Telefone	Debito	Total Pago
1	123.456.789-01	Jo�o Silva	joao@email.com	(11) 98765-4321	R\$908,05	R\$4537,19
2	987.654.321-02	Maria Oliveira	maria@email.com	(21) 99876-5432	R\$963,94	R\$2709,95
3	111.222.333-03	Pedro Santos	pedro@email.com	(31) 91234-5678	R\$515,56	R\$1143,73
4	444.555.666-04	Ana Costa	ana@email.com	(41) 97654-3210	R\$556,29	R\$3846,21
5	777.888.999-05	Lucas Pereira	lucas@email.com	(51) 98765-1234	R\$868,71	R\$4775,84

Voltar

Figura 13: Tela de ger ncia de clientes

Gerenciar Funcion rios

Dados do Funcion rio

ID

CPF

Pesquisar

Nome

E-Mail

Telefone

Sal rio

Administrador

Excluir

Editar

Confirmar

Cancelar

ID	Valor	Consertado	Pago	Cliente	Ve�culo
----	-------	------------	------	---------	---------

Voltar

Figura 14: Tela de ger ncia de funcion rios



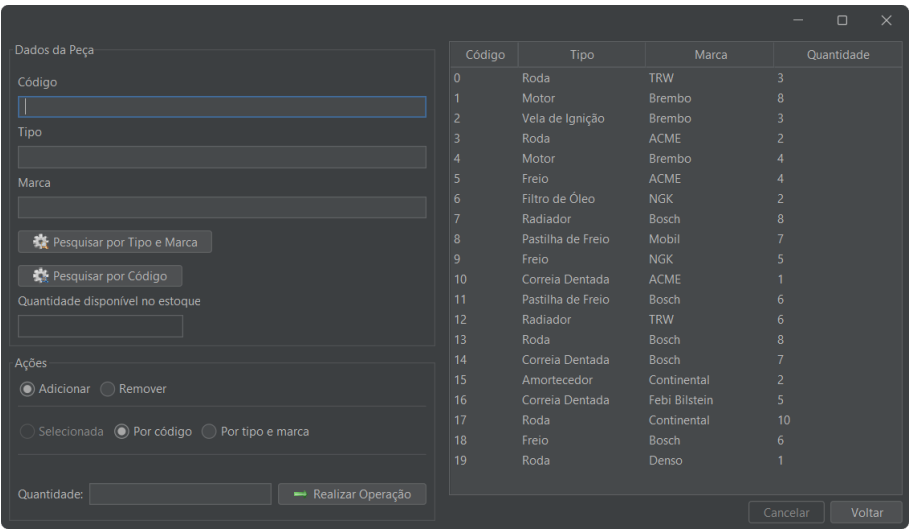


Figura 15: Tela de ger ncia do estoque

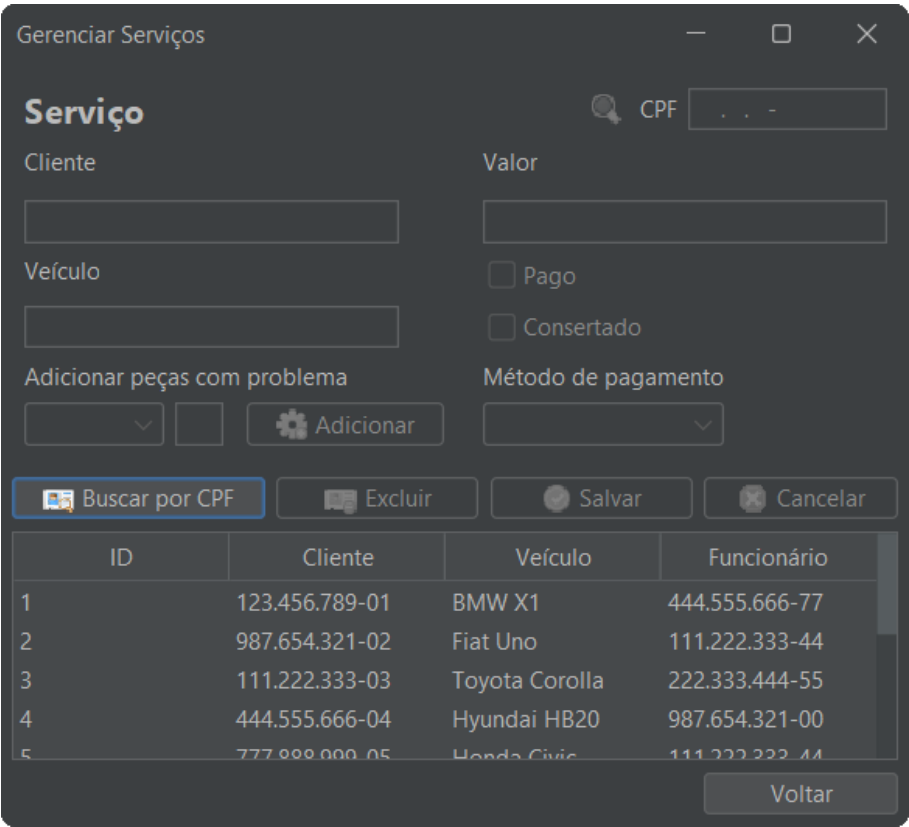


Figura 16: Tela de ger ncia de servi  os