**Car Maintenance API Project Structure and Documentation**

This document outlines the complete structure for building an ASP.NET Core Web API for a Car Maintenance system. The system is designed to manage car details, owners, maintenance records, services, and related operations. I'll cover system analysis, repository design pattern, database implementation using Entity Framework Core (EF Core), Enhanced Entity-Relationship Diagram (EERD) with mapping, all features, and provide this as a comprehensive guide. You can copy-paste this entire response into a Microsoft Word document for formatting (e.g., add headings, tables, and diagrams). I've used Markdown-like formatting here for clarity, which Word can handle via paste options.

For code snippets, I've included C# examples. This assumes you're using .NET 8 or later. The project is structured as a clean architecture with separation of concerns.

**1. System Analysis**

**1.1 Overview**

The Car Maintenance API is a RESTful service for tracking vehicle maintenance. It allows users (e.g., mechanics, owners, admins) to manage cars, schedule services, record repairs, and generate reports. Key goals:

* Improve efficiency in tracking maintenance history.
* Reduce errors in scheduling and billing.
* Provide secure access with authentication (optional extension).

**1.2 Requirements**

**Functional Requirements:**

* Manage Cars: CRUD operations for car details (make, model, year, VIN, etc.).
* Manage Owners: CRUD for car owners (name, contact, address).
* Manage Maintenance Records: Log services like oil changes, tire rotations, with dates, costs, and descriptions.
* Manage Services: Predefined service types (e.g., oil change, brake repair) with costs.
* Reporting: Get maintenance history for a car or owner.
* Search and Filtering: By date, car ID, owner, etc.

**Non-Functional Requirements:**

* Performance: Handle up to 1000 records efficiently.
* Security: Use JWT for authentication (implement in controllers).
* Scalability: Use repository pattern for easy switching to other DBs.
* Data Integrity: Enforce relationships (e.g., a maintenance record must link to a car).
* Tech Stack: ASP.NET Core API, EF Core for ORM, SQL Server (or SQLite for dev).

**1.3 Use Cases**

* Actor: Owner
  + View my car's maintenance history.
  + Schedule a new service.
* Actor: Mechanic
  + Add a new maintenance record.
  + Update service status.
* Actor: Admin
  + Manage service types.
  + Generate reports.

**1.4 Assumptions and Constraints**

* One car can have multiple owners (historical), but primary owner for simplicity.
* No real-time notifications (can be added later).
* Currency: USD for costs.
* Database: Relational (SQL).

**2. Repository Design Pattern**

The Repository Pattern abstracts data access, promoting separation of concerns. It acts as an in-memory collection for entities, hiding EF Core details. We'll use a Generic Repository for reusability, plus specific repositories for complex queries.

**2.1 Benefits**

* Testability: Mock repositories for unit tests.
* Flexibility: Switch from EF Core to another ORM easily.
* Clean Code: Business logic in services, data ops in repositories.

**2.2 Implementation Structure**

* **IRepository<T>**: Generic interface for CRUD.
* **Repository<T>**: Implements IRepository<T> using DbContext.
* **IUnitOfWork**: Manages multiple repositories and saves changes.
* **UnitOfWork**: Implements IUnitOfWork, injecting DbContext.

**3. Implement DB using EF Core**

**3.1 Setup**

* Install packages: Microsoft.EntityFrameworkCore, Microsoft.EntityFrameworkCore.SqlServer, Microsoft.EntityFrameworkCore.Tools.
* Connection String in appsettings.json: "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=CarMaintenanceDb;Trusted\_Connection=True;"

**3.2 Entities**

Define models in Models/ folder.

**3.3 DbContext**

**3.4 Migrations**

Run Add-Migration InitialCreate and Update-Database in Package Manager Console.

**4. EERD & Mapping**

**4.1 Enhanced Entity-Relationship Diagram (EERD)**

Text-based representation (you can draw this in Word using shapes or tools like Lucidchart):

* **Entities**:
  + Owner (Id PK, Name, ContactNumber, Address)
  + Car (Id PK, Make, Model, Year, VIN, OwnerId FK)
  + ServiceType (Id PK, Name, DefaultCost)
  + MaintenanceRecord (Id PK, Date, Description, Cost, CarId FK, ServiceTypeId FK)
* **Relationships**:
  + Owner 1:N Car (One owner owns many cars; Car has one owner. Cascade delete.)
  + Car 1:N MaintenanceRecord (One car has many records; Record belongs to one car. Cascade delete.)
  + ServiceType 1:N MaintenanceRecord (One type used in many records; Record uses one type. Restrict delete to prevent orphaning.)

Diagram in ASCII (copy to Word for better visuals):

text

Owner ----(1:N)---- Car ----(1:N)---- MaintenanceRecord ----(N:1)---- ServiceType

| | |

Id (PK) Id (PK) Id (PK) Id (PK)

Name Make Date Name

ContactNumber Model Description DefaultCost

Address Year Cost

VIN CarId (FK)

OwnerId (FK) ServiceTypeId (FK)

**4.2 Mapping**

Using EF Core Fluent API (as shown in DbContext). For DTOs, use AutoMapper.

Install AutoMapper.Extensions.Microsoft.DependencyInjection.

In Program.cs: services.AddAutoMapper(typeof(MappingProfile));

**5. All Features**

**5.1 Project Structure**

text

**5.2 Services Layer (Optional for Complex Logic)**

Example: CarService for reports.

**5.3 Controllers (Features/Endpoints)**

All use [ApiController], inject IUnitOfWork and IMapper.

Example: CarsController.cs

Similar controllers for Owners, MaintenanceRecords, ServiceTypes. Add filtering (e.g., [FromQuery] params).

**5.4 Additional Features**

* Authentication: Add [Authorize] to endpoints, use Microsoft.AspNetCore.Identity.
* Validation: Use Data Annotations on models (e.g., [Required]).
* Error Handling: Global exception filter.
* Logging: Use ILogger.
* Testing: Unit tests for repositories/services using Moq.
* Swagger: Add Swashbuckle for API docs.

**6. Deployment and Best Practices**

* Host on Azure/IIS.
* Use HTTPS.
* Optimize queries with .Include() for eager loading.
* Handle concurrency with timestamps.

This completes the guide. Copy to Word, format as needed (e.g., add table of contents, code highlighting). If you need code generation or expansions, provide more details!