DESENVOLVIMENTO WEB I

Car Maintenance API

API REST para Gestão de Manutenção de Automóveis

Grupo 33

Gonçalo Cidras • César Ramos • Jorge Espogeira

Universidade da Maia (UMAIA) • 2024/25

Agenda

1. **Introdução**
   Tema e objetivos do projeto
2. **Modelo de Dados**
   Tabelas e relações da BD
3. **Docker**
   Contentorização da aplicação
4. **Arquitetura**
   Estrutura em 3 camadas
5. **API REST**
   Endpoints e filtros HTTP
6. **Demonstração**
   API em funcionamento

INTRODUÇÃO

O que é a Car Maintenance API?

Sistema de gestão completo para oficinas mecânicas, permitindo registar e consultar toda a informação sobre manutenções de veículos.

PROPRIETÁRIOS

Gestão de clientes e os seus dados de contacto

VEÍCULOS

Registo de veículos por proprietário

SERVIÇOS

Catálogo de tipos de manutenção

MANUTENÇÕES

Histórico completo de serviços

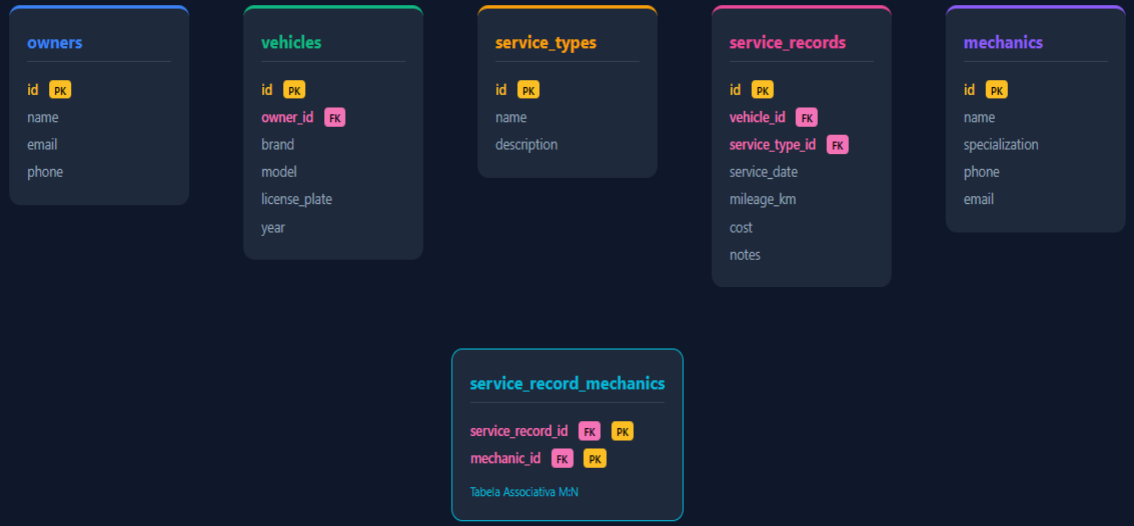
MECÂNICOS

Profissionais e especializações

DESIGN-FIRST

OpenAPI definida antes do código

Diagrama do Modelo de Dados



Relações 1:N (Um para Muitos)

owners → vehicles

vehicles → service_records

service_types → service_records

Um owner pode ter vários vehicles
Um vehicle pode ter vários service_records

Relação M:N (Muitos para Muitos)

service_records ↔ **mechanics**

Um serviço pode ter vários mecânicos
Um mecânico pode trabalhar em vários serviços

Implementado através da tabela associativa

PK = Primary Key (Chave Primária) FK = Foreign Key (Chave Estrangeira) Tabela Associativa (M:N)

INFRAESTRUTURA

Docker - Contentorização

2 CONTAINERS

MySQL 8.0

- Base de dados car_maintenance
- Script init.sql ao iniciar
- Porta: 3307 (dev) / 3308 (prod)

Node.js 20

- API Express.js
- Swagger UI em /docs
- Porta: 8080 (dev) / 8081 (prod)

DOCKERHUB

Imagens publicadas:

inf25dw1g33/sql:V1

inf25dw1g33/app:V1

depends_on

A app só inicia depois do MySQL

Docker: Desenvolvimento vs Produção

DESENVOLVIMENTO

Para programar e testar

`docker-compose.dev.yml`

- **Faz BUILD local**
  Constrói as imagens a partir dos Dockerfiles
- **Precisa do código**
  Necessita de todo o projeto localmente
- **Mais lento**
  1-3 minutos (compila e instala)

Portas

API:	`localhost:8080`
MySQL:	`localhost:3307`

DockerHub

PRODUÇÃO

Para demonstrar e distribuir

`docker-compose.prod.yml`

- **Usa imagens prontas**
  Descarrega do DockerHub (não faz build)
- **Não precisa do código**
  Só precisa do docker-compose.yml
- **Mais rápido**
  10-30 segundos (só descarrega)

Portas

API:	`localhost:8081`
MySQL:	`localhost:3308`

Imagens no DockerHub

`inf25dw1g33/sql:V1` `inf25dw1g33/app:V1`

Qualquer pessoa pode correr a aplicação sem ter o código fonte!

Desenvolvimento (dev) Produção (prod)

APLICAÇÃO

Arquitetura em 3 Camadas

Rotas (api/)

Define os endpoints
GET, POST, PUT, DELETE
Mapeia para controllers

Controllers

Recebe pedidos HTTP
Extrai parâmetros/filtros
Devolve resposta JSON

Services

Lógica de negócio
Constrói queries SQL
Prepared statements

API REST

Recursos e Endpoints

5 RECURSOS

- /api/owners
- /api/vehicles
- /api/service-types
- /api/service-records
- /api/mechanics

4 VERBOS HTTP

CREATE/POST

READ/GET

UPDATE/PUT

DELETE

ENDPOINTS M:N

Mecânicos de um serviço:

`/service-records/{id}/mechanics`

Serviços de um mecânico:

`/mechanics/{id}/service-records`

API REST

Filtros HTTP (Query Parameters)

Pesquisa e filtragem de recursos através de parâmetros na URL usando **LIKE** para pesquisa parcial.

```
// Filtrar owners por nome
GET /api/owners?name=Maria
```

```
// Filtrar por especialização
GET /api/mechanics?specialization=Motor
```

```
// Filtrar veículos por marca
GET /api/vehicles?brand=BMW
```

```
// Serviços de um veículo
GET /api/service-records?vehicle_id=1
```

```
// Múltiplos filtros
GET /api/vehicles?brand=BMW&year=2020
```

```
// Filtrar tipos de serviço
GET /api/service-types?name=óleo
```

🔒 **Segurança: Prepared Statements**

Os filtros usam parâmetros preparados (?) para prevenir SQL Injection.

Demonstração

API em funcionamento

`localhost:8081/docs`

Swagger UI
Documentação interativa

Postman
Coleção de testes

CRUD + M:N
Todas as operações

CONCLUSÃO

Requisitos Cumpridos ✓

REQUISITOS OBRIGATÓRIOS

- Arquitetura REST
- 4 verbos HTTP (CRUD)
- 5 recursos diferentes
- Relação 1:N
- Representação JSON
- Documentação OpenAPI 3.0
- Coleção Postman
- MySQL + Node.js
- Docker multi-container

VALORIZAÇÕES

Filtros HTTP

Pesquisa em todos os recursos

Relação M:N

service_records ↔ mechanics

Repositórios:

GitHub: inf25dw1g33
DockerHub: inf25dw1g33

Obrigado!

César Ramos
A035224

Gonçalo Cidras
A046393

Jorge Espogeira
A046409

Grupo 33 • Desenvolvimento Web I • UMAIA 2024/25

github.com/inf25dw1g33