# Maintenance Car API — Complete Design & Implementation Guide

Generated: Comprehensive document including system analysis, architecture, repository pattern, EF Core models & mappings, EERD (text/PlantUML), API endpoints, sample code snippets, and implementation guidance.

## 1. System Analysis

1.1 Purpose

The Maintenance Car API will manage vehicle maintenance operations for a fleet or private garage: vehicles, maintenance records, mechanics, spare parts, suppliers, service types, scheduling and reporting.

1.2 Actors

- Admin (manage users, vehicles, inventory, reports)

- Mechanic (view assigned jobs, update maintenance records)

- Service Manager (create/approve work orders, schedule)

- External Supplier (supply spare parts, invoices)

1.3 High-level Use Cases

- CRUD vehicles

- Create/assign maintenance jobs

- Record maintenance activity and spare parts used

- Manage inventory and reorder parts

- Generate maintenance history and reports

1.4 Non-functional Requirements

- RESTful HTTP API using JSON

- Authentication: JWT-based

- Role-based authorization

- Persistence: SQL Server via EF Core

- Containerizable (Docker), configurable via appsettings

- Logging and error handling, pagination, filtering, versioning

## 2. Architecture & Project Structure

Proposed layered architecture:

- API (ASP.NET Core Web API): Controllers, DTOs, input validation

- Application (Services): Business logic, interfaces

- Domain (Entities): POCO entities, domain rules

- Infrastructure (EF Core, Repositories): DbContext, Repositories, Migrations

- Cross-cutting: Logging, Authentication, AutoMapper, Validation

Suggested solution folders and projects:

- src/YourCompany.Maintenance.Api (Web API)

- src/YourCompany.Maintenance.Application (services, DTOs, interfaces)

- src/YourCompany.Maintenance.Domain (entities, enums, exceptions)

- src/YourCompany.Maintenance.Infrastructure (EF Core, Repositories, Migrations)

- tests/YourCompany.Maintenance.\* (unit/integration tests)

### 2.1 Project File Tree (example)

/src  
 /YourCompany.Maintenance.Api  
 /Controllers  
 /DTOs  
 /Middlewares  
 Program.cs  
 appsettings.json  
 /YourCompany.Maintenance.Application  
 /Interfaces  
 /Services  
 /DTOs  
 /YourCompany.Maintenance.Domain  
 /Entities  
 /Enums  
 /YourCompany.Maintenance.Infrastructure  
 /Data  
 MaintenanceDbContext.cs  
 Migrations/  
 /Repositories  
 /Mappings  
 /tests  
 /Unit  
 /Integration

## 3. Domain Model & EERD

Core entities (with main properties):

1. Vehicle (Car)  
 - Id (GUID)  
 - LicensePlate (string)  
 - VIN (string)  
 - Make (string)  
 - Model (string)  
 - Year (int)  
 - CurrentMileage (int)  
 - Status (enum: Active, InService, Retired)  
  
2. MaintenanceRecord  
 - Id (GUID)  
 - VehicleId (FK)  
 - MaintenanceTypeId (FK)  
 - MechanicId (FK, nullable)  
 - StartDate (datetime)  
 - EndDate (datetime, nullable)  
 - Description (string)  
 - Cost (decimal)  
 - Status (enum: Planned, InProgress, Completed, Cancelled)  
  
3. Mechanic (User)  
 - Id (GUID)  
 - FullName (string)  
 - Email (string)  
 - Phone (string)  
 - Skills (string / JSON)  
  
4. SparePart  
 - Id (GUID)  
 - PartNumber (string)  
 - Name (string)  
 - QuantityOnHand (int)  
 - ReorderLevel (int)  
 - SupplierId (FK)  
  
5. Supplier  
 - Id (GUID)  
 - Name (string)  
 - ContactEmail (string)  
 - Phone (string)  
  
6. MaintenanceType  
 - Id (int)  
 - Name (string) e.g. OilChange, BrakeService  
 - DefaultIntervalKm (int)  
 - DefaultIntervalMonths (int)  
  
7. MaintenanceSparePart (join table)  
 - Id (GUID)  
 - MaintenanceRecordId (FK)  
 - SparePartId (FK)  
 - QuantityUsed (int)  
 - UnitCost (decimal)  
  
8. WorkOrder (optional advanced)  
 - Id (GUID)  
 - MaintenanceRecordId (FK)  
 - CreatedById (FK User)  
 - Priority  
 - DueDate

EERD (textual description):

- Vehicle 1..\* MaintenanceRecord (1 vehicle can have many maintenance records)

- MaintenanceRecord \*..\* SparePart via MaintenanceSparePart

- MaintenanceRecord -> Mechanic (optional assignment)

- SparePart -> Supplier (many parts per supplier)

PlantUML (you can paste this to an online PlantUML editor):

@startuml  
entity "Vehicle" as Vehicle {  
 \*Id : uuid  
 --  
 LicensePlate : string  
 VIN : string  
 Make : string  
 Model : string  
 Year : int  
 CurrentMileage : int  
}  
entity "MaintenanceRecord" as MR {  
 \*Id : uuid  
 --  
 VehicleId : uuid  
 MaintenanceTypeId : int  
 MechanicId : uuid  
 StartDate : datetime  
 EndDate : datetime  
 Description : string  
 Cost : decimal  
}  
entity "SparePart" as SP {  
 \*Id : uuid  
 --  
 PartNumber : string  
 Name : string  
 QuantityOnHand : int  
 ReorderLevel : int  
}  
entity "Mechanic" as Mech {  
 \*Id : uuid  
 --  
 FullName : string  
 Email : string  
}  
entity "Supplier" as Sup {  
 \*Id : uuid  
 --  
 Name : string  
}  
entity "MaintenanceSparePart" as MSP {  
 \*Id : uuid  
 --  
 MaintenanceRecordId : uuid  
 SparePartId : uuid  
 QuantityUsed : int  
 UnitCost : decimal  
}  
Vehicle ||--o{ MR  
MR }o--o{ MSP  
SP ||--o{ MSP  
SP }o--|| Sup  
MR }o--|| Mech  
@enduml

## 4. Repository & Unit of Work Design

Pattern summary:

- Use a Generic Repository for common CRUD operations and specific repositories for entity-specific queries.

- Use UnitOfWork to coordinate commits across multiple repositories.

Interface examples (C#):

public interface IRepository<T> where T : class  
{  
 Task<T> GetByIdAsync(Guid id);  
 Task<IEnumerable<T>> GetAllAsync();  
 Task AddAsync(T entity);  
 void Update(T entity);  
 void Remove(T entity);  
 Task<bool> ExistsAsync(Expression<Func<T, bool>> predicate);  
 IQueryable<T> Query();  
}

Unit of Work interface:

public interface IUnitOfWork : IDisposable  
{  
 IRepository<Vehicle> Vehicles { get; }  
 IRepository<MaintenanceRecord> MaintenanceRecords { get; }  
 IRepository<SparePart> SpareParts { get; }  
 Task<int> SaveChangesAsync();  
}

## 5. EF Core Implementation (DbContext & Mappings)

Example DbContext and Fluent API mappings (C#):

public class MaintenanceDbContext : DbContext  
{  
 public MaintenanceDbContext(DbContextOptions<MaintenanceDbContext> options)  
 : base(options) { }  
  
 public DbSet<Vehicle> Vehicles { get; set; }  
 public DbSet<MaintenanceRecord> MaintenanceRecords { get; set; }  
 public DbSet<SparePart> SpareParts { get; set; }  
 public DbSet<Mechanic> Mechanics { get; set; }  
 public DbSet<Supplier> Suppliers { get; set; }  
 public DbSet<MaintenanceSparePart> MaintenanceSpareParts { get; set; }  
  
 protected override void OnModelCreating(ModelBuilder modelBuilder)  
 {  
 base.OnModelCreating(modelBuilder);  
  
 modelBuilder.Entity<Vehicle>(e =>  
 {  
 e.HasKey(v => v.Id);  
 e.Property(v => v.LicensePlate).HasMaxLength(20).IsRequired();  
 e.HasIndex(v => v.VIN).IsUnique();  
 });  
  
 modelBuilder.Entity<MaintenanceRecord>(e =>  
 {  
 e.HasKey(m => m.Id);  
 e.HasOne(m => m.Vehicle)  
 .WithMany(v => v.MaintenanceRecords)  
 .HasForeignKey(m => m.VehicleId)  
 .OnDelete(DeleteBehavior.Cascade);  
 });  
  
 modelBuilder.Entity<MaintenanceSparePart>(e =>  
 {  
 e.HasKey(x => x.Id);  
 e.HasOne(x => x.MaintenanceRecord)  
 .WithMany(m => m.MaintenanceSpareParts)  
 .HasForeignKey(x => x.MaintenanceRecordId);  
 e.HasOne(x => x.SparePart)  
 .WithMany(p => p.MaintenanceSpareParts)  
 .HasForeignKey(x => x.SparePartId);  
 });  
 }  
}

## 6. DTOs, AutoMapper & Validation

Use DTOs for API surface. Example DTOs: VehicleDto, CreateVehicleDto, MaintenanceRecordDto, CreateMaintenanceRecordDto.

Use AutoMapper profiles to map between Entities and DTOs. Use FluentValidation for input validation in the API layer.

## 7. API Endpoints (example)

/api/v1/vehicles  
 GET / -> list (with pagination, filters)  
 GET /{id} -> get  
 POST / -> create  
 PUT /{id} -> update  
 DELETE /{id} -> delete  
  
/api/v1/maintenance  
 GET / -> list maintenance records (filters: vehicleId, status, date range)  
 GET /{id}  
 POST / -> create maintenance record (optionally create work order)  
 PUT /{id} -> update (status changes, assign mechanic)  
 POST /{id}/complete -> mark complete and add spare parts used  
  
/api/v1/spareparts  
 CRUD endpoints, inventory endpoints (adjust stock), reorder endpoints  
  
/api/v1/reports  
 GET /vehicle/{id}/history  
 GET /maintenance/summary?from=...&to=...

## 8. Sample Code: VehiclesController (ASP.NET Core)

[ApiController]  
[Route("api/v1/[controller]")]  
public class VehiclesController : ControllerBase  
{  
 private readonly IUnitOfWork \_uow;  
 private readonly IMapper \_mapper;  
  
 public VehiclesController(IUnitOfWork uow, IMapper mapper)  
 {  
 \_uow = uow;  
 \_mapper = mapper;  
 }  
  
 [HttpGet]  
 public async Task<IActionResult> Get([FromQuery] int page = 1, [FromQuery] int pageSize = 20)  
 {  
 var items = await \_uow.Vehicles.Query()  
 .Skip((page-1)\*pageSize)  
 .Take(pageSize)  
 .ToListAsync();  
 var dto = \_mapper.Map<IEnumerable<VehicleDto>>(items);  
 return Ok(dto);  
 }  
  
 [HttpPost]  
 public async Task<IActionResult> Create(CreateVehicleDto input)  
 {  
 var entity = \_mapper.Map<Vehicle>(input);  
 await \_uow.Vehicles.AddAsync(entity);  
 await \_uow.SaveChangesAsync();  
 return CreatedAtAction(nameof(GetById), new { id = entity.Id }, \_mapper.Map<VehicleDto>(entity));  
 }  
  
 [HttpGet("{id}")]  
 public async Task<IActionResult> GetById(Guid id)  
 {  
 var entity = await \_uow.Vehicles.GetByIdAsync(id);  
 if (entity == null) return NotFound();  
 return Ok(\_mapper.Map<VehicleDto>(entity));  
 }  
}

## 9. Database Migrations & Setup

Use EF Core tools:

# Install tools (dotnet CLI)  
dotnet tool install --global dotnet-ef  
  
# Add migration:  
dotnet ef migrations add InitialCreate -p YourCompany.Maintenance.Infrastructure -s YourCompany.Maintenance.Api  
  
# Update database:  
dotnet ef database update -p YourCompany.Maintenance.Infrastructure -s YourCompany.Maintenance.Api  
  
# Connection string example (appsettings.json):  
"ConnectionStrings": {  
 "MaintenanceDb": "Server=.;Database=MaintenanceDb;User Id=sa;Password=Your\_password123;"  
}

## 10. Features Checklist

- Authentication (JWT) & Authorization (roles)

- Vehicle management CRUD

- Maintenance job/work order lifecycle (plan, assign, execute, complete)

- Mechanic assignment and timesheets

- Inventory management for spare parts (stock, reorder alerts)

- Supplier management and purchase orders (optional)

- Reporting: vehicle history, cost summaries, parts usage

- Notifications (email/SMS) for due maintenance or low stock

- Audit logs and soft-delete

- API versioning, pagination, filtering, sorting

- Unit and integration tests

## 11. Deployment & DevOps Notes

- Containerize API with Docker (multi-stage builds)  
- Use environment variables for secrets; don't store production secrets in appsettings.json  
- CI/CD: run tests, build image, push to registry, deploy to Kubernetes/App Service  
- Backups: schedule DB backups and test restores  
- Monitoring: Application Insights or Prometheus + Grafana

## 12. Deliverables Included

- This Word document (design, analysis, code snippets)

- Suggested next steps: implement domain models in Domain project, scaffold DbContext, create repositories, write DTOs & AutoMapper profiles, build controllers, write tests.

- Optional: provide ERD PNG or PlantUML render (not included).