**UNIFECAF**

CENTRO UNIVERSITÁRIO FACULDADE DE CIÊNCIAS APLICADAS

**Hadassa Ester de Souza Bezerra**

RA: 32455

**SISTEMA DE GESTÃO DE ESTOQUE DE VEÍCULOS**

Relatório Teórico

Trabalho apresentado como requisito parcial para avaliação da disciplina Object Oriented Programming, ministrada pelo professor Vitor Jesus.

2025

# 1. INTRODUÇÃO

O presente trabalho apresenta o desenvolvimento de um Sistema de Gestão de Estoque de Veículos, projetado para concessionárias e revendas automotivas. O sistema foi desenvolvido utilizando os princípios da Programação Orientada a Objetos (POO) e tecnologias modernas como Java 17, Spring Boot e PostgreSQL.

A motivação para este projeto surge da necessidade real do mercado automotivo em gerenciar de forma eficiente um grande volume de veículos, com diferentes marcas, modelos, preços e características. Um sistema bem estruturado pode otimizar processos, reduzir erros e melhorar a experiência tanto dos vendedores quanto dos clientes.

Este relatório documenta todo o processo de desenvolvimento, desde o levantamento de requisitos até a implementação final, demonstrando a aplicação prática dos conceitos de POO aprendidos na disciplina Object Oriented Programming.

# 2. LEVANTAMENTO DE REQUISITOS

Para compreender as necessidades do sistema, foram elaboradas 12 perguntas fundamentais que guiaram o desenvolvimento:

## 2.1 Qual é o principal objetivo do sistema?

O objetivo principal é gerenciar o estoque de veículos de forma eficiente, permitindo o cadastro, consulta, atualização e remoção de veículos, bem como o controle de marcas e modelos. O sistema deve facilitar a busca de veículos por diferentes critérios e manter o controle do status de cada veículo (disponível, vendido, reservado ou em manutenção).

## 2.2 Quem são os usuários do sistema?

Os principais usuários são vendedores, gerentes de estoque e administradores de concessionárias. Vendedores precisam consultar rapidamente informações sobre veículos disponíveis para atender clientes. Gerentes necessitam de visão geral do estoque, preços e status dos veículos. Administradores realizam cadastros, atualizações e manutenção dos dados do sistema.

## 2.3 Quais são as principais entidades do sistema?

O sistema possui três entidades principais: Marca (representando fabricantes como Toyota, Volkswagen), Modelo (representando modelos específicos como Corolla, Gol) e Veículo (representando cada unidade física no estoque com suas características específicas como cor, ano, preço e quilometragem).

## 2.4 Quais informações devem ser armazenadas sobre cada veículo?

Cada veículo deve ter: identificador único, modelo associado, ano de fabricação, cor, preço, quilometragem e status. O status indica se o veículo está disponível para venda, já foi vendido, está reservado ou em manutenção. Essas informações são essenciais para tomada de decisões e atendimento ao cliente.

## 2.5 Como os veículos se relacionam com marcas e modelos?

O sistema utiliza relacionamentos hierárquicos: uma Marca pode ter vários Modelos (Toyota tem Corolla, Hilux, etc.), e um Modelo pode ter vários Veículos (podem existir múltiplos Corollas com diferentes cores e anos). Essa estrutura permite organização lógica e facilita buscas específicas.

## 2.6 Quais operações básicas o sistema deve suportar?

O sistema deve implementar operações CRUD completas (Create, Read, Update, Delete) para todas as entidades. Adicionalmente, deve permitir buscas avançadas com múltiplos filtros, como buscar veículos de uma marca específica, dentro de uma faixa de preço, com determinado status, e combinações desses critérios.

## 2.7 Quais validações são necessárias?

O sistema deve validar: campos obrigatórios não podem ser vazios, anos devem estar entre 1900 e 2100, preços devem ser positivos, quilometragem não pode ser negativa, nomes de marcas devem ser únicos, e relacionamentos devem ser válidos (não é possível criar um veículo sem um modelo válido associado).

## 2.8 Como o sistema deve tratar erros?

O sistema implementa tratamento global de exceções, retornando mensagens claras e códigos HTTP apropriados. Erros de validação retornam 400 (Bad Request) com detalhes específicos, recursos não encontrados retornam 404 (Not Found), e operações bem-sucedidas retornam códigos 200 (OK) ou 201 (Created).

## 2.9 Qual arquitetura será utilizada?

O sistema utiliza arquitetura em camadas: Controller (recebe requisições HTTP e retorna respostas), Service (contém lógica de negócio e validações), Repository (acessa o banco de dados) e Model (representa as entidades). Essa separação promove manutenibilidade, testabilidade e facilita futuras expansões.

## 2.10 Por que usar PostgreSQL?

PostgreSQL foi escolhido por ser um banco de dados robusto, open-source, com excelente suporte a relacionamentos complexos, transações ACID, e recursos avançados como constraints e triggers. É amplamente utilizado em produção e oferece ótimo desempenho para aplicações empresariais.

## 2.11 Como o sistema será testado?

O sistema pode ser testado através de ferramentas como Postman, Insomnia ou cURL. Todos os endpoints REST estão documentados com exemplos de requisições e respostas esperadas. São fornecidos dados de exemplo para popular o banco e realizar testes completos de todas as funcionalidades.

## 2.12 Quais são os diferenciais do sistema?

Os principais diferenciais são: filtros combinados avançados que permitem buscas complexas, estrutura escalável que facilita adição de novas funcionalidades, código limpo seguindo boas práticas da indústria, documentação completa e detalhada, e validações rigorosas que garantem integridade dos dados.

# 3. DESCRIÇÃO DAS ENTIDADES

## 3.1 Marca

**Atributos:**

* id: Identificador único da marca (Long)
* nome: Nome da marca, deve ser único (String)
* descricao: Descrição opcional da marca (String)
* modelos: Lista de modelos associados a esta marca (List<Modelo>)

A entidade Marca representa fabricantes de veículos como Toyota, Volkswagen, Ford. Ela mantém a lista de todos os modelos produzidos por aquele fabricante. A validação garante que não existam duas marcas com o mesmo nome no sistema.

## 3.2 Modelo

**Atributos:**

* id: Identificador único do modelo (Long)
* nome: Nome do modelo (String)
* descricao: Descrição opcional do modelo (String)
* marca: Marca à qual este modelo pertence (Marca)
* veiculos: Lista de veículos deste modelo (List<Veiculo>)

A entidade Modelo representa modelos específicos como Corolla, Gol, Civic. Cada modelo está obrigatoriamente associado a uma marca. Um modelo pode ter diversos veículos cadastrados (diferentes anos, cores e configurações do mesmo modelo).

## 3.3 Veículo

**Atributos:**

* id: Identificador único do veículo (Long)
* modelo: Modelo do veículo (Modelo)
* anoFabricacao: Ano de fabricação, entre 1900 e 2100 (Integer)
* cor: Cor do veículo (String)
* preco: Preço do veículo, deve ser positivo (BigDecimal)
* quilometragem: Quilometragem rodada, não pode ser negativa (Integer)
* status: Status do veículo: DISPONIVEL, VENDIDO, RESERVADO ou EM\_MANUTENCAO (Enum)

A entidade Veículo representa cada unidade física no estoque. Contém todas as informações específicas necessárias para venda: características físicas (cor, ano, quilometragem), informações comerciais (preço, status) e relacionamento com o modelo. O status permite controlar o ciclo de vida do veículo no estoque.

# 4. CONCEITOS DE POO APLICADOS

## 4.1 Encapsulamento

O encapsulamento foi aplicado em todas as entidades através de atributos privados acessados via getters e setters. Isso protege os dados de acesso direto e permite validações nos métodos setters. Por exemplo, ao definir o preço de um veículo, o setter pode validar se o valor é positivo antes de atribuir. O uso da anotação @Data do Lombok automatiza a criação desses métodos, mantendo o código limpo.

## 4.2 Abstração

A abstração está presente na separação de responsabilidades em camadas. As interfaces Repository abstraem a lógica de persistência, permitindo que o Service trabalhe com conceitos de alto nível sem se preocupar com detalhes de SQL. Da mesma forma, os DTOs (Data Transfer Objects) abstraem a estrutura interna das entidades, expondo apenas os dados necessários para a API.

## 4.3 Herança

A herança é utilizada através das interfaces JpaRepository que são estendidas pelos repositories personalizados. MarcaRepository, ModeloRepository e VeiculoRepository herdam métodos prontos como save(), findAll(), findById() do JpaRepository, podendo adicionar métodos customizados específicos de cada entidade. Isso promove reuso de código e padronização.

## 4.4 Polimorfismo

O polimorfismo aparece na implementação de métodos com mesmo nome mas comportamentos diferentes. Os métodos de busca (findByMarca, findByModelo, findByStatus) têm o mesmo propósito geral (buscar veículos) mas cada um filtra de forma diferente. O tratamento global de exceções também demonstra polimorfismo, tratando diferentes tipos de exceções de formas apropriadas.

## 4.5 Classes e Objetos

O sistema é estruturado em classes bem definidas: Marca, Modelo e Veículo são classes de modelo anotadas com @Entity do JPA. Controllers são classes que recebem requisições HTTP. Services são classes que contêm lógica de negócio. Cada requisição cria objetos dessas classes, manipula seus dados e persiste no banco através do Hibernate.

## 4.6 Composição

A composição está presente nos relacionamentos entre entidades. Um Veículo é composto por um Modelo (relação @ManyToOne), e um Modelo pertence a uma Marca. Essa composição reflete a estrutura real: um veículo não existe sem um modelo, e um modelo não existe sem uma marca. A anotação cascade garante que operações em cascata mantenham a integridade dessas composições.

# 5. REQUISITOS FUNCIONAIS

Os requisitos funcionais descrevem o que o sistema deve fazer:

* RF01: O sistema deve permitir cadastro completo de marcas com nome único e descrição opcional
* RF02: O sistema deve permitir cadastro de modelos associados a marcas existentes
* RF03: O sistema deve permitir cadastro de veículos com todas as informações obrigatórias (modelo, ano, cor, preço, quilometragem, status)
* RF04: O sistema deve permitir busca de veículos por status (disponível, vendido, reservado, em manutenção)
* RF05: O sistema deve permitir busca de veículos por marca ou modelo
* RF06: O sistema deve permitir filtros combinados (marca + faixa de ano + faixa de preço + status)
* RF07: O sistema deve permitir atualização e remoção de marcas, modelos e veículos respeitando integridade referencial

# 6. REQUISITOS NÃO FUNCIONAIS

Os requisitos não funcionais descrevem qualidades e restrições do sistema:

* RNF01: Desempenho - O sistema deve responder requisições em menos de 2 segundos para operações simples
* RNF02: Segurança - O sistema deve validar todos os dados de entrada para prevenir injeção de SQL e ataques
* RNF03: Escalabilidade - A arquitetura em camadas permite fácil expansão de funcionalidades
* RNF04: Manutenibilidade - O código deve seguir padrões de projeto e boas práticas para facilitar manutenção
* RNF05: Portabilidade - O sistema deve funcionar em Windows, Linux e macOS
* RNF06: Usabilidade - A API REST deve fornecer mensagens de erro claras e códigos HTTP apropriados

# 7. JUSTIFICATIVA DAS TECNOLOGIAS

## 7.1 Java 17

Java 17 é uma versão LTS (Long Term Support) com suporte estendido até 2029. Oferece recursos modernos como pattern matching, records, e melhorias de performance. É amplamente utilizado em ambiente corporativo e possui vasta documentação e comunidade ativa. A orientação a objetos nativa do Java facilitou a implementação dos conceitos de POO.

## 7.2 Spring Boot

Spring Boot simplifica o desenvolvimento de aplicações Java através de configuração automática e convenção sobre configuração. Reduz drasticamente código boilerplate, possui injeção de dependências nativa, e oferece integração perfeita com JPA/Hibernate. É o framework mais utilizado para desenvolvimento backend em Java, garantindo que o projeto segue padrões da indústria.

## 7.3 PostgreSQL

PostgreSQL é um banco de dados relacional robusto, open-source e gratuito. Oferece conformidade total com ACID (Atomicidade, Consistência, Isolamento, Durabilidade), suporte avançado a constraints e triggers, excelente performance para queries complexas, e é amplamente utilizado em produção por empresas de todos os tamanhos. Sua maturidade e confiabilidade foram decisivos na escolha.

## 7.4 Maven

Maven é a ferramenta padrão para gerenciamento de dependências e build de projetos Java. Através do arquivo pom.xml, centraliza todas as dependências do projeto, facilita download automático de bibliotecas, padroniza estrutura de diretórios, e automatiza processo de compilação e empacotamento. Isso garante que o projeto pode ser facilmente replicado em diferentes ambientes.

## 7.5 Hibernate e JPA

Hibernate é o ORM (Object-Relational Mapping) mais popular para Java, implementando a especificação JPA. Permite trabalhar com objetos Java ao invés de SQL direto, traduzindo automaticamente operações em objetos para comandos SQL. Isso simplifica desenvolvimento, reduz erros, e torna o código independente do banco de dados específico utilizado.

# 8. CONCLUSÃO

O desenvolvimento do Sistema de Gestão de Estoque de Veículos proporcionou aplicação prática e aprofundada dos conceitos de Programação Orientada a Objetos. O projeto demonstrou como princípios fundamentais de POO - encapsulamento, abstração, herança e polimorfismo - são essenciais para criar sistemas robustos, manuteníveis e escaláveis.

A arquitetura em camadas implementada (Controller, Service, Repository, Model) evidenciou a importância da separação de responsabilidades e como isso facilita testes, manutenção e evolução do código. O uso de tecnologias modernas e amplamente adotadas pelo mercado (Java 17, Spring Boot, PostgreSQL) garantiu que o projeto não apenas atende aos requisitos acadêmicos, mas também reflete práticas reais da indústria de software.

Os desafios enfrentados durante o desenvolvimento - desde o levantamento de requisitos até a implementação de filtros complexos - contribuíram significativamente para o aprendizado. A necessidade de validações rigorosas, tratamento adequado de exceções e manutenção da integridade referencial demonstrou que desenvolvimento de software vai além de apenas escrever código que funciona: é necessário pensar em segurança, performance e experiência do usuário.

Como trabalhos futuros, o sistema pode ser expandido com autenticação e autorização de usuários, implementação de interface gráfica (frontend), geração de relatórios em PDF, integração com sistemas de pagamento, e recursos de business intelligence para análise de vendas. A base sólida construída facilita essas expansões futuras.

Este projeto consolidou o conhecimento adquirido na disciplina Object Oriented Programming e demonstrou preparação para desenvolver sistemas profissionais seguindo as melhores práticas da engenharia de software.

# REFERÊNCIAS

BLOCH, Joshua. Effective Java. 3. ed. Boston: Addison-Wesley, 2018.

FOWLER, Martin. Patterns of Enterprise Application Architecture. Boston: Addison-Wesley, 2002.

GAMMA, Erich et al. Design Patterns: Elements of Reusable Object-Oriented Software. Boston: Addison-Wesley, 1994.

ORACLE. Java Platform, Standard Edition Documentation. Disponível em: https://docs.oracle.com/en/java/. Acesso em: 30 out. 2025.

POSTGRESQL GLOBAL DEVELOPMENT GROUP. PostgreSQL Documentation. Disponível em: https://www.postgresql.org/docs/. Acesso em: 30 out. 2025.

PIVOTAL SOFTWARE. Spring Boot Reference Documentation. Disponível em: https://spring.io/projects/spring-boot. Acesso em: 30 out. 2025.

RED HAT. Hibernate ORM Documentation. Disponível em: https://hibernate.org/orm/documentation/. Acesso em: 30 out. 2025.

SOMMERVILLE, Ian. Software Engineering. 10. ed. Boston: Pearson, 2015.