


- 1 ☐ Vehicle Resale API
  - 1.1 Clean Architecture & Kubernetes Implementation
  - 1.2 ☐ Links Principais
    - 1.2.1 ☐ Repositórios GitHub
    - 1.2.2 ☐ Vídeo Demonstrativo
  - 1.3 ☐ Arquitetura do Sistema
    - 1.3.1 ☐ Microservices
    - 1.3.2 ☐ Clean Architecture (Ambos os Serviços)
    - 1.3.3 ☐ Princípios SOLID
  - 1.4 ☐ Stack Tecnológico
    - 1.4.1 ☐ Backend
    - 1.4.2 ☐ Bancos de Dados
    - 1.4.3 ☐ Infraestrutura
  - 1.5 ☐ Execução Local
    - 1.5.1 ☐ Docker Compose (Recomendado)
    - 1.5.2  ☐ Kubernetes com Minikube
    - 1.5.3 ☐ Desenvolvimento Local
  - 1.6 ☐ Funcionalidades
    - 1.6.1 ☐ Vehicle Catalog API
    - 1.6.2 ☐ Vehicle Sales API
  - 1.7 ☐ Fluxo de Integração
  - 1.8 ☐ Endpoints Principais
    - 1.8.1 Vehicle Catalog API
    - 1.8.2 Vehicle Sales API
  - 1.9 ☐ Segurança & Qualidade
    - 1.9.1 ☐ Segurança
    - 1.9.2 ☐ Monitoramento
  - 1.10 ☐ Performance
    - 1.10.1 ☐ Benchmarks
    - 1.10.2 ☐ Otimizações
  - 1.11 ☐ Testes
    - 1.11.1 ☐ Cobertura
    - 1.11.2 ☐ CI/CD
  - 1.12 ☐ Autor
  - 1.13 ☐ Suporte Técnico

## 1 ☐ Vehicle Resale API

### 1.1 Clean Architecture & Kubernetes Implementation

---

Sistema de microservices em .NET 8 implementando Clean Architecture e princípios SOLID para gerenciamento de catálogo de veículos e processamento de vendas, com infraestrutura completa utilizando bancos SQL (PostgreSQL) e NoSQL (MongoDB Atlas), Docker e Kubernetes.

---

### 1.2 ☐ Links Principais

#### 1.2.1 ☐ Repositórios GitHub

**Vehicle Sales API:** <https://github.com/ohntrebor/vehicle-sales>

**Vehicle Catalog API:** <https://github.com/ohntrebor/vehicle-catalog>

#### 1.2.2 ☐ Vídeo Demonstrativo

[https://www.youtube.com/watch?v=LKEupUM92\\_Q](https://www.youtube.com/watch?v=LKEupUM92_Q) (16 min)

---

## 1.3 Arquitetura do Sistema

### 1.3.1 Microservices

- **Vehicle Catalog API:** Gerenciamento de catálogo de veículos (PostgreSQL)
- **Vehicle Sales API:** Processamento de vendas e pagamentos (MongoDB Atlas)
- **Integração:** Comunicação via HTTP entre serviços

### 1.3.2 Clean Architecture (Ambos os Serviços)

- **Domain:** Entidades, Value Objects, Interfaces
- **Application:** Use Cases, DTOs, Validações
- **Infrastructure:** Persistência, Repositórios, Serviços Externos
- **Presentation:** Controllers, API, Middlewares

### 1.3.3 Princípios SOLID

- **S:** Single Responsibility Principle
  - **O:** Open/Closed Principle
  - **L:** Liskov Substitution Principle
  - **I:** Interface Segregation Principle
  - **D:** Dependency Inversion Principle
- 

## 1.4 Stack Tecnológico

### 1.4.1 Backend

- **.NET 8** - Framework principal
- **Entity Framework Core** - ORM para PostgreSQL
- **MongoDB Driver** - Acesso ao MongoDB Atlas
- **MediatR** - CQRS pattern
- **AutoMapper** - Mapeamento de objetos
- **FluentValidation** - Validação de dados

### 1.4.2 Bancos de Dados

- **PostgreSQL** - Catálogo de veículos (relacional)
- **MongoDB Atlas** - Vendas e transações (documento)

### 1.4.3 Infraestrutura

- **Docker & Docker Compose** - Containerização
  - **Kubernetes** - Orquestração de containers
  - **Minikube** - Cluster local para desenvolvimento
  - **Health Checks** - Monitoramento de saúde
- 

## 1.5 Execução Local

### 1.5.1 Docker Compose (Recomendado)

```
# Clone os repositórios
git clone https://github.com/ohntrebor/vehicle-sales
git clone https://github.com/ohntrebor/vehicle-catalog
```

```
# Execute o sistema completo
cd vehicle-sales
docker compose up -d --build
```

**Acessos:** - **Vehicle Catalog API:** <http://localhost:5000/swagger> - **Vehicle Sales API:** <http://localhost:5001/swagger> - **MongoDB Express:** <http://localhost:8081>

### 1.5.2 Kubernetes com Minikube

```
# Setup automático completo
cd vehicle-sales
make k8s-full-deploy
```

**Acessos:** - **Vehicle Catalog API:** <http://localhost:5000> - **Vehicle Sales API:** <http://localhost:9000/swagger>

### 1.5.3 Desenvolvimento Local





```
# Vehicle Catalog API
cd vehicle-catalog
dotnet restore
dotnet run --project VehicleCatalog.API

# Vehicle Sales API
cd vehicle-sales
dotnet restore
dotnet run --project VehicleSales.API
```





---

## 1.6 Funcionalidades

### 1.6.1 Vehicle Catalog API

-  **CRUD de Veículos** - Cadastro, edição, consulta, exclusão
-  **Busca Avançada** - Filtros por marca, modelo, preço, ano
-  **Gestão de Status** - Disponível, vendido, reservado
-  **Notificações** - Recebe webhooks de vendas

### 1.6.2 Vehicle Sales API

-  **Consulta de Catálogo** - Proxy para Vehicle Catalog API
  -  **Registro de Vendas** - Processamento de transações
  -  **Webhook de Pagamento** - Integração com gateway
  -  **Histórico de Vendas** - Auditoria completa
- 

## 1.7 Fluxo de Integração

```
graph TD
    A[Cliente] --> B[Vehicle Sales API]
    B --> C[Vehicle Catalog API]
    C --> D[PostgreSQL]
    B --> E[MongoDB Atlas]
    B --> F[Payment Gateway]
    F --> B
    B --> C
```

1. **Consulta de Veículos** - Sales API → Catalog API

- 2. **Registro de Venda** - Dados salvos no MongoDB Atlas
- 3. **Webhook de Pagamento** - Gateway → Sales API
- 4. **Notificação de Venda** - Sales API → Catalog API
- 5. **Atualização de Status** - Catalog API → PostgreSQL

## 1.8 Endpoints Principais

### 1.8.1 Vehicle Catalog API





Método	Endpoint	Descrição
GET	/api/vehicles	Listar veículos
POST	/api/vehicles	Cadastrar veículo
PUT	/api/vehicles/{id}	Atualizar veículo
DELETE	/api/vehicles/{id}	Remover veículo
GET	/api/vehicles/search	Busca com filtros

### 1.8.2 Vehicle Sales API





Método	Endpoint	Descrição
GET	/api/catalog/vehicles	Consultar catálogo
POST	/api/sales	Registrar venda
GET	/api/sales	Listar vendas
POST	/api/sales/payment-webhook	Webhook pagamento

## 1.9 Segurança & Qualidade

### 1.9.1 Segurança

-  **HTTPS** - Comunicação criptografada
-  **Secrets** - Gerenciamento via Kubernetes
-  **Validação** - FluentValidation em todas as entradas
-  **CORS** - Configurado adequadamente

### 1.9.2 Monitoramento

-  **Health Checks** - /health, /health/live, /health/ready
-  **Logging** - Estruturado com Serilog
-  **Métricas** - Prometheus ready
-  **Observabilidade** - Traces distribuídos

## 1.10 Performance

### 1.10.1 Benchmarks

- **Response Time:** < 200ms (consultas)
- **Throughput:** 100+ req/s por instância
- **Disponibilidade:** 99.9% com múltiplas réplicas
- **Auto-scaling:** Baseado em CPU/Memória

### 1.10.2 Otimizações

- **Connection Pooling** - PostgreSQL e MongoDB
  - **Async/Await** - Programação assíncrona
  - **Caching** - Em memória para consultas frequentes
  - **Lazy Loading** - Entity Framework otimizado
- 

## 1.11 □ Testes

### 1.11.1 □ Cobertura

- **Unit Tests** - Domínio e Application
- **Integration Tests** - Controllers e Repositories
- **Health Check Tests** - Monitoramento
- **Load Tests** - Performance e stress

### 1.11.2 □ CI/CD

- **GitHub Actions** - Build e testes automatizados
  - **Docker Registry** - Imagens versionadas
  - **Kubernetes Deploy** - Rolling updates
  - **Sonarqube** - Análise de código
- 

## 1.12 □ Autor

**Robert A. dos Anjos** - **Email:** robert.ads.anjos@gmail.com - **GitHub:** @ohntrebor -  
**LinkedIn:** [Robert dos Anjos](#)

---

## 1.13 □ Suporte Técnico

Para suporte, dúvidas ou contribuições: - **Email:** robert.ads.anjos@gmail.com - **Issues:**  
GitHub Issues nos repositórios - **Documentation:** README.md em cada repositório

---

*Sistema completo de revenda de veículos desenvolvido com foco em arquitetura limpa, escalabilidade e boas práticas de desenvolvimento.*