Car Maintenance System - Full Stack (.NET 8 + Angular 18)

# System Analysis

### Overview  
The Car Maintenance System is a full-stack solution built with ASP.NET Core 8 Web API and Angular 18.  
It allows management of cars, owners, maintenance records, and predictive maintenance through ML.NET integration.  
  
### Functional Requirements  
- CRUD operations for Cars, Owners, MaintenanceRecords, ServiceTypes.  
- Authentication/Authorization using JWT and ASP.NET Identity.  
- Real-time chat (SignalR) and notifications (MailKit).  
- Predictive maintenance using ML.NET.  
- Multi-language support (EN/ES).  
- Caching, background jobs, health checks, and CI/CD pipeline.  
  
### Non-Functional Requirements  
- Secure (JWT, HTTPS, secrets in environment variables).  
- Scalable (Docker, Redis, Azure App Service).  
- Maintainable (Clean Architecture, repository/unit of work pattern).  
- Observable (Serilog logging, health checks).  
  
### Assumptions  
- Admins manage the system; owners access their own cars.  
- Azure SQL, Redis, and Azure App Service are used for hosting.

# Architecture Overview

### Clean Architecture Layers  
- \*\*API Layer:\*\* Controllers, SignalR, Swagger, localization middleware.  
- \*\*Application Layer:\*\* DTOs, services, mapping, and business logic.  
- \*\*Infrastructure Layer:\*\* EF Core, repositories, MailKit, Hangfire, Redis.  
- \*\*Domain Layer:\*\* Entities, interfaces, and validation rules.  
- \*\*ClientApp:\*\* Angular 18 frontend integrated via SPA.  
  
### Folder Structure  
- CarMaintenance.Api/  
- CarMaintenance.Application/  
- CarMaintenance.Domain/  
- CarMaintenance.Infrastructure/  
- CarMaintenance.ClientApp/

# Entities & EF Core Mapping

Entities include Car, Owner, MaintenanceRecord, ServiceType, AppUser, and ChatMessage.  
  
Example entity:  
```csharp  
public class Car {  
 public int Id { get; set; }  
 public string Model { get; set; } = string.Empty;  
 public string PlateNumber { get; set; } = string.Empty;  
 public int OwnerId { get; set; }  
 public Owner Owner { get; set; }  
 public ICollection<MaintenanceRecord> MaintenanceRecords { get; set; } = new List<MaintenanceRecord>();  
}  
```  
  
EF Core mapping:  
```csharp  
modelBuilder.Entity<Car>()  
 .HasOne(c => c.Owner)  
 .WithMany(o => o.Cars)  
 .HasForeignKey(c => c.OwnerId);  
```

# Repository Pattern & UnitOfWork

Generic repository interface:  
```csharp  
public interface IGenericRepository<T> where T : class {  
 Task<IEnumerable<T>> GetAllAsync();  
 Task<T?> GetByIdAsync(int id);  
 Task AddAsync(T entity);  
 void Update(T entity);  
 void Delete(T entity);  
}  
```  
  
UnitOfWork example:  
```csharp  
public interface IUnitOfWork : IDisposable {  
 IGenericRepository<Car> Cars { get; }  
 IGenericRepository<Owner> Owners { get; }  
 Task<int> CompleteAsync();  
}  
```

# Authentication & JWT

- ASP.NET Identity manages users.  
- JWT token generation and validation.  
- AuthController handles register/login endpoints.  
  
Example:  
```csharp  
[HttpPost("login")]  
public async Task<IActionResult> Login(LoginDto dto) {  
 var user = await \_userManager.FindByNameAsync(dto.Username);  
 if (user == null || !await \_userManager.CheckPasswordAsync(user, dto.Password))  
 return Unauthorized();  
 var token = \_jwtService.GenerateToken(user);  
 return Ok(new { token });  
}  
```

# Predictive Maintenance (ML.NET)

ML.NET model training script:  
```csharp  
var mlContext = new MLContext();  
var data = mlContext.Data.LoadFromTextFile<CarData>("data.csv", hasHeader: true, separatorChar: ',');  
var pipeline = mlContext.Transforms.Concatenate("Features", "Mileage", "Age")  
 .Append(mlContext.BinaryClassification.Trainers.SdcaLogisticRegression());  
var model = pipeline.Fit(data);  
mlContext.Model.Save(model, data.Schema, "model.zip");  
```  
Prediction endpoint:  
```csharp  
[HttpPost("predict")]  
public IActionResult Predict([FromBody] CarData data) {  
 var result = \_mlService.Predict(data);  
 return Ok(result);  
}  
```

# Angular 18 Frontend

Frontend Features:  
- Angular Material UI and i18n support.  
- SignalR integration for chat.  
- Auth interceptor for JWT.  
- Lazy-loaded modules and route guards.  
  
Example `auth.service.ts`:  
```typescript  
login(credentials: any) {  
 return this.http.post(`${this.api}/auth/login`, credentials).pipe(  
 tap((res: any) => localStorage.setItem('token', res.token))  
 );  
}  
```  
  
Example `app.component.ts`:  
```typescript  
export class AppComponent implements OnInit {  
 constructor(private chatService: ChatService) {}  
 ngOnInit() {  
 this.chatService.connect();  
 }  
}  
```

# Docker & CI/CD

\*\*Dockerfile (multi-stage):\*\*  
```dockerfile  
FROM node:20 AS client-build  
WORKDIR /app  
COPY ClientApp ./ClientApp  
RUN npm install && npm run build -- --output-path=dist  
  
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base  
WORKDIR /app  
COPY --from=client-build /app/ClientApp/dist ./wwwroot  
COPY . .  
ENTRYPOINT ["dotnet", "CarMaintenance.Api.dll"]  
```  
  
\*\*docker-compose.yml:\*\*  
```yaml  
services:  
 api:  
 build: .  
 ports: ["5000:80"]  
 depends\_on: [db, redis]  
 db:  
 image: mcr.microsoft.com/mssql/server:2022-latest  
 environment:  
 SA\_PASSWORD: "Strong!Pass123"  
 ACCEPT\_EULA: "Y"  
 redis:  
 image: redis:alpine  
```  
  
\*\*GitHub Actions Workflow:\*\*  
```yaml  
name: CI/CD  
on: [push]  
jobs:  
 build:  
 runs-on: ubuntu-latest  
 steps:  
 - uses: actions/checkout@v3  
 - name: Build  
 run: dotnet build  
 - name: Test  
 run: dotnet test  
 - name: Docker Build & Push  
 run: |  
 docker build -t carmaintenanceapi .  
 docker push carmaintenanceapi  
```