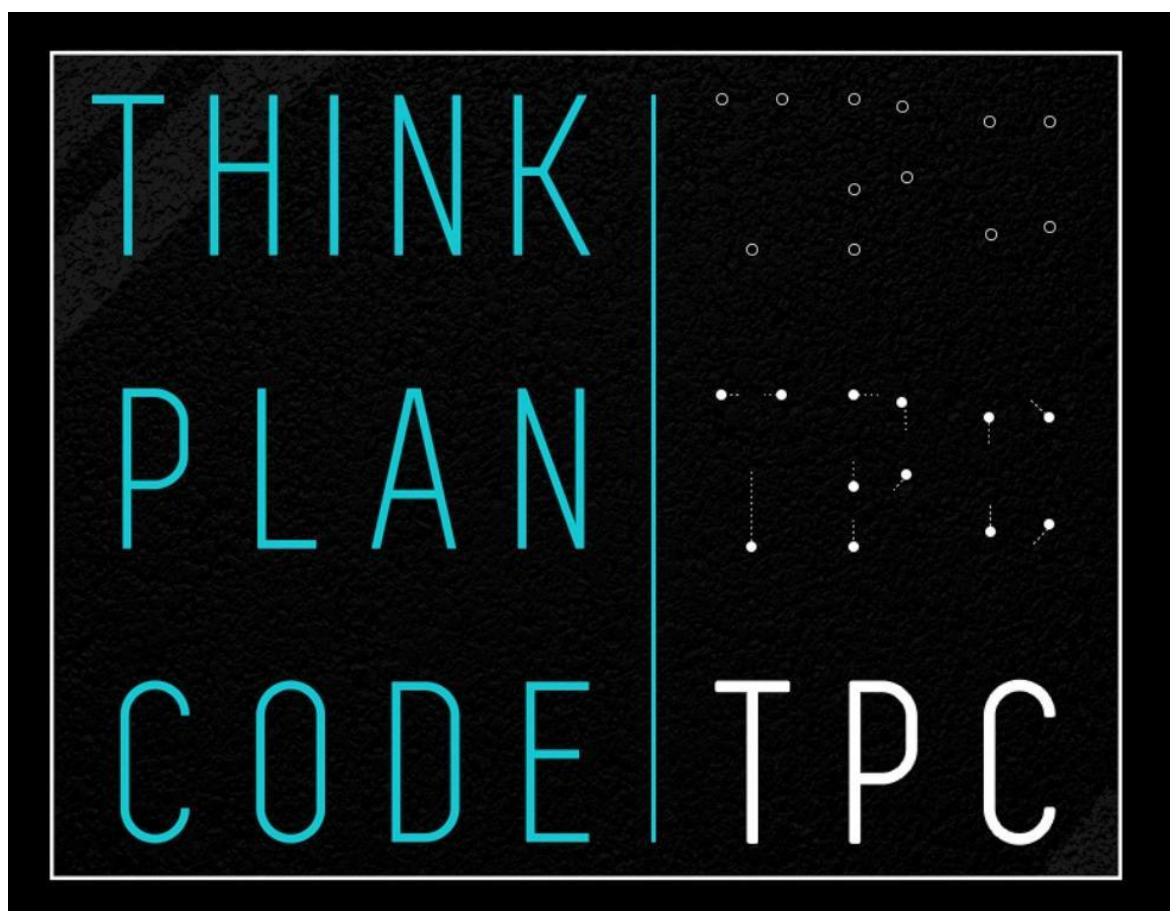


12 DE SETEMBRO DE 2023



DOMAIN DRIVEN DESIGN CHALLENGE – SPRINT 03

THINK, PLAN & CODE

BEATRIZ LUCAS - RM99104 | ENZO FARIAS - RM98792 | EWERTON GONÇALVES - RM98571
| GUILHERME TANTULLI - RM97890 | THIAGO ZUPELLI - RM99085

SUMÁRIO

Domain Driven Design	2
Objetivo e Escopo do projeto	2
Breve descrição das principais funcionalidades	3
Protótipo do Projeto	3
Diagrama Relational (ou modelo Relacional – Banco de dados).....	6
Diagrama de classes	7
Procedimentos para rodar a aplicação	9

Domain Driven Design

Objetivo e Escopo do projeto

O cenário atual em que se encontra a Porto Seguro revela uma dificuldade relevante na coordenação de atendimento a clientes com veículos pesados. A lacuna reside principalmente na alocação ineficaz de prestadores de serviço destinados a tais veículos, particularmente no modal de guinchos pesados. Isso acarreta não apenas em atrasos significativos que causam frustração aos clientes, mas também em retrabalho, que aumenta a sobrecarga sobre os prestadores de serviços e reduz a produtividade geral.

Visando resolver esta questão crucial, estamos propondo um projeto que visa desenvolver um sistema on-demand, com o principal objetivo de permitir que os clientes solicitem serviços e, mais importante, localizem rapidamente um prestador de serviço disponível. Este sistema será essencialmente fundamentado na taxa tarifária (apólice) paga pelo cliente, oferecendo um mecanismo eficiente para localizar o prestador mais próximo que esteja disponível dentro do intervalo de valores vinculado ao cliente.

Ao agilizar a solicitação e a alocação de serviços, prevemos que este projeto permitirá à Porto Seguro aumentar significativamente a qualidade do atendimento ao cliente. Isso será alcançado, principalmente, reduzindo o tempo de espera para o cliente, resultando no aumento na satisfação geral. Este efeito positivo estender-se-á também aos prestadores de serviço, proporcionando-lhes uma melhor organização de suas agendas. Isso, em última análise, diminuirá o tempo ocioso, permitindo-lhes atender mais clientes e, assim, aumentar a eficiência operacional. Com esse aprimoramento, a Porto Seguro poderá fortalecer sua posição no mercado de seguros e manter seu compromisso de fornecer serviços de alta qualidade a seus clientes.

Breve descrição das principais funcionalidades

O aplicativo Porto Seguro HitchLift tem como ambição facilitar e melhorar os chamados de guinchos para veículos pesados, assim como melhorar a experiência do usuário com o serviço da empresa. Assim, o sistema disponibiliza opções para escolha do serviço necessitado, futuramente disponibilizando não somente a localização em que o cliente se encontra, como a localização em tempo real do prestador de serviços que irá atendê-lo.

Além disso, as informações dos veículos registradas e vinculadas àquele cliente estarão disponíveis dentro do perfil do mesmo, de modo à fácil consulta e acesso por meio do banco de dados. O cliente também poderá acessar serviços terceirizados que façam parceria com a empresa Porto Seguro, de modo a ter acesso a recomendações de borracharia, guinchos, etc.

Protótipo do Projeto





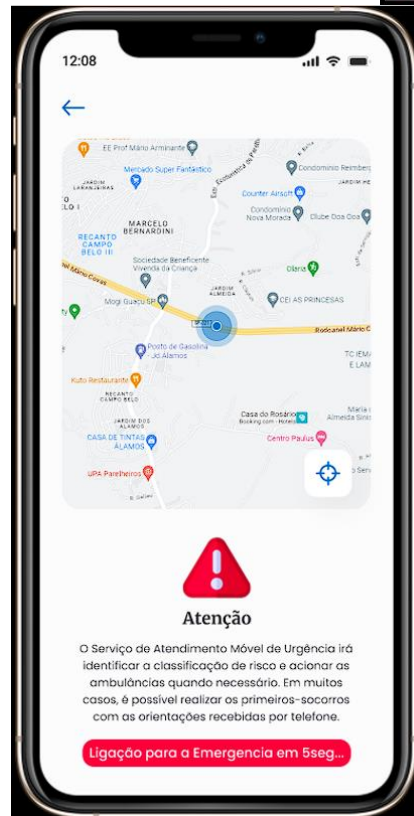
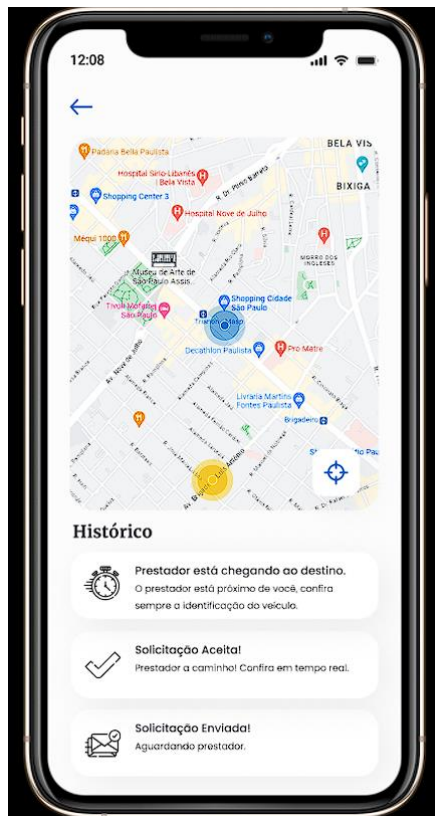


Diagrama Relational (ou Modelo Relacional - Banco de Dados)

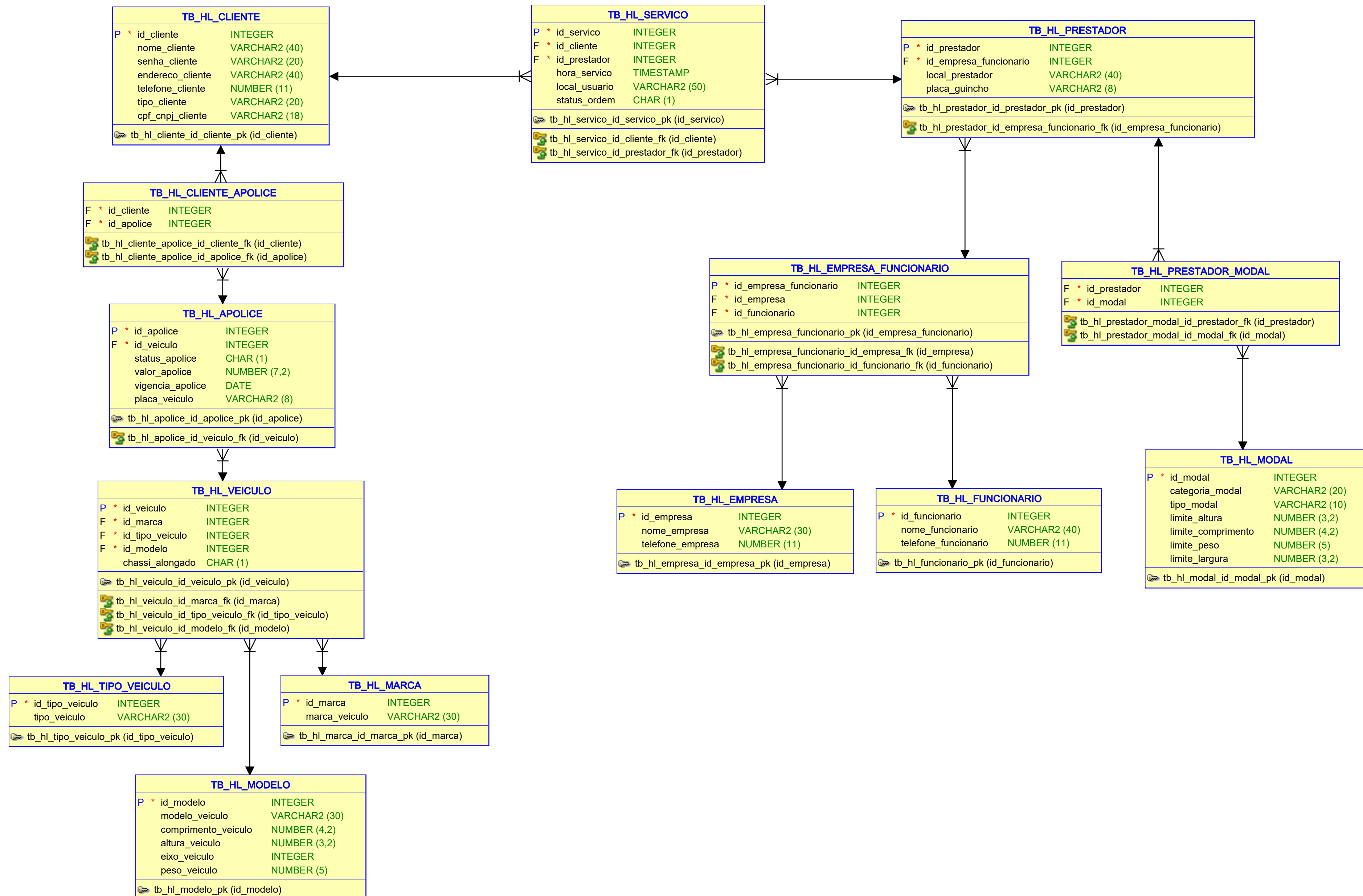
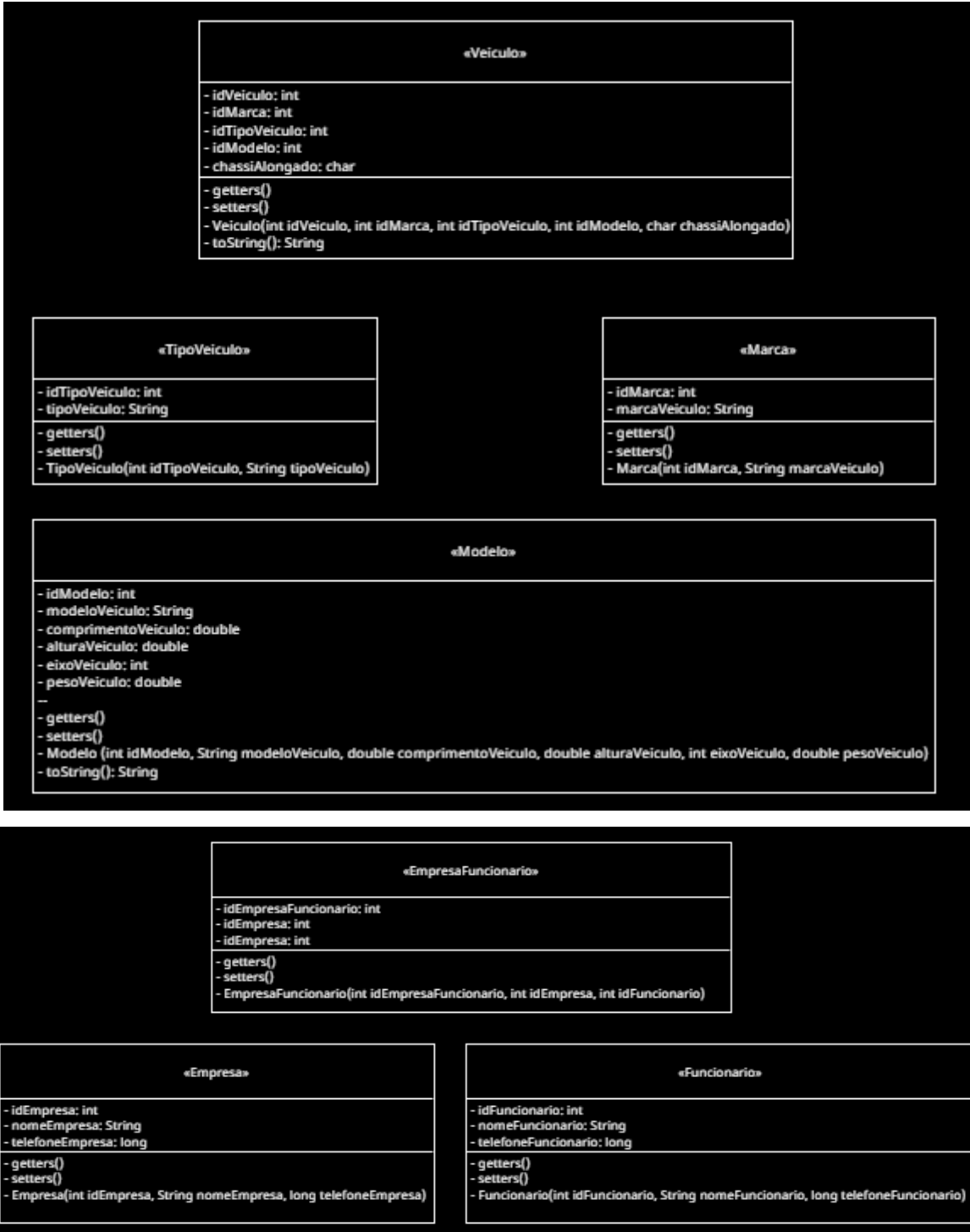


Diagrama de classes



«Prestador»
- idPrestador: int - idEmpresaFuncionario: int - localPrestador: String - placaGuincho: String
- getters() - setters() - Prestador(int idPrestador, int idEmpresaFuncionario, String localPrestador, String placaGuincho) - toString(): String

«PrestadorModal»
- idPrestador: int - idModal: int
- getters() - setters() - PrestadorModal(int idPrestador, int idModal)

«Modal»
- idModal: int - categoriaModal: String - tipoModal: String - limiteAltura: double - limiteComprimento: double - limitePeso: double - limiteLargura: double - -- - getters() - setters() - Modal (int idModal, String tipoModal, String categoriaModal, double limiteAltura, double limiteComprimento, double limitePeso, double limiteLargura)

«Cliente»
- idCliente: int - nomeCliente: String - senhaCliente: String - enderecoCliente: String - telefoneCliente: long - tipoCliente: String - cpfCnpjCliente: String
- getters() - setters() - Cliente(int idCliente, String nomeCliente, String senhaCliente, String enderecoCliente, long telefoneCliente, String tipoCliente, String cpfCnpjCliente) - toString(): String

«ClienteApolice»
- idCliente: int - idApolice: int
- getters() - setters() - ClienteApolice(int idCliente, int idApolice)

«Apolice»
- idApolice: int - idVeiculo: int - statusApolice: String - valorApolice: double - vigenciaApolice: date - placaVeiculo: String
- getters() - setters() - Veiculo(int idApolice, int idVeiculo, String statusApolice, double valorApolice, date vigenciaApolice, String placaVeiculo) - toString(): String

«Utils»
- encontrarVeiculo(List<Veiculo>, int idVeiculo): Veiculo - calcularValorTotalApolice(List<Apolice>): valorTotal - criarArrayList(): ArrayList<T>

Procedimentos para rodar a aplicação

Nossa aplicação funciona de maneira simples, sendo iniciada com a abertura da conexão com o banco de dados, previamente criado e configurado. Caso não haja essa conexão, não é possível executar o programa. Após a configuração inicial ter sido feita, nosso programa iniciará com a criação de listas para armazenar todos os tipos de dados, facilitando sua consulta no futuro. Essa criação é feita por meio de uma função genérica que irá retornar a lista específica, chamada “criarArrayList()”. Após instanciar os objetos por meio dos construtores parametrizados, é possível adicioná-los às listas, por meio do comando “.add()”. Assim, esses dados serão salvos e passíveis de consulta futuramente.

Após instanciar os objetos, através dos métodos da classe Gerenciador, que tem como função conectar, inserir e incluir dados diretamente com nosso banco de dados, é possível fazer essa transmissão de dados, utilizando os métodos específicos para inserção de dados em sua tabela específica, ou utilizando o método excluir, que é genérico, para apagar informações na tabela específica com seu id correspondente. É importante ressaltar que, devido à ligação de chaves no banco de dados (Primary Key e Foreign Key), o método excluir deve ser chamado em uma ordem que respeite a exclusão das chaves e seus relacionamentos, do contrário, haverá erro e não será aplicado corretamente.

Também é possível obter as principais informações por meio do método sobrescrito “toString()”, que trará os dados armazenados no objeto criado das classes como Cliente, Apólice e Veículo. Além disso, é possível, também, fazer a consulta de um veículo, analisando a lista criada para armazenar estes, assim como consultar o valor total de apólices, também armazenadas em sua lista.