



İSKENDERUN TEKNİK
ÜNİVERSİTESİ

Bilgisayar Mühendisliği

2022-2023 Öğretim Yılı

Bahar Dönemi

Bitirme Projesi Raporu

Rent A Car

Hazırlayan: Kutay TISTİK

182523059

Proje Tanıtımı:

Bu raporda sunulan bitirme projesi konusu proje adından da anlaşılacağı üzere bir araba kiralama sistemi web uygulamasıdır. Bu uygulamada oluşturduğum RentACar veri tabanında bulunan Brands, Cars, Colors, Customers, Rentals, Users tablolarındaki ürünler ile ilgili Entity

Framework yardımıyla Ekleme, Listeleme, Güncelleme, Silme işlemleri yapılabilmektedir. Uygulamada ayrıca bağımlı nesneleri kontrol etmek adına Autofac kütüphanesi, doğrulama işlemlerini kontrol etmek adına ise FluentValidation kütüphanesi kullanılmıştır. Uygulamada bulunan WebApi katmanı sayesinde ‘CRUD’ işlemlerini Postman rest client kullanarak test edebiliyoruz. Ayrıca AOP yapısı da uygulamada mevcuttur. Projede

R Araba Ekle/Sil/Güncelle/Listele

R Marka Ekle/Sil/Güncelle/Listele

R Renk Ekle/Sil/Güncelle/Listele

R Kullanıcı Ekle/Sil/Güncelle/Listele

R Müşteri Ekle/Sil/Güncelle/Listele

R Araba Kiralama Özelliği

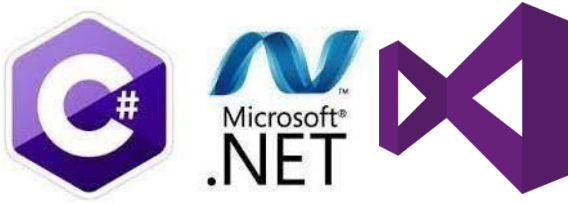
gibi imkanlar mevcuttur.

EntityFramework kullanılarak CRUD işlemlerinin yapıldığı, Validation işlemlerini Autofac paketi ile oluşturulan Aspectleri kullanarak gerçekleştiren, Wpf ara yüzü ile çalışan, araç kiralama iş yerlerine yönelik örnek bir web projesidir. Proje içerisinde data kaynakları ve bütün iş istekleri kolayca değiştirilebilir ve yeni nesneler eklenebilir. Yapılacak olan işlemler eski kodları bozmadan sürekli ekleme ile yapılabilir ve projeye farklı bir teknoloji

eklemek istediğimizde proje herhangi bir zorluk yaratmayacaktır. Tamamen "Plug and Play" prensibi göz önüne alınarak tasarlanmıştır. Proje sürdürülebilirlik prensibini yerine getirmektedir. Ayrıca proje katmanlı mimari kullanılarak geliştirilmiştir. Katmanlı mimarisi sayesinde kodun anlaşılabilirliği daha yüksektir. Tüm yazılan kodlar bir düzen içerisinde çalışmaktadır.

KULLANILAN TEKNOLOJİLER VE TEKNİKLER

1-Uygulama C# ile Visual Studio IDE üzerinde geliştirilmiştir.



a) Visual Studio :

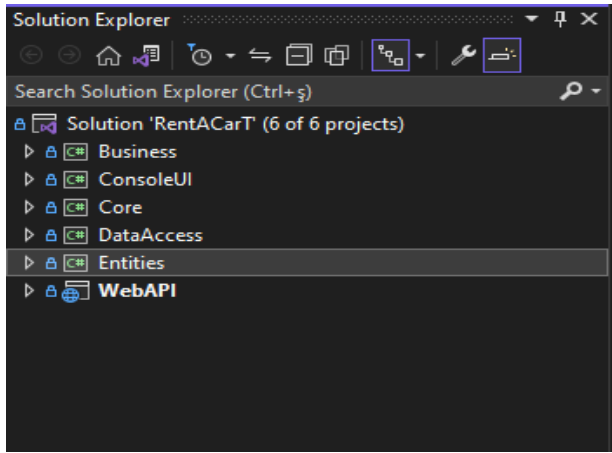
Microsoft Visual Studio, Microsoft tarafından geliştirilen bir tümleşik geliştirme ortamıdır (IDE). Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework ve Microsoft Silverlight tarafından desteklenen tüm platformlar için yönetilen kod ile birlikte yerel kod ve Windows Forms uygulamaları, web siteleri, web uygulamaları ve web servisleri ile birlikte konsol ve grafiksel kullanıcı arayüzü uygulamaları geliştirmek için kullanılır. Visual Studio, değişik programlama dillerini destekler, bu da kod editörü ve hata ayıklayıcısının neredeyse tüm programlama dillerini desteklemesini sağlamaktadır. Dahili diller C/C++ (Görsel yoluyla C++), VB.NET(Visual Basic .NET üzerinden), C# (Visual C# ile), ve F# (Visual Studio 2010 itibarıyla) içermektedir.

a.1) Visual studio neden kullanıldı ?

Bu projede visual studio kullanılmasının sebebi .NET Framework içindir. .NET Framework, Microsoft tarafından geliştirilen, açık İnternet protokolleri ve standartları üzerine kurulmuş bir "uygulama" geliştirme platformu. Daha önce Sun Microsystems tarafından geliştirilmiş olan Java platformuna önemli benzerlikler göstermektedir. Sistemin arka tarafta ihtiyaç duyulan işlemleri ve yönergeleri takip edebilmesi için .Net dili kullanılmıştır. Buradaki uygulama kavramının kapsamı çok geniştir. Bir masaüstü uygulamasından bir web tarayıcı uygulamasına kadar her şey bu platform içinde düşünülmüştür ve desteklenmiştir. Bu

uygulamaların birbirleriyle ve geliştirildiği ortam fark etmeksizin dünyadaki tüm uygulamalarla iletişimi için kolayca web servisleri oluşturulmasına imkân verilmiştir. Bu platform, işletim sisteminden ve donanımdan daha üst seviyede taşınabilir olarak tasarlanmıştır. .Net mimarisi, ortak bir yürütme ortamı, ortak bir değişken tür sistemi, ve devingen bağlantılı kütüphanelerden oluşur. .Net kütüphanesi eski visual basic için tasarlanmış API (programcılar için birçok fonksiyon) lerin sınıflanmış halidir. Çünkü API sınıflandırılmamış ve bu nedenler programcılar için bir kâbus halini almaktaydı. .Net kütüphanesi programın işletim sistemi ile kolayca uyum içinde çalışmasını sağlamıştır.

2-Proje , çok katmanlı mimariye uygun şekilde geliştirilmiştir.

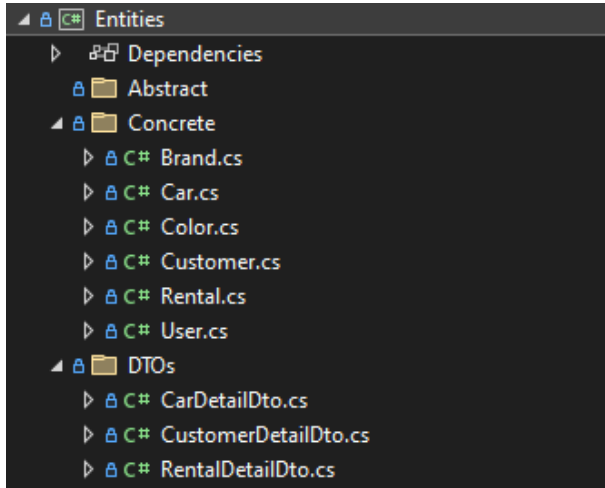


Şekil 1:Katmanlar



Entities Katmanı:

Entities Katmanı'nda Abstract , Dtos ve Concrete olmak üzere üç adet klasör bulunmaktadır. Concrete klasörü veri tabanından gelen somut nesnelerin özelliklerini tutmak için Abstract klasörü soyut nesneleri tutmak için Dtos klasörü ise veri tabanında birbiri ile ilişkili olan nesnelerin ilişkili özelliklerini birlikte kullanabilmek için oluşturulmuştur.

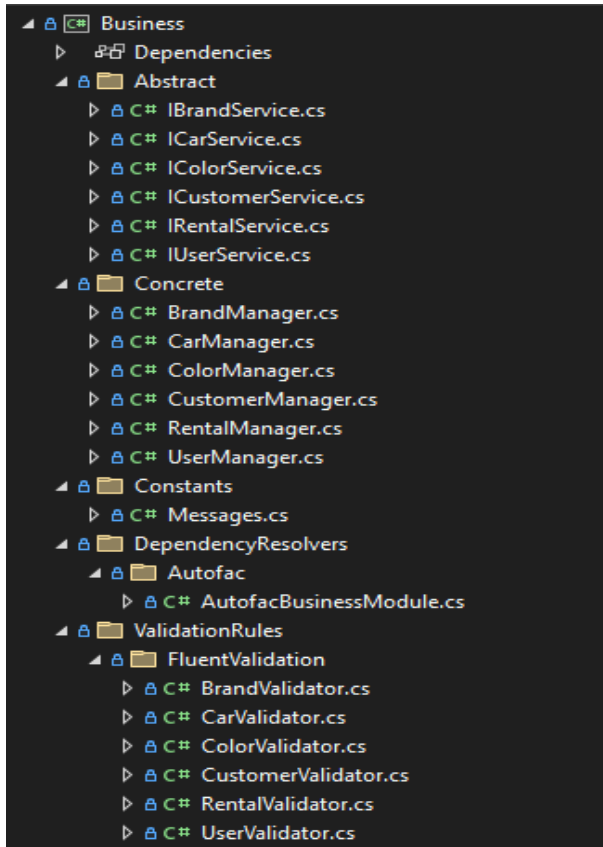


Şekil 2:Entities Katmanı İçeriği

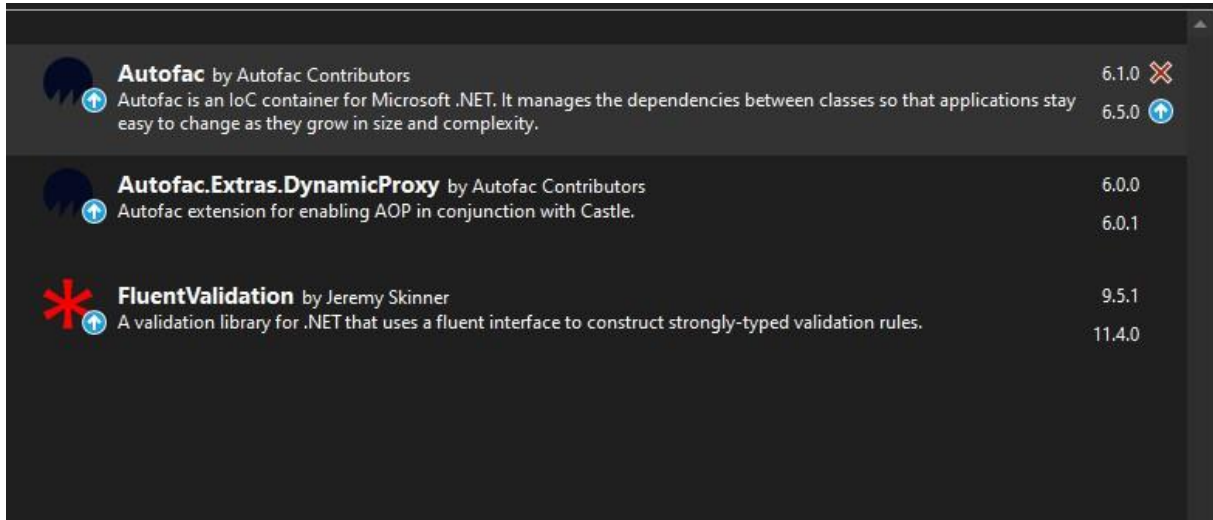


Business Katmanı :

Business Katmanında altyapısı hazır olan bütün servislerin yönetimleri yazıldı. Sürekli değişebilen iş kodlarımızı altyapıyı değiştirmeden ekleyebildiğimiz katmandır. Sürekliliğin korunduğu projemde birçok değişikliğin sadece burada yapılıyor olması yönetimi ve sürekli gelişimi çok kolaylaştırmaktadır.



Şekil 3:Business Katmanı İçeriği

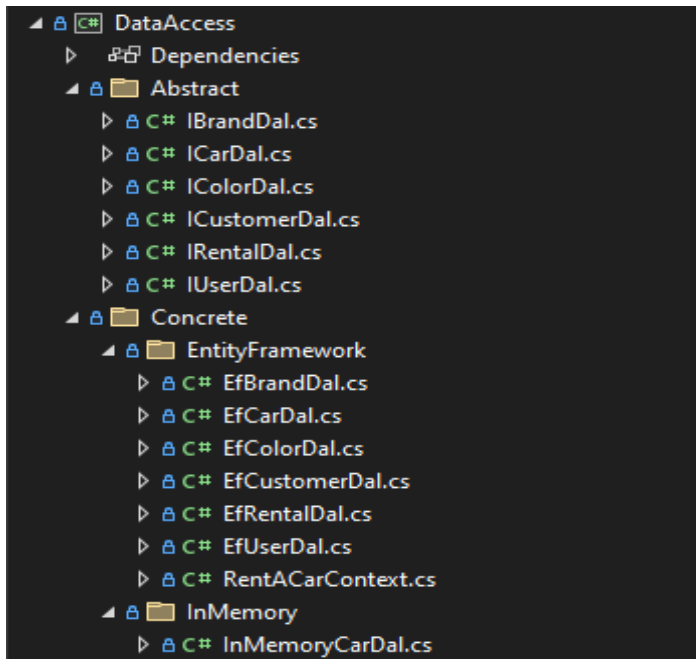


Şekil 4: Business Katmanında Kullanılan Paketler



Data Access Katmanı :

Data Access Katmanı'nda Abstract interfacerleri barındıran ve Concrete classları barındıran klasörler bulunmaktadır . Crud operasyonlarını Core katmanından miras alarak gerçekleştirmektedir. Gelebilecek iş kodları için altyapı burada hazırlanır. Objelerin data transferleri için kullanacağı data baseler ve varlıkların bağlantıları Data Access Katmanında yapılandırıldı.



Şekil 5: DataAccess Katmanı İçeriği

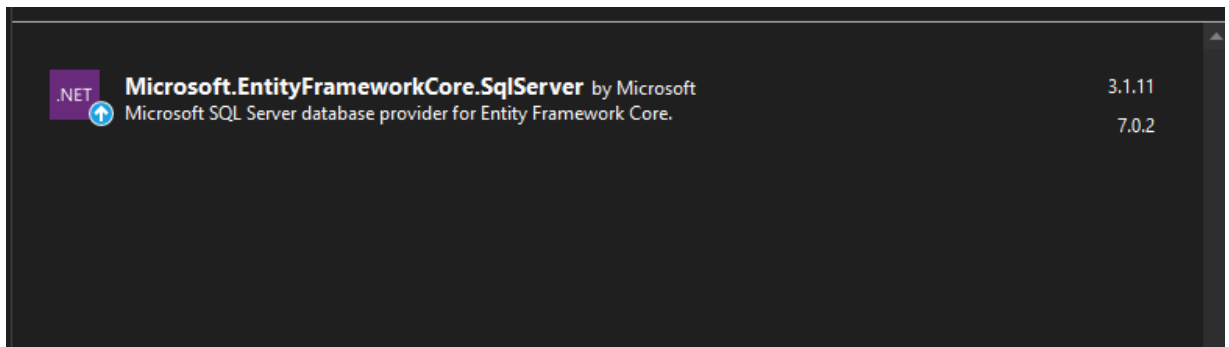


Şekil 6: DataAccess Katmanında Kullanılan Paketler

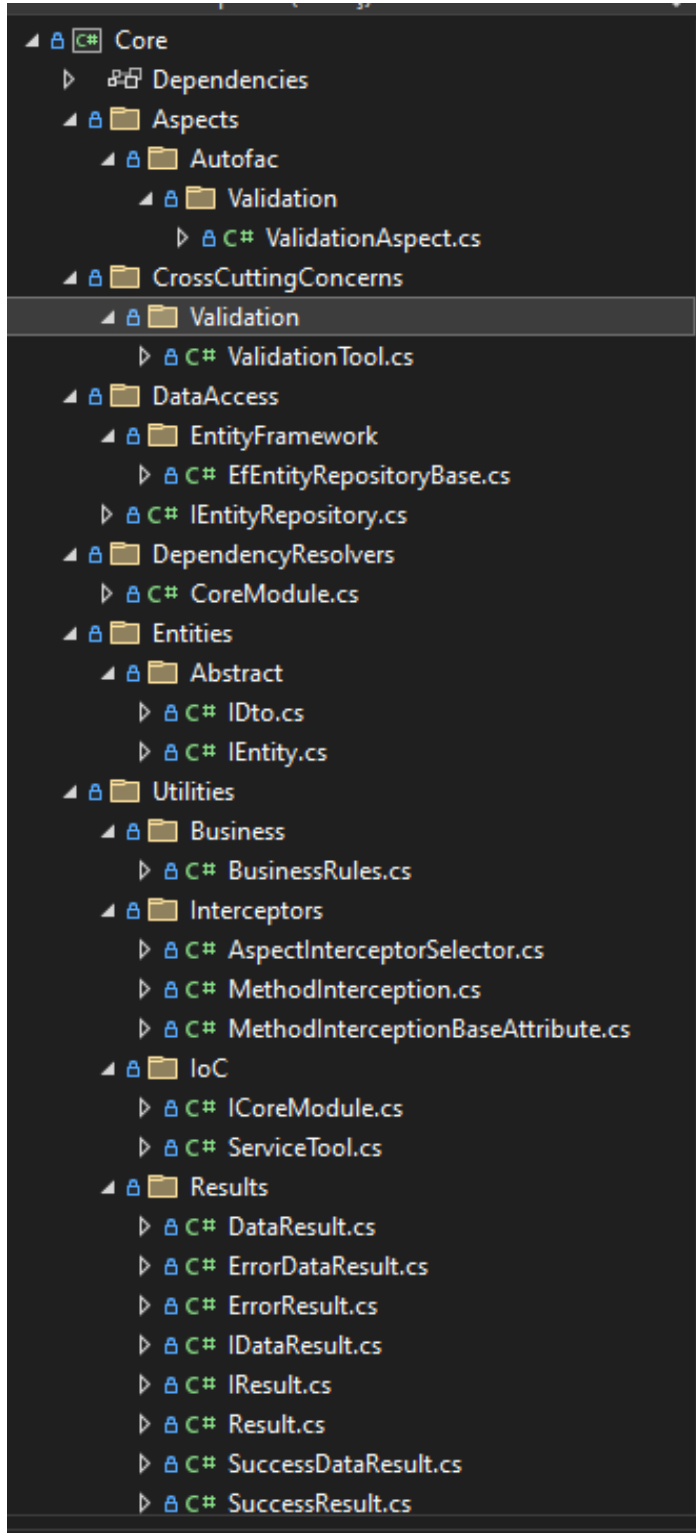


Core Katmanı :

Core Katmanı evrensel bir katmandır. Geliştirilecek her projede kullanılabilir, isimlendirme kuralları ve oluşturulma düzeni sebebi ile oldukça kullanışlıdır. Core Katmanında DataAccess, Entities, Utilities, Aspects, CrossCuttingConcerns, DependencyResolvers klasörleri bulunuyor. Aspects klasörü Validation işlemlerinin Autofac attribute altyapısını hazırlar. CrossCuttingConcerns klasöründe Validation yönetimi proje içerisinde, dikey katmanda dinamik çalışabilmesi için (generics) geliştirildi. DependencyResolvers klasöründe servis konfigürasyonları, DataAccess klasöründe bütün CRUD operasyonları ve DataBaseler generic olarak yapılandırıldı. Utilities içerisinde iş metodu kurallarının yönetimi kolaylaştırılacak, belge ekleme işlemleri kodlandı, Results yapısı kurularak hata yönetimi yapılandırıldı.



Şekil 7: Core Katmanında Kullanılan Paketler



Şekil 8:Core Katmanı İçeriği

ConsoleUI :

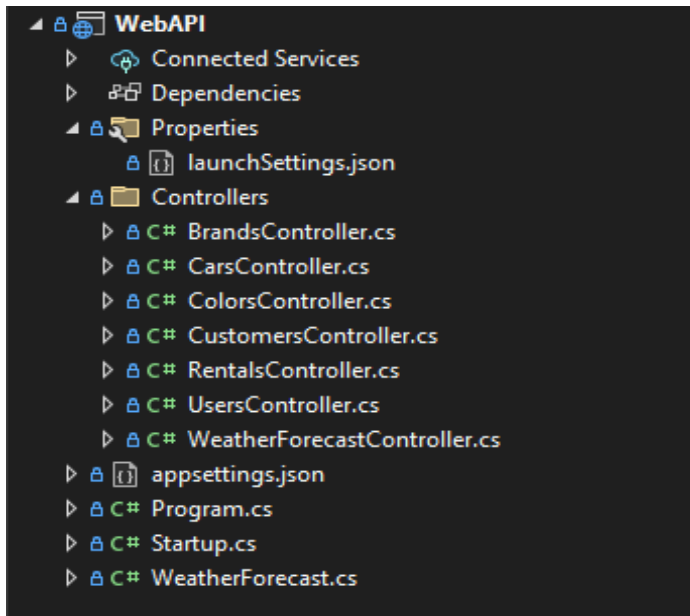
Yazdığımız kodları konsol ekranında geçici olarak test etmek için kullanıldı.

```
using Business.Concrete;
using DataAccess.Abstract;
using DataAccess.Concrete.EntityFramework;
using DataAccess.Concrete.InMemory;
using Entities.Concrete;
using System;

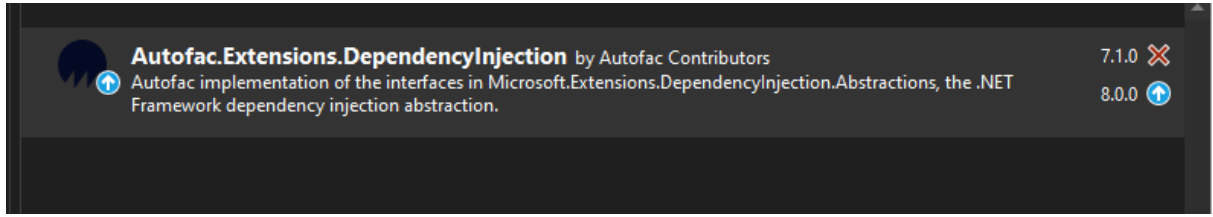
namespace ConsoleUI
{
    class Program
    {
        static void Main(string[] args)
        {
            ICustomerDal customerDal = new EfCustomerDal();
            CustomerManager customerManager = new CustomerManager(customerDal);
            var result = customerManager.GetCustomerDetail();
            if (result.Success)
            {
                foreach (var customer in result.Data)
                {
                    Console.WriteLine(customer.FirstName + " " + customer.LastName + " - " + customer.CompanyName);
                }
            }
            else
            {
                Console.WriteLine(result.Message);
            }
        }
    }
}
```

WebAPI Layer :

Uygulamamızı konsol haricinde Postman gibi bir rest client kullanarak internet tarayıcısı üzerinden de test edebiliriz. Bunu yapabilmek için Business katmanındaki tüm servislerin Api karşılığı yazıldı ve bu katmanda ki kodlar diğer uygulamalar ile iletişim kurmak için kullanıldı.




Şekil 9: WebAPI Katmanı İçeriği




Şekil 10: WebAPI Katmanında Kullanılan Paketler


3- Veri tabanı olarak MSSQL Kullanıldı.


Database and Tables


Brands	
	BrandID
	BrandName

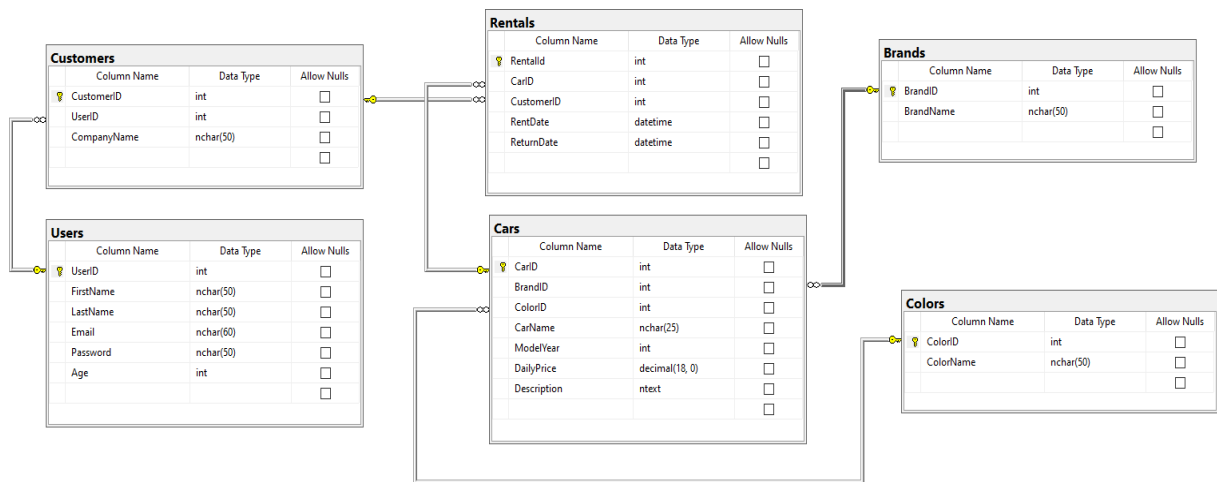
Customers	
	CustomerID
	UserID
	CompanyName

Cars	
	CarID
	BrandID
	ColorID
	CarName
	ModelYear
	DailyPrice
	Description

Rentals	
	RentalId
	CarID
	CustomerID
	RentDate
	ReturnDate

Users	
	UserID
	FirstName
	LastName
	Email
	Password
	Age

Colors	
	ColorID
	ColorName



Projedeki SQL Kodları:

```

CREATE TABLE [dbo].[Cars] (
    [CarID] INT IDENTITY (1, 1) NOT NULL,
    [BrandID] INT NOT NULL,
    [ColorID] INT NOT NULL,
    [CarName] NCHAR (25) NOT NULL,
    [ModelYear] INT NOT NULL,
    [DailyPrice] DECIMAL (18) NOT NULL,
    [Description] NTEXT NOT NULL,
    PRIMARY KEY CLUSTERED ([CarID] ASC),
    FOREIGN KEY ([BrandID]) REFERENCES [dbo].[Brands] ([BrandID]),
    FOREIGN KEY ([ColorID]) REFERENCES [dbo].[Colors] ([ColorID])
);
  
```

```

CREATE TABLE [dbo].[Users]
(
    [UserID] INT IDENTITY (1, 1) NOT NULL,
    [FirstName] NCHAR(50) NOT NULL,
    [LastName] NCHAR(50) NOT NULL,
    [Email] NCHAR(60) NOT NULL,
    [Password] NCHAR(50) NOT NULL,
    [Age] int NOT NULL,
    PRIMARY KEY CLUSTERED ([UserID] ASC),
)
  
```

```

CREATE TABLE [dbo].[Brands]
(
    [BrandID] INT IDENTITY (1, 1) NOT NULL,
    [BrandName] NCHAR(50) NOT NULL
    PRIMARY KEY CLUSTERED ([BrandID] ASC)
)
  
```

```

CREATE TABLE [dbo].[Customers]
(
    [CustomerID] INT IDENTITY (1, 1) NOT NULL,
    [UserID] INT NOT NULL,
    [CompanyName] NCHAR(50) NOT NULL
    PRIMARY KEY CLUSTERED ([CustomerID] ASC),
    FOREIGN KEY ([UserID]) REFERENCES [dbo].[Users] ([UserID]),
)
  
```

```

CREATE TABLE [dbo].[Colors]
(
    [ColorID] INT IDENTITY (1, 1) NOT NULL,
    [ColorName] NCHAR(50) NOT NULL
    PRIMARY KEY CLUSTERED ([ColorID] ASC)
)
  
```

```

CREATE TABLE [dbo].[Rentals]
(
    [RentalID] INT IDENTITY (1, 1) NOT NULL,
    [CarID] INT NOT NULL,
    [CustomerID] INT NOT NULL,
    [RentDate] DateTime NOT NULL,
    [ReturnDate] DateTime NOT NULL,
    PRIMARY KEY CLUSTERED ([RentalID] ASC),
    FOREIGN KEY ([CustomerID]) REFERENCES [dbo].[Customers] ([CustomerID]),
    FOREIGN KEY ([CarID]) REFERENCES [dbo].[Cars] ([CarID])
)
  
```



a) Microsoft SQL Server :

Microsoft SQL Server™, verilerin güvenli ve bütünlük içerisinde depolanmasını ve aynı anda birden fazla kullanıcı tarafından erişilmesini sağlayan kurumsal çaplı bir ilişkisel veri tabanı yönetim sistemidir (RDBMS). Birbiriyle ilişkili verilerin sistematik bir şekilde kaydedilmesini ve bu verilerden beslenen uygulamalar tarafından ihtiyaç anında kullanılmasına olanak sağlar.

a.1)Microsoft SQL Server neden kullanıldı?

ASP.NET programlama diliyle daha hızlı ve entegre kullanılabileceği için bu SQL dili tercih edildi. Bu proje için bazı avantajları şunlardır: Verileri taşımadan veya çoğaltmadan SQL Server ile tüm veri varlıklarında sorgulama yaparak verilerden iç görüler elde edilmesi, SQL Server'ın, Windows ve Linux işletim sistemleri ile kullanılması, Veri sınıflandırması, veri koruması ve izleme yaparak uyarılar için yerleşik özelliklerin kullanılması, SQL Server'ın şüpheli etkinlikleri izlemesi, tanımlanması ve bunlarla ilgili uyarılar yapılması, Hatta güvenlik boşluklarını ve hatalı yapılandırmaları tespit edip bunlarla ilgili sorunların giderilmesi

4-Projede SOLID ilkelerine uygun olarak OOP teknikleri kullanıldı .

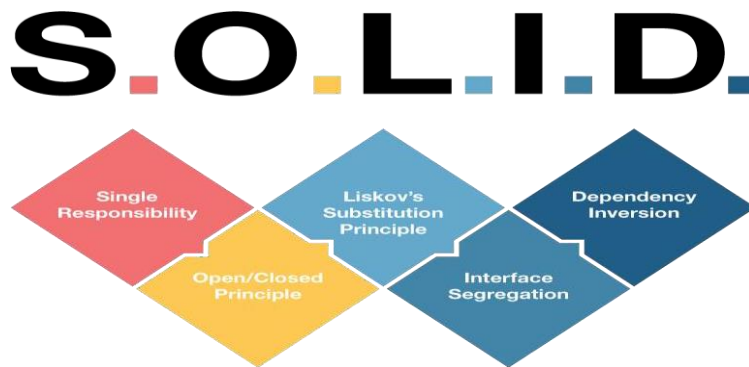
S—Single Responsibility Principle : Bir sınıfın yapması gereken sadece bir iş olacak şekilde tasarlandı.

O—Open-Closed Principle : Projeye yeni bir özellik eklendiğinde var olan kodlar geliştirilebilir ama değiştirilmez şekilde oluşturuldu.

L — Liskov Substitution Principle : Kodlarda herhangi bir değişiklik yapmaya gerek duymadan alt sınıfları, türedikleri(üst) sınıfların yerine kullanılabilir.

I — Interface Segregation Principle : Sorumlulukların hepsini tek bir ara yüze toplamak yerine daha özelleştirilmiş birden fazla ara yüz oluşturuldu.

D — Dependency Inversion Principle : Bir sınıf diğer bir sınıfa doğrudan bağımlı olmayacak şekilde kurgulandı.



5-Entity Framework :

Kullanacağım veri tabanı ile projedeki nesneleri bağlamak ve veri alışverişi yapmak için kullanıldı.

```
7 references
public void Add(TEntity entity)
{
    using (TContext context = new TContext())
    {
        var addedEntity = context.Entry(entity);
        addedEntity.State = EntityState.Added;
        context.SaveChanges();
    }
}

7 references
public void Delete(TEntity entity)
{
    using (TContext context = new TContext())
    {
        var deletedEntity = context.Entry(entity);
        deletedEntity.State = EntityState.Deleted;
        context.SaveChanges();
    }
}

3 references
public TEntity Get(Expression<Func<TEntity, bool>> filter)
{
    using (TContext context = new TContext())
    {
        return context.Set<TEntity>().SingleOrDefault(filter);
    }
}

9 references
public List<TEntity> GetAll(Expression<Func<TEntity, bool>> filter = null)
{
    using (TContext context = new TContext())
    {
        return filter == null ? context.Set<TEntity>().ToList() : context.Set<TEntity>().Where(filter).ToList();
    }
}
```

6- LINQ(Language Integrated Query) :

Projedeki sorgular için kullanıldı.

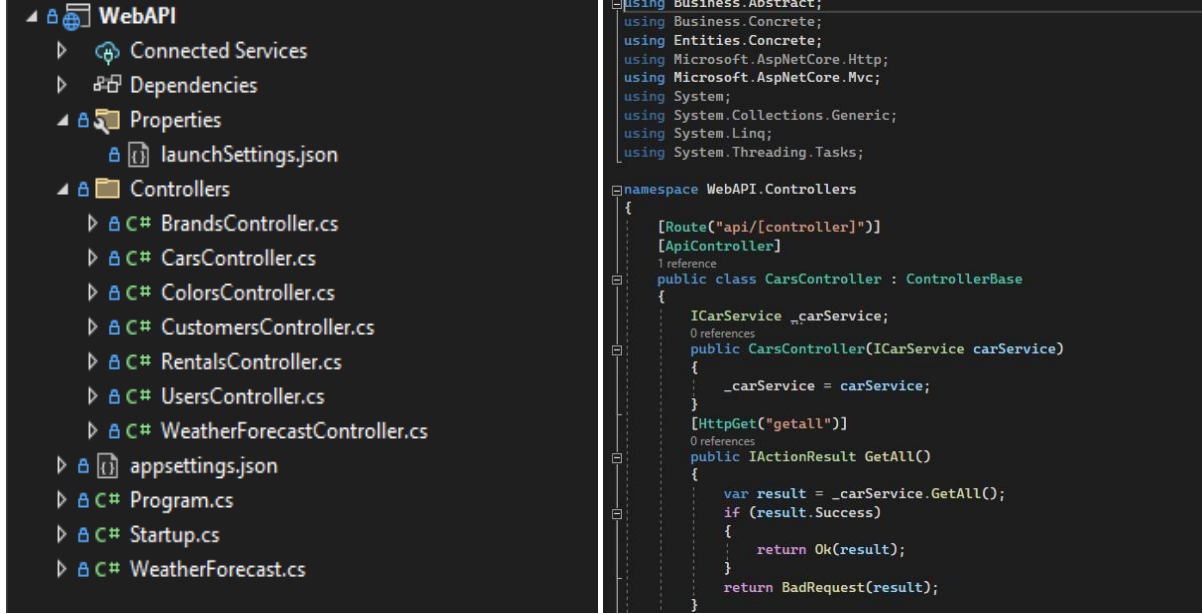
```
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Text;

namespace DataAccess.Concrete.EntityFramework
{
    1 reference
    public class EfCarDal : EfEntityRepositoryBase<Car, RentACarContext>, ICarDal
    {
        2 references
        public List<CarDetailDto> GetCarDetails()
        {
            using (RentACarContext context = new RentACarContext())
            {
                var result = from c in context.Cars
                             join b in context.Brands
                             on c.BrandId equals b.BrandId
                             join co in context.Colors
                             on c.ColorId equals co.ColorId
                             select new CarDetailDto
                             {
                                 CarName = c.CarName,
                                 BrandName = b.BrandName,
                                 ColorName = co.ColorName,
                                 DailyPrice = c.DailyPrice
                             };

                return result.ToList();
            }
        }
    }
}
```

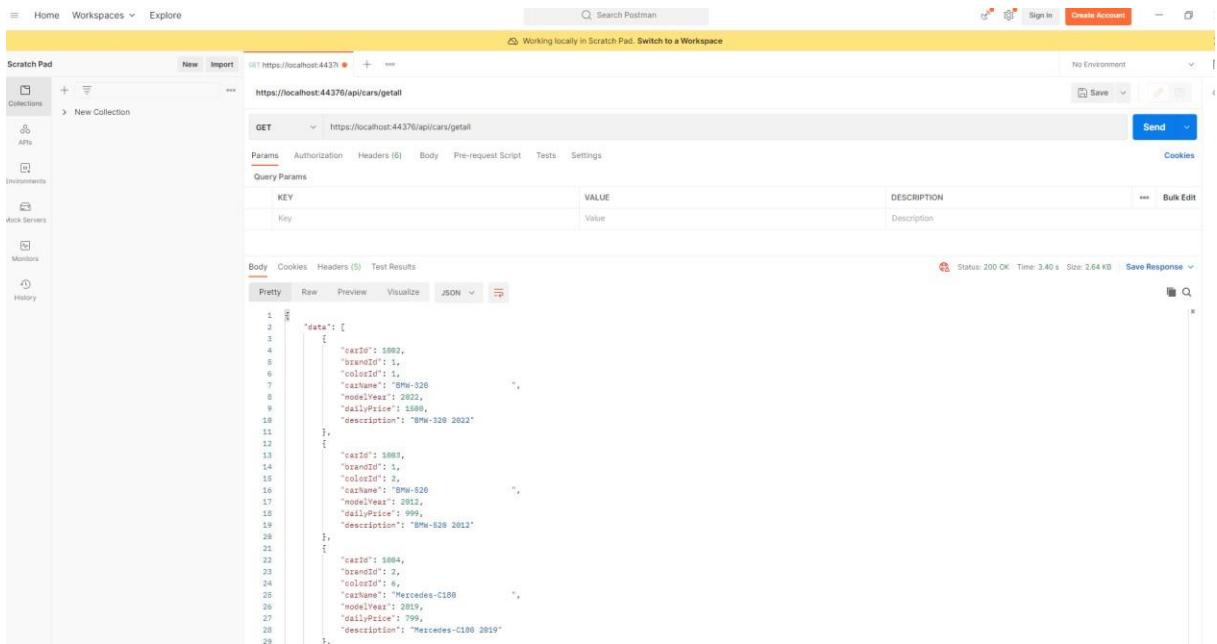
7-Web API :

Uygulamamızı konsol haricinde Postman gibi bir rest client kullanarak internet tarayıcısı üzerinden de test edebiliriz. Bunu yapabilmek için Business katmanındaki tüm servislerin Api karşılığı yazılarak ve bu katman ki kodlar Front-End tarafı ve diğer uygulamalar ile iletişim kurmak için kullanılıyor.



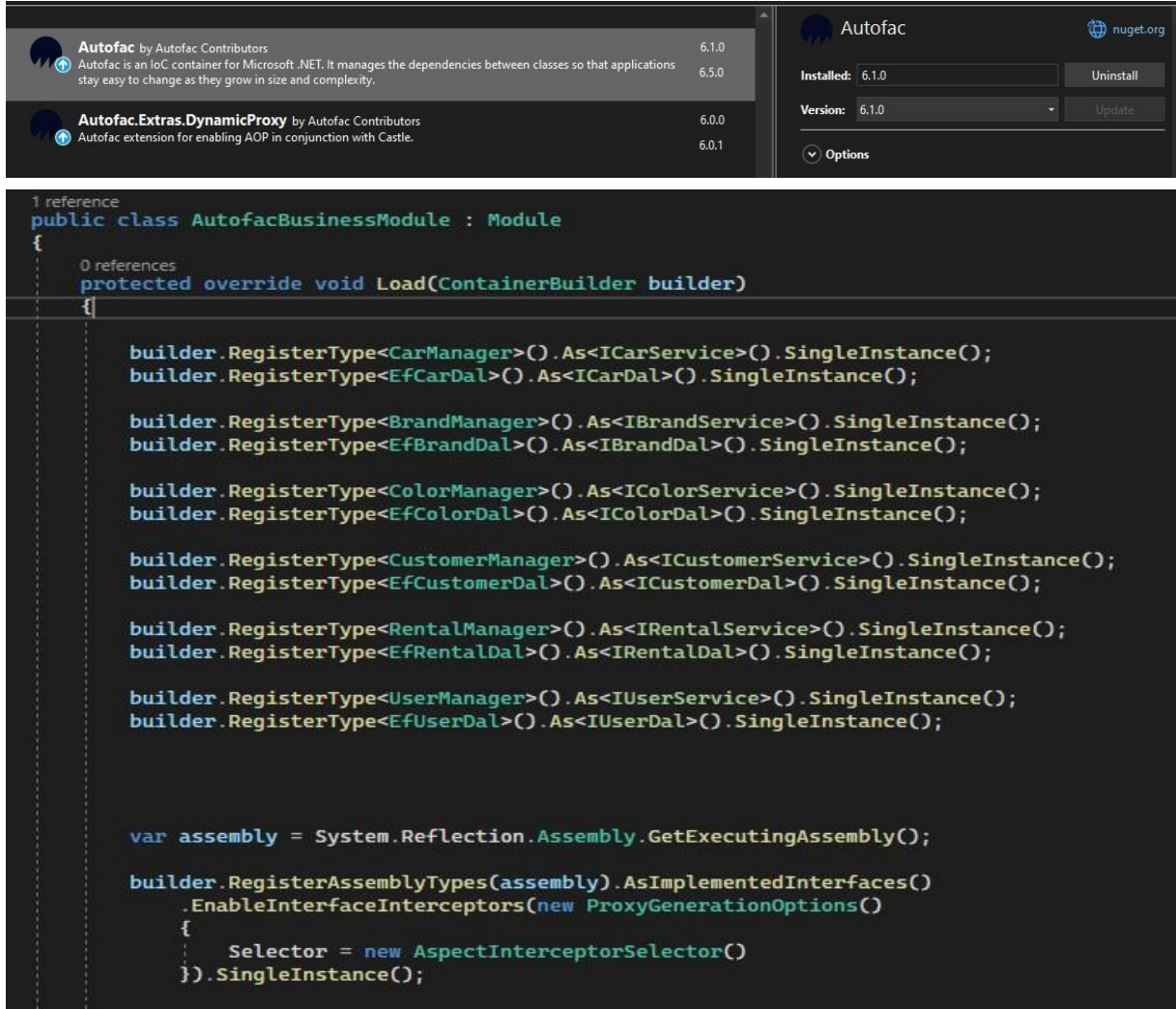
8-Postman :

Projemizde test ortamı için Postman kullanıldı çünkü Postman, özellikle test etme ve hata ayıklama söz konusu olduğunda, API'lerle çalışan geliştiriciler için yararlı olabilecek güçlü ve esnek bir araçtır.



9-Autofac :

Bağımlı nesneleri kontrol etmek için kullanıldı.



The screenshot shows the NuGet package manager interface. On the left, a list of packages is displayed:

Package Name	Version
Autofac by Autofac Contributors	6.1.0
Autofac.Extras.DynamicProxy by Autofac Contributors	6.0.0

On the right, the details for the 'Autofac' package are shown, including the 'Installed' version (6.1.0) and an 'Uninstall' button. Below the package list, the code for the 'AutofacBusinessModule' class is displayed:

```
1 reference
public class AutofacBusinessModule : Module
{
    0 references
    protected override void Load(ContainerBuilder builder)
    {
        builder.RegisterType<CarManager>().As<ICarService>().SingleInstance();
        builder.RegisterType<EfCarDal>().As<ICarDal>().SingleInstance();

        builder.RegisterType<BrandManager>().As<IBrandService>().SingleInstance();
        builder.RegisterType<EfBrandDal>().As<IBrandDal>().SingleInstance();

        builder.RegisterType<ColorManager>().As<IColorService>().SingleInstance();
        builder.RegisterType<EfColorDal>().As<IColorDal>().SingleInstance();

        builder.RegisterType<CustomerManager>().As<ICustomerService>().SingleInstance();
        builder.RegisterType<EfCustomerDal>().As<ICustomerDal>().SingleInstance();

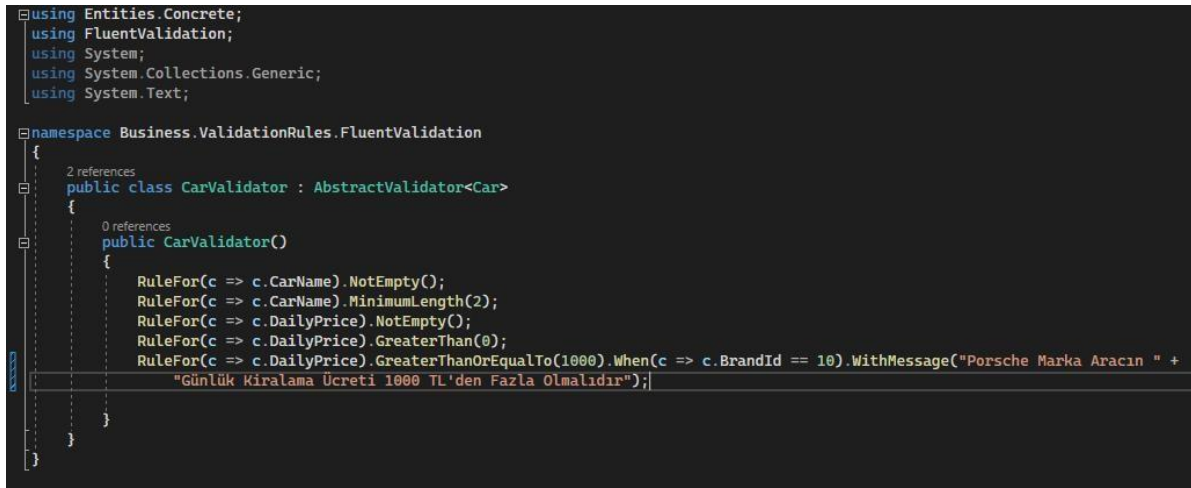
        builder.RegisterType<RentalManager>().As<IRentalService>().SingleInstance();
        builder.RegisterType<EfRentalDal>().As<IRentalDal>().SingleInstance();

        builder.RegisterType<UserManager>().As<IUserService>().SingleInstance();
        builder.RegisterType<EfUserDal>().As<IUserDal>().SingleInstance();

        var assembly = System.Reflection.Assembly.GetExecutingAssembly();
        builder.RegisterAssemblyTypes(assembly).AsImplementedInterfaces()
            .EnableInterfaceInterceptors(new ProxyGenerationOptions()
            {
                Selector = new AspectInterceptorSelector()
            }).SingleInstance();
    }
}
```

10- Validation(Fluent Validation) :

Doğrulama işlemlerini kontrol etmek için kullanılıyor.



The screenshot shows a C# code file with the following structure:

```
using Entities.Concrete;
using FluentValidation;
using System;
using System.Collections.Generic;
using System.Text;

namespace Business.ValidationRules.FluentValidation
{
    2 references
    public class CarValidator : AbstractValidator<Car>
    {
        0 references
        public CarValidator()
        {
            RuleFor(c => c.CarName).NotEmpty();
            RuleFor(c => c.CarName).MinimumLength(2);
            RuleFor(c => c.DailyPrice).NotEmpty();
            RuleFor(c => c.DailyPrice).GreaterThan(0);
            RuleFor(c => c.DailyPrice).GreaterThanOrEqualTo(1000).When(c => c.BrandId == 10).WithMessage("Porsche Marka Aracın " +
                "Günlük Kiralama Ücreti 1000 TL'den Fazla Olmalıdır");
        }
    }
}
```


11- Generic Repository Design Pattern :

Veri tabanı sorgulama işlemlerinin bir merkezden yapılmasını sağlayarak Business katmanına taşınmasını önleyerek sorgu ve kod tekrarı azaltılmasını sağlıyor.

```
6 references
public class EfEntityRepositoryBase<TEntity, TContext> : IEntityRepository<TEntity>
    where TEntity : class, IEntity, new()
    where TContext : DbContext, new()
{
}

1 reference
public class EfCarDal : EfEntityRepositoryBase<Car, RentACarContext>, ICarDal
{
}
```

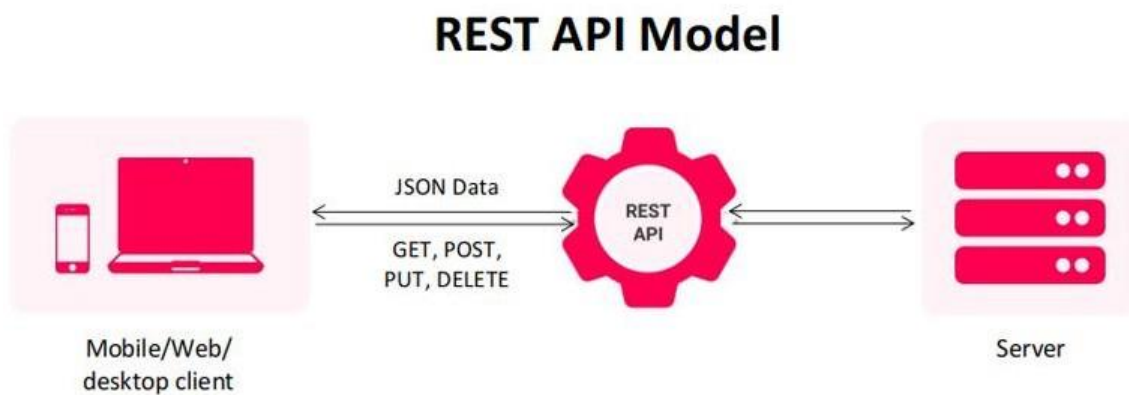
12- Disposable Pattern :

Bazı nesneler bellekte çok fazla yer kapladığından dolayı bu nesnelerin kullanımı bittikten sonra bellek yönetimi yapmak için kullanıldı.

```
using (TContext context = new TContext())
{
    var addedEntity = context.Entry(entity);
    addedEntity.State = EntityState.Added;
    context.SaveChanges();
}
```

13- Restful API :

İki bilgisayar sisteminin internet üzerinden güvenli bir şekilde bilgi alışverişi yapmak için kullanacağımız arabirimdir.



14- Adapter Pattern :

Adapter birbiriyle uyumlu olmayan arayüzlere (interface) sahip nesnelerin birlikte çalışabilmelerini sağlayacak olan yapısal tasarım deseni.

15- IoC(Inversion Of Control) :

Inversion Of Control kullanımındaki amaç Loosely Coupled (bağımlılığı az) sınıflar oluşturulmaktır. Projemde IoC altyapısını kullanırken .NET in sağlamış olduğu IoC yapısı yerine projeye AOP desteği eklendiği için Autofac'in sunmuş olduğu IoC altyapısı kullanılmıştır.

16- Interceptor :

Interceptor metot çağrımları sırasında araya girerek kesişen bilgilerimizi işletmemizi ve yönetmemizi sağlamaktadır.

```
1 reference
public abstract class MethodInterception : MethodInterceptionBaseAttribute
{
    2 references
    protected virtual void OnBefore(IInvocation invocation) { }
    1 reference
    protected virtual void OnAfter(IInvocation invocation) { }
    1 reference
    protected virtual void OnException(IInvocation invocation, System.Exception e) { }
    1 reference
    protected virtual void OnSuccess(IInvocation invocation) { }
    1 reference
    public override void Intercept(IInvocation invocation)
    {
        var isSuccess = true;
        OnBefore(invocation);
        try
        {
            invocation.Proceed();
        }
        catch (Exception e)
        {
            isSuccess = false;
            OnException(invocation, e);
            throw;
        }
        finally
        {
            if (isSuccess)
            {
                OnSuccess(invocation);
            }
        }
        OnAfter(invocation);
    }
}
```

Şekil 11: Projedeki Interceptor Kullanımı

18- AOP (Aspect Oriented Programming) :

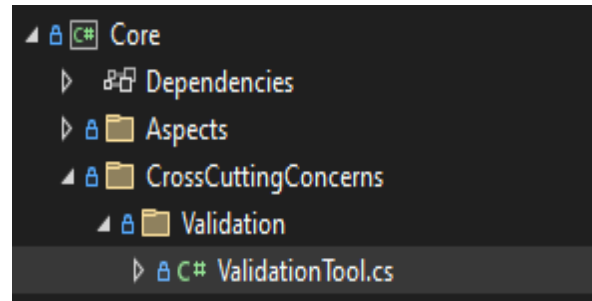
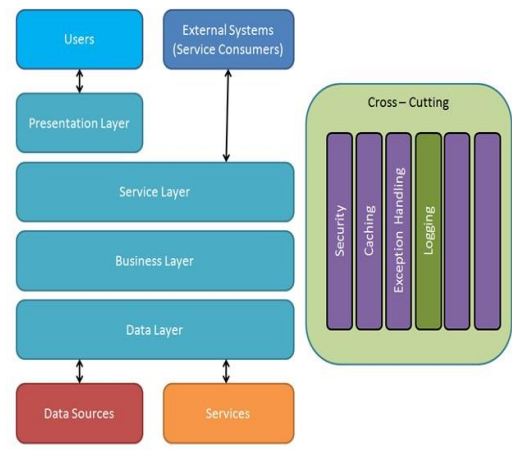
AOP(Aspect Oriented Programming) tekrar eden kod bloklarından sayısını azaltmayı sağlayarak temiz ve anlaşılır bir yapı sunuyor aynı zamanda Modüler bir yapı elde etmemizi sağlıyor.

```
}  
  
[ValidationAspect(typeof(CarValidator))]  
2 references  
public IResult Add(Car car)  
{  
    //business codes  
  
    _carDal.Add(car);  
    return new SuccessResult(Messages.CarAdded);  
}
```

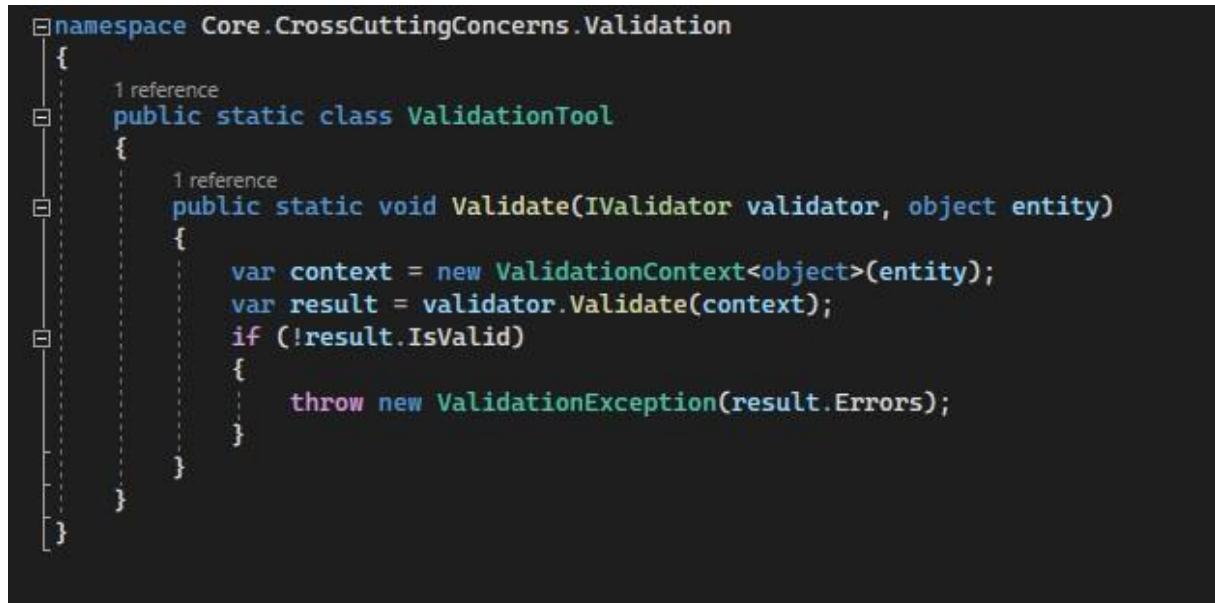
Şekil 12:Proje İçerisindeki Örnek AOP Kullanımı

19- Cross Cutting Concerns :

AOP birbiriyle kesişen ilgilerin (Cross Cutting Concerns) ayrılması üzerinedir. Uygulama genelinde kullanılacak olan yapıları Core katmanında yazdığımız koddan ayırarak bir çeşit ayrı küçük programcıklar şeklinde yazıp projede kullanıldı.



Şekil 13:Projedeki CrossCuttingConcerns Klasör Yapılandırması



Şekil 14:Projedeki CrossCuttingConcerns Kullanımı

Proje Görselleri

GET https://localhost:44376/api/cars/getall

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 4.06 s Size: 2.64 KB Save Response

Pretty Raw Preview Visualize JSON

```
1  {
2    "data": [
3      {
4        "carId": 1002,
5        "brandId": 1,
6        "colorId": 1,
7        "carName": "BMW-320",
8        "modelYear": 2022,
9        "dailyPrice": 1500,
10       "description": "BMW-320 2022"
11     },
12     {
13       "carId": 1003,
14       "brandId": 1,
15       "colorId": 2,
16       "carName": "BMW-520",
17       "modelYear": 2012,
18       "dailyPrice": 999,
19       "description": "BMW-520 2012"
20     },
21     {
22       "carId": 1004,
23       "brandId": 2,
24       "colorId": 6,
25       "carName": "Mercedes-C180",
26       "modelYear": 2019,
27       "dailyPrice": 799,
28       "description": "Mercedes-C180 2019"
29     },
30     {
31       "carId": 1005,
```

Şekil 15:Araba Listeleme

GET https://localhost:44376/api/cars/getbyid?id=1002

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
id	1002	
Key	Value	Description

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 20 ms Size: 368 B Save Response

Pretty Raw Preview Visualize JSON

```
1  {
2    "data": {
3      "carId": 1002,
4      "brandId": 1,
5      "colorId": 1,
6      "carName": "BMW-320",
7      "modelYear": 2022,
8      "dailyPrice": 1500,
9      "description": "BMW-320 2022"
10    },
11    "success": true,
12    "message": "Araba listelendi"
13  }
```

Şekil 16:Araba Numarasına Göre Araba Listeleme

GET https://localhost:44376/api/cars/getbybrand?id=2

Params Authorization Headers (6) Body Pre-request Script Tests Settings

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> id	2	
Key	Value	Description

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 32 ms Size: 531 B Save Response

Pretty Raw Preview Visualize JSON

```
1  {
2    "data": [
3      {
4        "carId": 1004,
5        "brandId": 2,
6        "colorId": 6,
7        "carName": "Mercedes-C180",
8        "modelYear": 2019,
9        "dailyPrice": 799,
10       "description": "Mercedes-C180 2019"
11     },
12     {
13       "carId": 1005,
14       "brandId": 2,
15       "colorId": 4,
16       "carName": "Mercedes-A180",
17       "modelYear": 2014,
18       "dailyPrice": 499,
19       "description": "Mercedes-A180 2014"
20     }
21   ],
22   "success": true,
23   "message": "Arabalar Listelendi"
24 }
```

Şekil 17:Araba Markasına Göre Listeleme

GET https://localhost:44376/api/cars/getbycolor?id=5

Params Authorization Headers (6) Body Pre-request Script Tests Settings

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> id	5	
Key	Value	Description

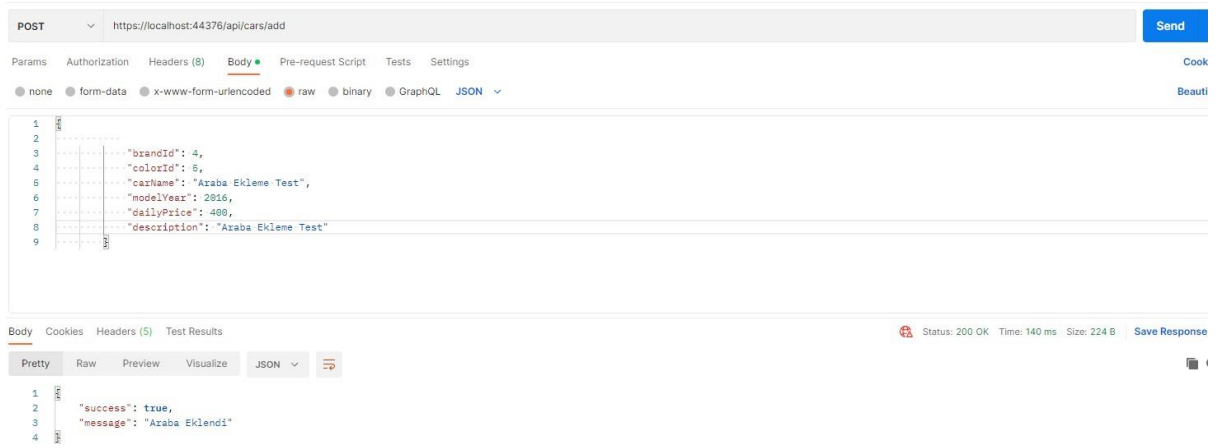
Body Cookies Headers (5) Test Results

Status: 200 OK Time: 28 ms Size: 377 B Save Res

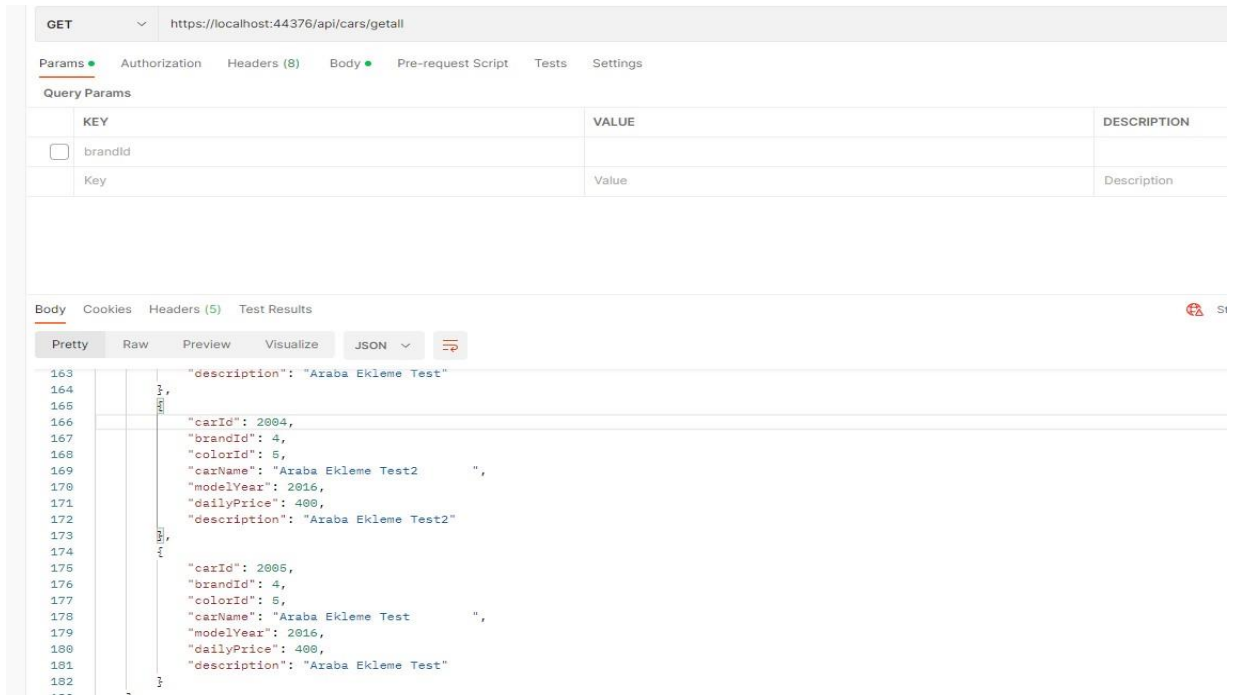
Pretty Raw Preview Visualize JSON

```
1  {
2    "data": [
3      {
4        "carId": 1007,
5        "brandId": 4,
6        "colorId": 6,
7        "carName": "Renault-Clio",
8        "modelYear": 2016,
9        "dailyPrice": 400,
10       "description": "Renault-Clio 2016"
11     }
12   ],
13   "success": true,
14   "message": "Arabalar Listelendi"
15 }
```

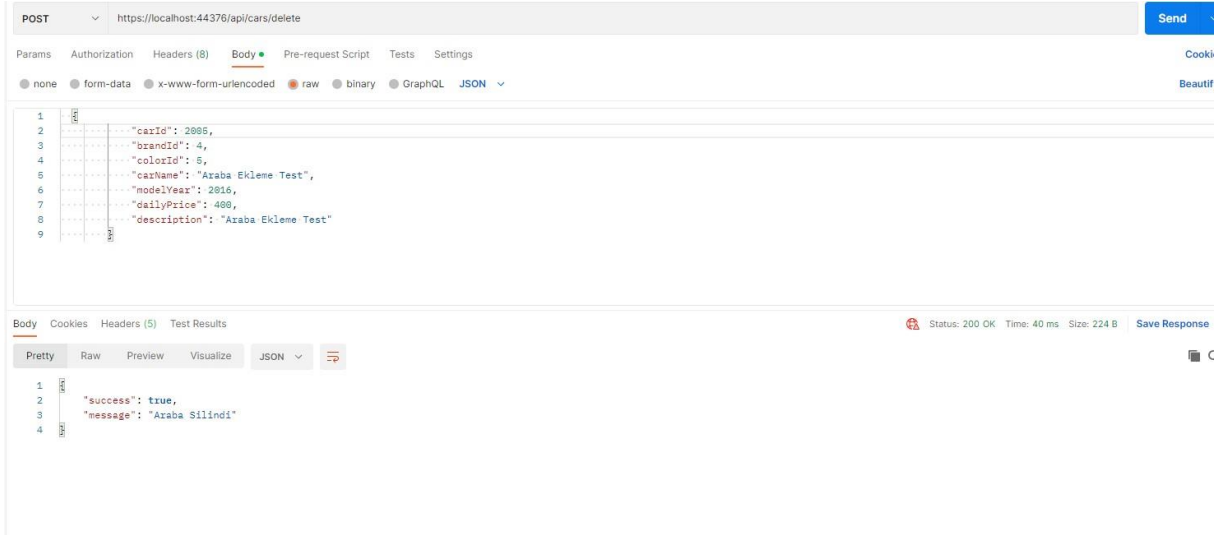
Şekil 18:Araba Rengine Göre Listeleme



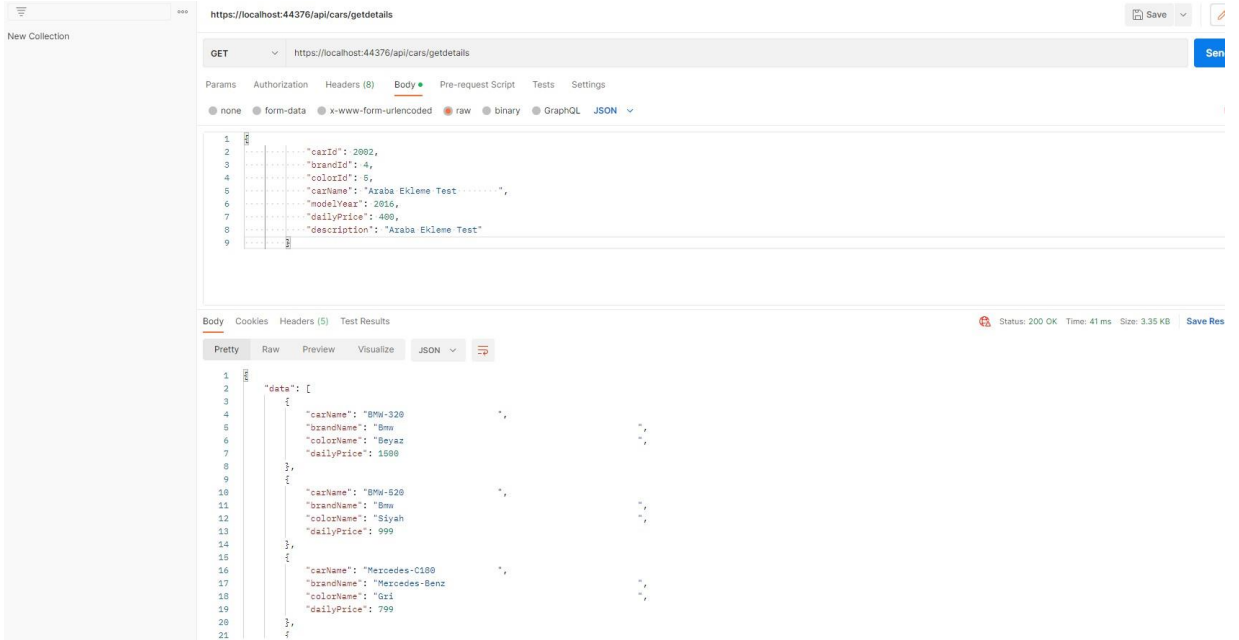
Şekil 19:Araba Ekleme



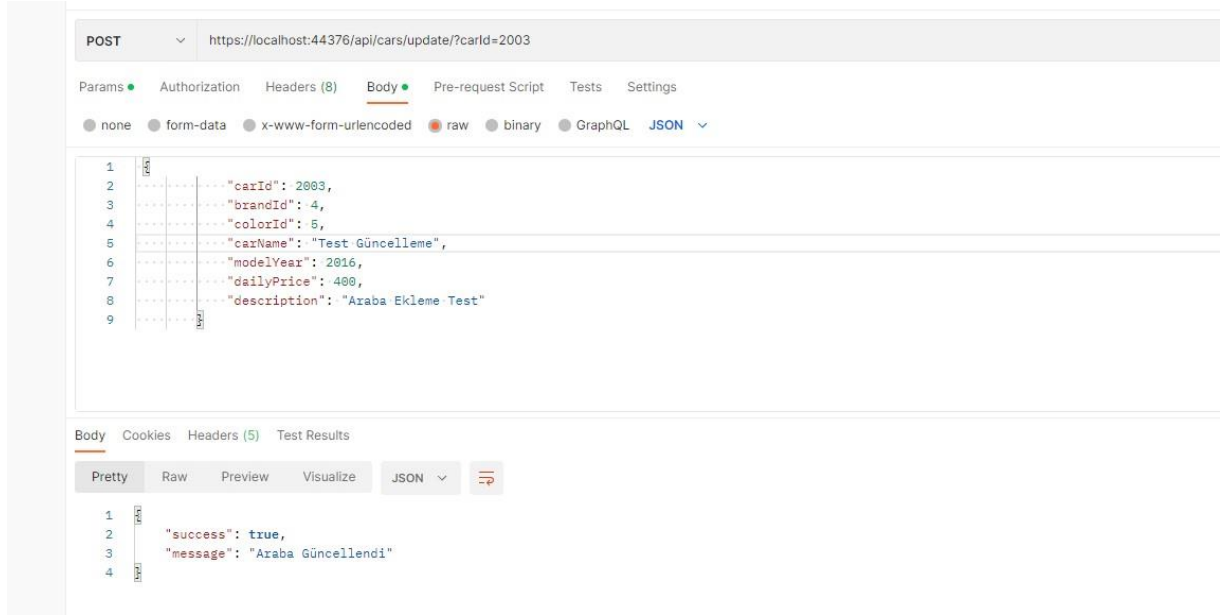
Şekil 20:Araba Eklendikten Sonra Güncel Araba Listesi



Şekil 21:Araba Silme



Şekil 22:Detaylı Araba Listeleme



Şekil 23:Araba Bilgileri Güncelleme

Projeyi Daha Detaylı İncelemek İçin Github Linki:



<https://github.com/kutaytistik/RentACarProject-Backend>

KAYNAKÇA:

- 1-<https://arstechnica.com/information-technology/2015/11/visual-studio-now-supports-debugging-linux-apps-code-editor-now-open-source/>
- 2-https://tr.wikipedia.org/wiki/Microsoft_Visual_Studio
- 3-<https://ieeexplore.ieee.org/document/7395166>
- 4-<https://www.plesk.com/blog/various/mysql-vs-mssql/>
- 5-https://en.wikipedia.org/wiki/Visual_Paradigm
- 6-<https://learn.microsoft.com/en-us/sql/relational-databases/tables/create-tables-database-engine?view=sql-server-ver16>
- 7-https://www.w3schools.com/sql/sql_create_table.asp
- 8-https://tr.wikipedia.org/wiki/Microsoft_SQL_Server
- 9-<https://autofac.readthedocs.io/en/latest/>
- 10-<https://docs.fluentvalidation.net/en/latest/>
- 11-<https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-7.0>
- 12-<https://learn.microsoft.com/en-us/dotnet/csharp/>
- 13-<https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver16>
- 14-<https://www.microsoft.com/tr-tr/sql-server/sql-server-2019>