T.C

ATATÜRK ÜNİVERSİTESİ

İKTİSADİ VE İDARİ BİLİMLER FAKÜLTESİ

YÖNETİM BİLİŞİM SİSTEMLERİ

4. SINIF ÖRGÜN EĞİTİM

YBS BİTİRME PROJESİ

Asp.NET Core 6 + Angular ile E-Ticaret

Ümeyir ATASOY / 200307010

Danışman: Prof.Dr. Uğur YAVUZ

İçindekiler

[**Backend Yazılım Mimarisi Onion Architecture** 5](#_Toc167895092)

[**Temel Kavramlar ve Katmanlar** 5](#_Toc167895093)

[**Domain Katmanı (Çekirdek)** 5](#_Toc167895094)

[**Application Katmanı** 5](#_Toc167895095)

[**Infrastructure Katmanı** 5](#_Toc167895096)

[**Presentation Katmanı** 5](#_Toc167895097)

[**Bağımlılık ve Avantajlar** 6](#_Toc167895098)

[**FrontEnd Angular Mimarisi** 8](#_Toc167895099)

[**Bileşenler (Components)** 8](#_Toc167895100)

[**Modüller (Modules)** 8](#_Toc167895101)

[**Servisler (Services)** 8](#_Toc167895102)

[**Yönlendirme (Routing)** 9](#_Toc167895103)

[**Diğer Yardımcı Öğeler** 9](#_Toc167895104)

[**Angular Projesi Yapısı** 10](#_Toc167895105)

[**Backend yazılımı için kullanılan teknolojiler** 11](#_Toc167895106)

[**Entity Framework (EF)** 11](#_Toc167895107)

[**Entity Framework'ün Temel Kavramları** 11](#_Toc167895108)

[**Entity Framework'ün Avantajları** 11](#_Toc167895109)

[**CORS Politikası ve Uygulaması** 13](#_Toc167895110)

[**CORS Nedir?** 13](#_Toc167895111)

[**CORS'un Çalışma Prensibi** 13](#_Toc167895112)

[**.NET Core'da CORS Uygulaması** 13](#_Toc167895113)

[**CORS Politikası Parametreleri** 13](#_Toc167895114)

[**Fluent Validation** 15](#_Toc167895115)

[**Fluent Validation'in Özellikleri ve Avantajları** 15](#_Toc167895116)

[**Pagination Yapısı** 16](#_Toc167895117)

[**Dosya isim Seo Optimizasyonu** 18](#_Toc167895118)

[**Table Per Hierarhy** 19](#_Toc167895119)

[**Table Per Hierarchy'nin Özellikleri** 19](#_Toc167895120)

[**.NET Entity Framework ve TPH** 19](#_Toc167895121)

[**Dosya Yönetimi İçin Storage Altyapısı // azure-local-aws** 21](#_Toc167895122)

[**MediatR - MediatR.DependencyInjection** 24](#_Toc167895123)

[**MediatR'in Temel İşlevleri** 25](#_Toc167895124)

[**Identity Mekanizması** 29](#_Toc167895125)

[**Identity Mekanizması'nın temel bileşenleri** 30](#_Toc167895126)

[**JWT Teknolojisi** 31](#_Toc167895127)

[**JWT'ler üç bölümden oluşur** 31](#_Toc167895128)

[**Guard Yapılandırılması** 32](#_Toc167895129)

[**Login İşlemleri** 34](#_Toc167895130)

[**Google Login** 34](#_Toc167895131)

[**Facebook Login** 35](#_Toc167895132)

[**Normal Login** 35](#_Toc167895133)

[**Serilog Sinks PostgreSQL** 36](#_Toc167895134)

[**Serilog’u Seq ile Günlük Görselleştirme** 38](#_Toc167895135)

[**Seq'in temel özelliklerinden bazıları** 38](#_Toc167895136)

[**SignalR** 39](#_Toc167895137)

[**SignalR Nasıl Çalışır?** 39](#_Toc167895138)

[**SignalR'ın temel özellikleri şunlardır** 39](#_Toc167895139)

[**Mail Servisi** 41](#_Toc167895140)

[**Rol Yapılandırılması Mimari** 42](#_Toc167895141)

[**QR Coder** 44](#_Toc167895142)

[**Backend için Generics Repositories yapılandırılması** 45](#_Toc167895143)

[**Generics Kullanmanın Avantajları** 45](#_Toc167895144)

[**Frontend yazılımı için kullanılan teknolojiler** 50](#_Toc167895145)

[**Angular Moduler Yapı** 50](#_Toc167895146)

[**Bootstrap(UI) ve Angular Material(Admin) – Dizayn/Tasarım** 51](#_Toc167895147)

[**Bootstrap(UI)** 51](#_Toc167895148)

[**Angular Material(Admin)** 51](#_Toc167895149)

[**Angular Toastr(UI) ve Angular Alertify(Admin) -- Bildiri** 52](#_Toc167895150)

[**Angular Toastr(UI)** 52](#_Toc167895151)

[**Angular Alertify(Admin)** 53](#_Toc167895152)

[**Spinner - Loading** 53](#_Toc167895153)

[**Openbase – Dosya Yükleme** 54](#_Toc167895154)

[**SignalR - Angular** 56](#_Toc167895155)

[**Dynamic Component Loading** 57](#_Toc167895156)

[**Dynamic Component Loading Nedir?** 57](#_Toc167895157)

[**Dynamic Component Loading'in Avantajları** 57](#_Toc167895158)

[**Delete Directive** 59](#_Toc167895159)

[**Angular Directive Nedir?** 59](#_Toc167895160)

[**Base Yapılar** 62](#_Toc167895161)

[**Spinner Base Yapısı** 62](#_Toc167895162)

[**Alertify Base Yapısı** 63](#_Toc167895163)

[**Toastr Base Yapısı** 64](#_Toc167895164)

[**Service Yapılar** 66](#_Toc167895165)

[**HttpClientService** 66](#_Toc167895166)

[**Dialog Penceresi** 67](#_Toc167895167)

[**KAYNAKÇA** 69](#_Toc167895168)

[**Teknolojilerin Kaynakçaları** 69](#_Toc167895169)

[**Backend** 69](#_Toc167895170)

[**Frontend** 69](#_Toc167895171)

# **Backend Yazılım Mimarisi Onion Architecture**

Onion Architecture, yazılım geliştirme süreçlerinde sürdürülebilirliği ve esnekliği artırmak amacıyla oluşturulmuş bir yazılım mimarisi desenidir. Jeffrey Palermo tarafından tanıtılan bu mimari, katmanlı yapısıyla bağımlılıkları yönetir ve iş mantığı ile altyapı kodlarını birbirinden ayırır. Bu yapı, uygulamaların daha kolay test edilmesini, bakımının yapılmasını ve genişletilmesini sağlar.

## **Temel Kavramlar ve Katmanlar**

Onion Architecture'da uygulama, iç içe geçmiş halkalar şeklinde yapılandırılır. En içteki halkadan başlayarak en dıştaki halkaya kadar her katman belirli bir işlevi yerine getirir ve bağımlılık ilişkisi yalnızca bir sonraki iç katmana doğru olur. İşte bu katmanların detaylı açıklamaları:

### **Domain Katmanı (Çekirdek)**

Tanım: Bu katman, uygulamanın iş kurallarını ve mantığını içerir. En içte yer alır ve bağımsızdır; yani, diğer katmanlara bağımlı değildir.

Bileşenler:

Entity (Varlıklar): Uygulamanın ana iş nesneleri.

Value Objects (Değer Nesneleri): Entity’lerin özelliklerini temsil eden nesneler.

Domain Services: İş mantığını içeren ve birden fazla Entity'yi kullanarak çalışan servisler.

### **Application Katmanı**

Tanım: Bu katman, uygulama mantığını yönetir ve domain katmanı ile dış dünya arasında köprü görevi görür. Kullanıcı isteklerinin nasıl işlendiği burada tanımlanır.

Bileşenler:

DTO (Data Transfer Objects): Veri transferi için kullanılan nesneler.

Application Services: Domain nesneleri üzerinde işlem yapan servisler.

Interfaces: Diğer katmanlarla iletişimi sağlamak için kullanılan arabirimler.

### **Infrastructure Katmanı**

Tanım: Altyapı hizmetleri ve teknik detayları içerir. Veritabanı, dosya sistemi, dış servisler gibi altyapı bağımlılıkları burada yer alır.

Bileşenler:

Repositories: Veritabanı işlemlerini gerçekleştiren sınıflar.

External Services: Diğer sistemler veya servislerle iletişimi sağlayan bileşenler.

Configuration: Uygulamanın yapılandırma ayarları.

### **Presentation Katmanı**

Tanım: Kullanıcı arayüzünü temsil eden bu katman, kullanıcıdan gelen istekleri Application katmanına yönlendirir ve sonuçları kullanıcıya sunar.

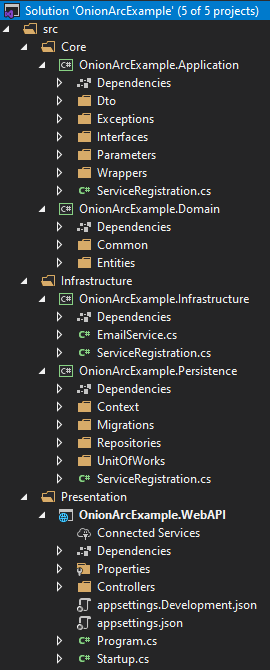
Bileşenler:

Controllers/Views (MVC modelinde): Kullanıcı isteklerini işleyen ve sonuçları gösteren bileşenler.

API Endpoints: RESTful API'lerde, kullanıcı isteklerini karşılayan uç noktalar.

UI Components: Kullanıcı arayüz bileşenleri (Angular, React gibi frontend frameworkleri kullanılarak).

Örnek bir Onion Arc. MimariTasarımı



### **Bağımlılık ve Avantajlar**

Onion Architecture, bağımlılıkların dış katmanlardan iç katmanlara doğru olmasını sağlar. Bu, iş mantığının altyapıdan bağımsız olmasını ve kolayca test edilebilmesini sağlar. İşte bu mimarinin bazı avantajları:

Test Edilebilirlik: İş kuralları ve mantığı, altyapıdan ayrıldığı için kolayca birim testlere tabi tutulabilir.

Bakım Kolaylığı: Katmanlar arasındaki bağımlılıkların yönetilmesi daha kolaydır, bu da bakım süreçlerini basitleştirir.

Esneklik: Yeni teknolojiler veya bileşenler eklenebilir ya da değiştirilebilir, çünkü bağımlılıklar minimize edilmiştir.

Sürdürülebilirlik: Uzun vadeli projelerde sürdürülebilir bir yapı sağlar ve gelişen ihtiyaçlara uyum sağlar.

Örnek Uygulama:

Örneğin, bir e-ticaret uygulaması oluştururken Onion Architecture kullanırsak, işte nasıl yapılandırılacağına dair bir örnek:

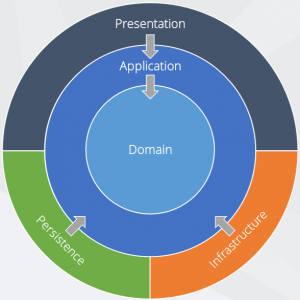
Domain Katmanı: Ürün, Sipariş, Kullanıcı gibi varlıklar ve bunların iş kuralları (örneğin, bir siparişin geçerliliğini kontrol eden servisler).

Application Katmanı: Sipariş oluşturma, ürün ekleme gibi işlemleri gerçekleştiren servisler ve DTO'lar.

Infrastructure Katmanı: Veritabanı erişim sınıfları (EF Core, Dapper gibi), dosya sistemi işlemleri, ödeme servisleri entegrasyonu.

Presentation Katmanı: ASP.NET Core MVC veya Angular tabanlı kullanıcı arayüzü, API uç noktaları.

Bu yapı sayesinde her katman kendi sorumluluğunu üstlenir ve birbirlerinden bağımsız olarak gelişir, bu da uygulamanın sürdürülebilirliğini ve esnekliğini artırır.



# **FrontEnd Angular Mimarisi**

Angular mimarisi, birkaç temel yapı taşı üzerine inşa edilmiştir: bileşenler (components), modüller (modules), servisler (services), yönlendirme (routing) ve diğer yardımcı öğeler. Bu yapı taşlarının her biri belirli bir işlevi yerine getirir ve birlikte çalışarak güçlü ve esnek bir uygulama geliştirme ortamı sunar.

## **Bileşenler (Components)**

Tanım: Angular uygulamalarının temel yapı taşıdır. Her bileşen, kullanıcı arayüzünün belirli bir bölümünü temsil eder.

Yapı: Bir bileşen, HTML şablonu (template), CSS stilleri ve TypeScript sınıfından (class) oluşur.

Özellikler:

Şablonlar (Templates): HTML ve Angular'ın özel sözdizimlerini kullanarak bileşenin kullanıcı arayüzünü tanımlar.

Veri Bağlama (Data Binding): Şablon ve bileşen sınıfı arasında veri alışverişi yapmayı sağlar. Angular'da dört tür veri bağlama vardır: interpolation, property binding, event binding ve two-way data binding.

Bileşen Yaşam Döngüsü (Component Lifecycle): Angular, bileşenlerin belirli olaylar (ngOnInit, ngOnChanges, ngOnDestroy gibi) sırasında belirli işlevleri yerine getirmesine olanak tanıyan yaşam döngüsü kancaları sağlar.

## **Modüller (Modules)**

Tanım: Modüller, uygulamanın farklı bölümlerini organize eden ve bir araya getiren yapılardır. Angular'da her uygulama en az bir ana modüle (root module) sahiptir.

Yapı: Modüller, diğer modülleri, bileşenleri, direktifleri (directives), boruları (pipes) ve servisleri bir araya getirir.

Özellikler:

NgModule Dekoratörü: Modüller, @NgModule dekoratörü ile tanımlanır ve bu dekoratör, modülün bileşenleri, direktifleri, boruları ve diğer modülleri hakkında bilgi sağlar.

Root Module: Angular uygulamasının başlangıç noktasıdır ve genellikle AppModule olarak adlandırılır.

Feature Modules: Uygulamanın belirli işlevlerini kapsüllemek için kullanılan modüllerdir. Örneğin, kullanıcı yönetimi veya sipariş yönetimi gibi.

## **Servisler (Services)**

Tanım: Uygulama genelinde kullanılacak iş mantığını ve veri işlemlerini kapsüller. Genellikle bileşenler arasında veri paylaşımı veya API çağrıları için kullanılır.

Yapı: Servisler, Angular'ın dependency injection (bağımlılık enjeksiyonu) sistemi ile yönetilir ve @Injectable dekoratörü ile tanımlanır.

Özellikler:

Dependency Injection: Angular, servislerin ve diğer bağımlılıkların yönetimini kolaylaştırmak için dependency injection kullanır. Bu, bileşenlerin ve servislerin kolayca test edilebilmesini sağlar.

## **Yönlendirme (Routing)**

Tanım: Uygulama içindeki farklı bileşenler arasında gezinmeyi sağlayan sistemdir. Kullanıcıların URL tabanlı navigasyon yapmasına olanak tanır.

Yapı: Angular Router, yönlendirme modülü (RouterModule) tarafından yönetilir ve yönlendirme yapılandırmaları (route configurations) ile belirlenir.

Özellikler:

Route Configuration: Yönlendirme yollarını ve bunlara karşılık gelen bileşenleri tanımlar.

Router Outlet: Şablonda, yönlendirme işlemlerinin hangi bileşen tarafından doldurulacağını belirten yer tutucudur.

Guards: Kullanıcıların belirli rotalara erişimini kontrol eden ve yönlendirme kararlarını dinamik olarak değiştiren yapılar.

## **Diğer Yardımcı Öğeler**

Direktifler (Directives):

Tanım: HTML öğelerine ek davranışlar kazandırır.

Türler: Yapısal direktifler (ngIf, ngFor) ve öznitelik direktifleri (ngClass, ngStyle).

Borular (Pipes):

Tanım: Şablonlarda veriyi formatlamak için kullanılır. Örneğin, tarih formatlama, büyük/küçük harf dönüşümleri.

Özellikler: Angular, birçok yerleşik boru sunar ve geliştiriciler kendi özel borularını da oluşturabilirler.

Formlar:

Tanım: Kullanıcıdan veri toplamak için kullanılan araçlar.

Türler: Reactive Forms ve Template-driven Forms.

Özellikler: Doğrulama, hata yönetimi ve veri bağlama.

## **Angular Projesi Yapısı**

Tipik bir Angular projesi, aşağıdaki ana klasör ve dosya yapısına sahiptir:

src/app: Uygulamanın ana kaynak dosyalarının bulunduğu klasör.

app.module.ts: Uygulamanın ana modülü.

app.component.ts: Uygulamanın ana bileşeni.

app.component.html: Ana bileşenin şablonu.

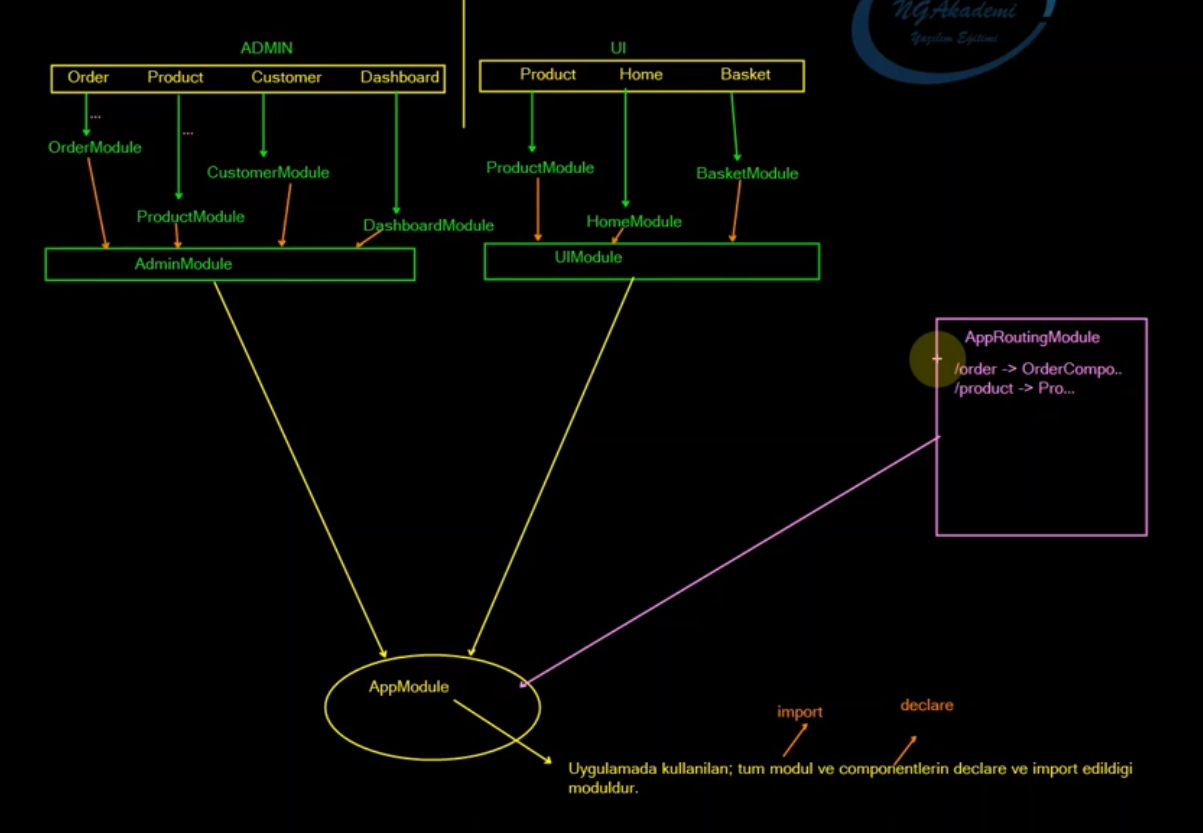
app.component.css: Ana bileşenin stilleri.

assets: Statik varlıklar (resimler, JSON dosyaları, vb.)

environments: Farklı ortamlar (development, production) için yapılandırma dosyaları.

angular.json: Proje yapılandırma dosyası.

package.json: Projenin bağımlılıklarını listeleyen dosya.



# **Backend yazılımı için kullanılan teknolojiler**

## **Entity Framework (EF)**

Entity Framework (EF), .NET uygulamaları için geliştirilen bir nesne-ilişkisel eşleme (Object-Relational Mapping, ORM) aracıdır. Microsoft tarafından geliştirilmiş olan Entity Framework, veritabanları ile çalışmayı kolaylaştırır ve veritabanı işlemlerini daha verimli hale getirir. ORM aracı olarak, veritabanı tablolarını ve kayıtlarını, .NET sınıfları ve nesneleri ile temsil eder. Bu sayede geliştiriciler, SQL sorguları yazmadan veritabanı işlemlerini gerçekleştirebilirler.

### **Entity Framework'ün Temel Kavramları**

**Model**

Entity (Varlık): Veritabanındaki tabloları temsil eden sınıflardır. Her sınıf, bir tabloya karşılık gelir ve sınıfın özellikleri (properties), tablodaki sütunları temsil eder.

DbContext: Veritabanı ile etkileşimde bulunmak için kullanılan temel sınıftır. Veritabanı bağlantısı kurar ve CRUD (Create, Read, Update, Delete) işlemlerini gerçekleştirir.

#### **Code First, Database First, ve Model First Yaklaşımları**

Code First: Veritabanı şemasını, C# sınıfları yazarak oluşturma yaklaşımıdır. Geliştiriciler, sınıfları tanımlar ve EF bu tanımları kullanarak veritabanını oluşturur.

Database First: Mevcut bir veritabanını kullanarak C# sınıflarını oluşturma yaklaşımıdır. EF, veritabanı şemasını okur ve buna uygun sınıfları otomatik olarak üretir.

Model First: Görsel bir model kullanarak veritabanı şemasını tasarlama ve ardından bu modeli kullanarak veritabanını ve sınıfları oluşturma yaklaşımıdır.

#### **LINQ (Language Integrated Query)**

EF, veritabanı sorguları için LINQ kullanır. LINQ, SQL sorguları yazmak yerine C# dilinde sorgular yazmayı sağlar. Bu sorgular, EF tarafından SQL'e çevrilir ve veritabanına gönderilir.

#### **Migrations**

Veritabanı şemasındaki değişiklikleri yönetmek için kullanılan araçtır. Migrations, veritabanı şemasını güncellemek için gerekli SQL komutlarını otomatik olarak oluşturur ve uygular.

### **Entity Framework'ün Avantajları**

Kolaylık ve Hız: SQL sorguları yazmadan veritabanı işlemlerini gerçekleştirmek, geliştiricilerin işini kolaylaştırır ve hızlandırır.

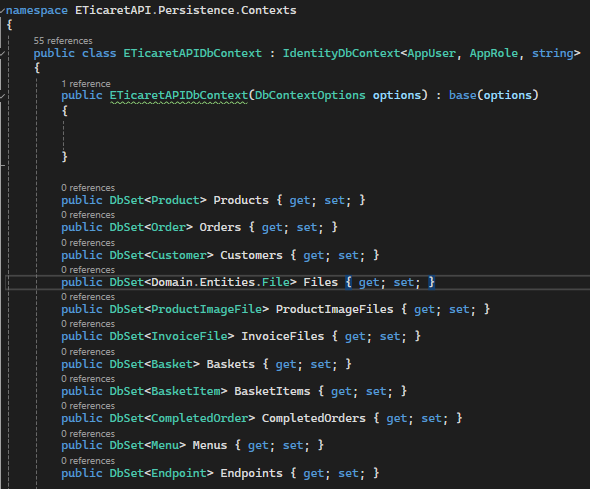
Veritabanı Bağımsızlığı: EF, farklı veritabanı sağlayıcılarını destekler. Aynı kodla farklı veritabanları (SQL Server, MySQL, PostgreSQL vb.) ile çalışabilirsiniz.

Bakım Kolaylığı: Kod tabanlı modelleme, veritabanı şemasını ve veritabanı işlemlerini daha okunabilir ve yönetilebilir hale getirir.

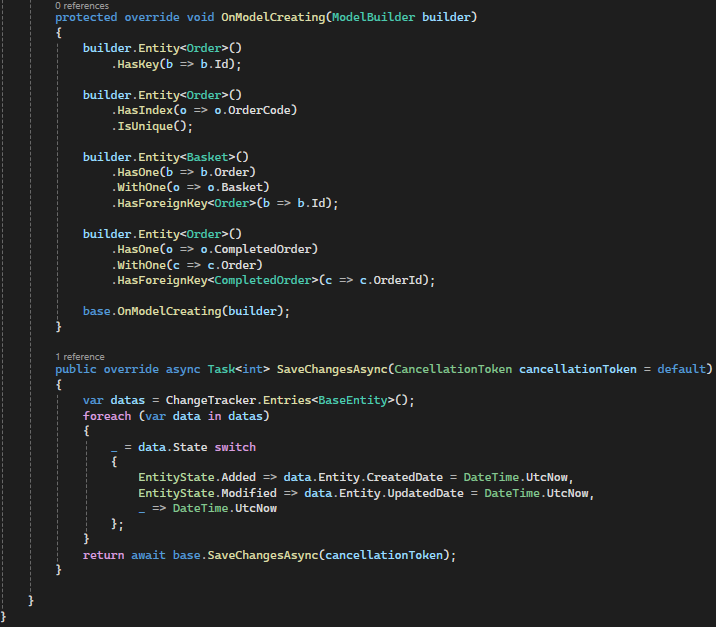
Güçlü Tip Güvenliği: EF, C# tipi güvenliğini kullanarak daha az hata ile daha güvenli kod yazmanızı sağlar.

Zengin Özellik Seti: LINQ, Migrations, Lazy Loading, Eager Loading, ve Query Caching gibi zengin özellikler sunar.

EF kullanarak Code first ile migration basılarak database oluşumu



Üründe bir güncelleme yapılacaksa o zaman updatedate güncellenmesi için gereken kod oluşumu



## **CORS Politikası ve Uygulaması**

### **CORS Nedir?**

CORS, web uygulamalarının kendi kaynaklarını (veritabanı, API, vb.) farklı orijinlerden gelen isteklere açmasına olanak tanır. Bu, güvenliği tehlikeye atmadan farklı orijinler arasında veri paylaşımını sağlar. CORS, sunucu tarafında yapılandırılır ve tarayıcı, bu politikaları uygulayarak istemci tarafında güvenliği sağlar.

### **CORS'un Çalışma Prensibi**

Preflight Request (Ön Uçuş İsteği): Bazı durumlarda tarayıcı, asıl isteği yapmadan önce bir "OPTIONS" isteği gönderir. Bu istek, sunucudan izin isteyerek, tarayıcının isteği yapıp yapamayacağını kontrol eder.

Simple Request (Basit İstek): GET, POST ve HEAD gibi basit HTTP metotları kullanılırsa ve belirli koşullar sağlanırsa, tarayıcı doğrudan isteği gönderir.

Headers: Sunucu, CORS isteklerine yanıt verirken bazı özel başlıklar (headers) ekler. Bu başlıklar, hangi orijinlerin erişimine izin verildiğini, hangi HTTP metotlarının kullanılabileceğini ve hangi başlıkların paylaşılabileceğini belirler.

### **.NET Core'da CORS Uygulaması**

ASP.NET Core'da CORS politikalarını yapılandırmak oldukça basittir. Aşağıdaki adımlar, temel bir CORS yapılandırması için gereklidir.

Program.cs de işlemler yapılır.



Bu kod, AddCors metoduyla CORS hizmetini uygulamaya ekler. Ardından AddDefaultPolicy metoduyla varsayılan bir politika eklenir. Bu politika, belirtilen kökenlerden gelen isteklere izin verir (http://localhost:4200 ve https://localhost:4200). .AllowAnyHeader() ve .AllowAnyMethod() metotlarıyla herhangi bir başlık veya HTTP metodu kabul edilir. .AllowCredentials() metodu, kimlik bilgisinin paylaşılmasına izin verir. Bu kod, genellikle geliştirme aşamasında yerel sunucu ile istemci arasında CORS engellerini kaldırmak için kullanılır.

//Yukarıdaki kodda angular portuna izin vermektedir sadece

### **CORS Politikası Parametreleri**

CORS politikaları yapılandırılırken kullanılabilecek bazı yaygın parametreler şunlardır:

WithOrigins: Hangi orijinlerin (domainden gelen isteklerin) izinli olduğunu belirtir.

AllowAnyOrigin: Herhangi bir orijine izin verir. Güvenlik açısından dikkatli kullanılmalıdır.

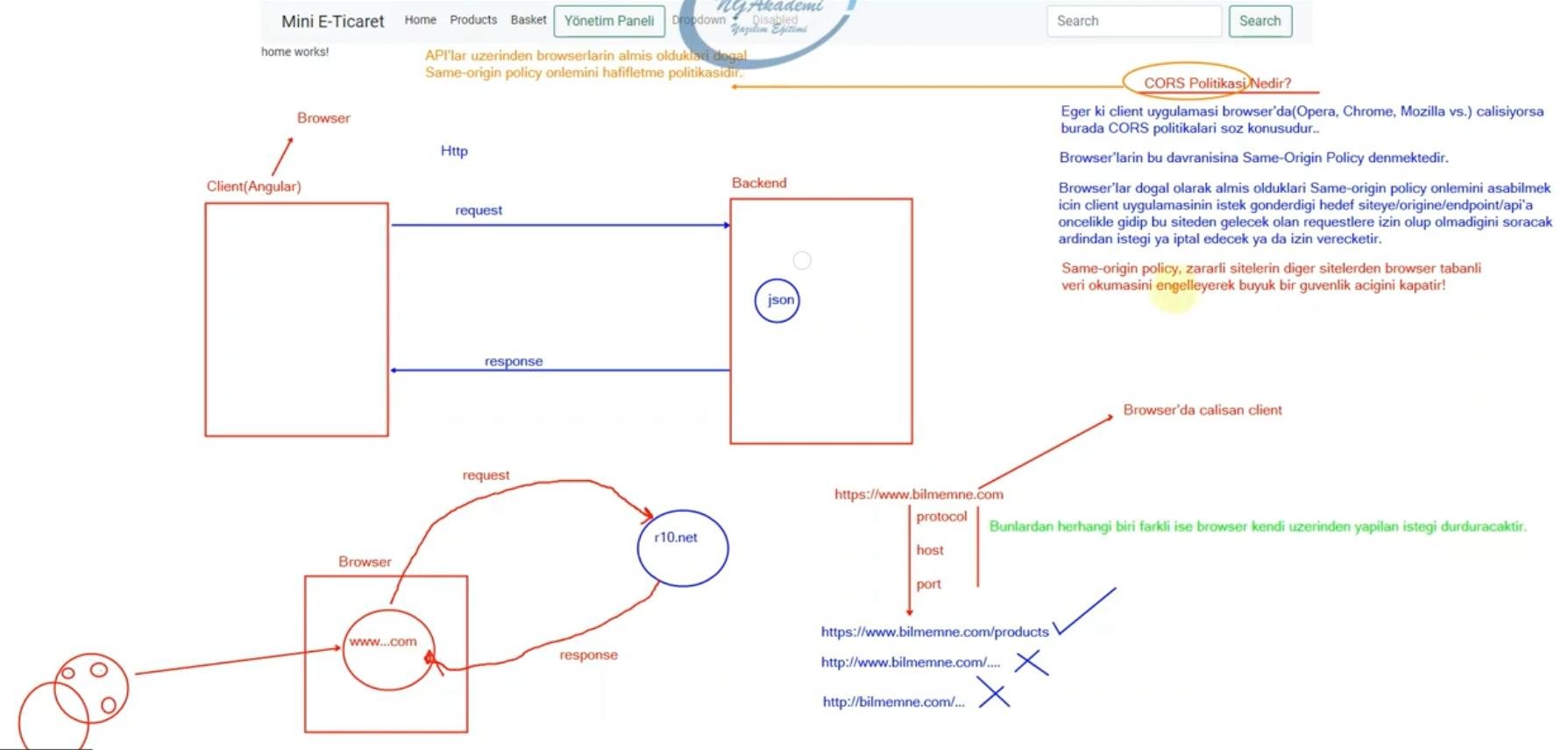
AllowAnyMethod: Herhangi bir HTTP metoduna (GET, POST, PUT, DELETE, vb.) izin verir.

AllowAnyHeader: Herhangi bir başlığa izin verir.

WithMethods: İzin verilen HTTP metotlarını belirtir

WithHeaders: İzin verilen başlıkları belirtir.

AllowCredentials: Kimlik bilgilerini (çerezler, HTTP kimlik doğrulama) gönderen isteklere izin verir.



## **Fluent Validation**

Fluent Validation, .NET platformu için geliştirilmiş bir doğrulama kütüphanesidir. Bu kütüphane, .NET uygulamalarında giriş verilerinin doğrulanmasını kolaylaştırmak için kullanılır. Özellikle, MVC, Web API ve diğer .NET projelerinde kullanılan giriş verilerinin (form alanları, API istekleri vb.) doğrulanması için yaygın olarak tercih edilir.

### **Fluent Validation'in Özellikleri ve Avantajları**

Zengin Doğrulama Kuralları: Fluent Validation, çeşitli doğrulama kurallarını destekler ve bunların uygulanması için zengin bir API sağlar. Basit veri türlerinden nesnelere kadar geniş bir yelpazede doğrulama kuralları tanımlanabilir.

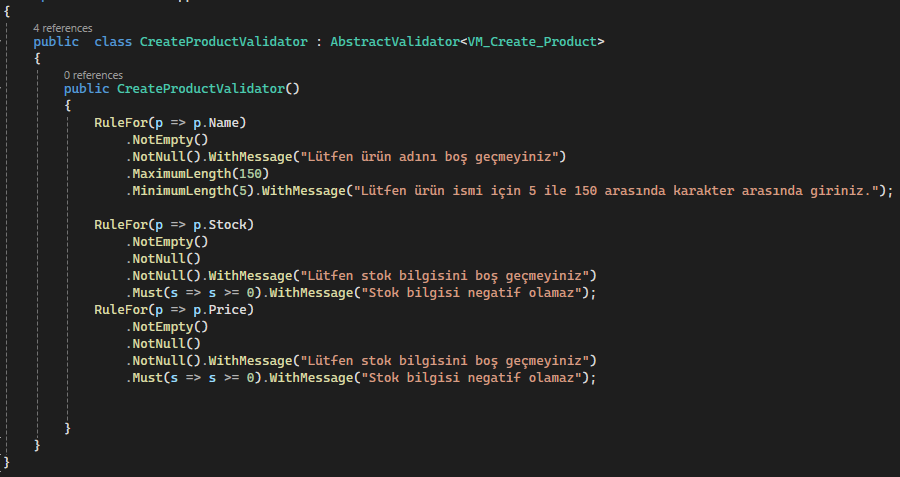
Okunabilir ve Esnek Sözdizimi: Fluent Validation, doğrulama kurallarını tanımlamak için akıcı (fluent) bir sözdizimi kullanır. Bu, doğrulama kurallarının okunabilirliğini artırır ve kuralların kolayca anlaşılmasını sağlar.

Modüler ve Genişletilebilir: Fluent Validation, doğrulama kurallarını ayrı sınıflar halinde organize etmeyi sağlar. Bu, doğrulama mantığını modüler hale getirir ve kod tekrarını azaltır. Ayrıca, özel doğrulama kuralları oluşturmak ve mevcut doğrulama kurallarını genişletmek de mümkündür.

Integrasyon Kolaylığı: Fluent Validation, .NET platformundaki birçok farklı bileşenle kolayca entegre edilebilir. Özellikle, ASP.NET Core MVC, Web API ve diğer .NET projeleriyle sorunsuz bir şekilde çalışır.

Hata Mesajları Özelleştirme: Kütüphane, doğrulama hatası durumunda üretilen hata mesajlarını kolayca özelleştirmenize olanak tanır. Bu, kullanıcıya daha anlamlı ve açıklayıcı hata mesajları sunmanızı sağlar.

Ürün eklenirken yapılması gerekenler için bir yapılmış bir Validator



## **Pagination Yapısı**

Sayfalama (Pagination), web uygulamalarında büyük veri kümesini kullanıcı dostu bir şekilde yönetmek için kullanılan bir tekniktir. Özellikle, büyük listeler, tablolar veya sonuç kümesi gerektiren sorgular gibi durumlarda kullanılır. Sayfalama, kullanıcının veri kümesinin farklı bölümlerini kolayca görebilmesini ve gezinebilmesini sağlar.

ASP.NET Core MVC veya Web API gibi .NET projelerinde sayfalama yapısı genellikle şu adımlarla uygulanır:

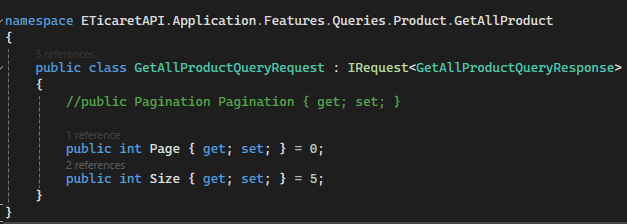
Veri Kaynağından Veri Alma: İlk olarak, veri kaynağından (örneğin, veritabanı) ilgili veri kümesi alınır. Bu veri kümesi, sayfalamaya uygun bir şekilde sıralanabilir ve bölünebilir olmalıdır.

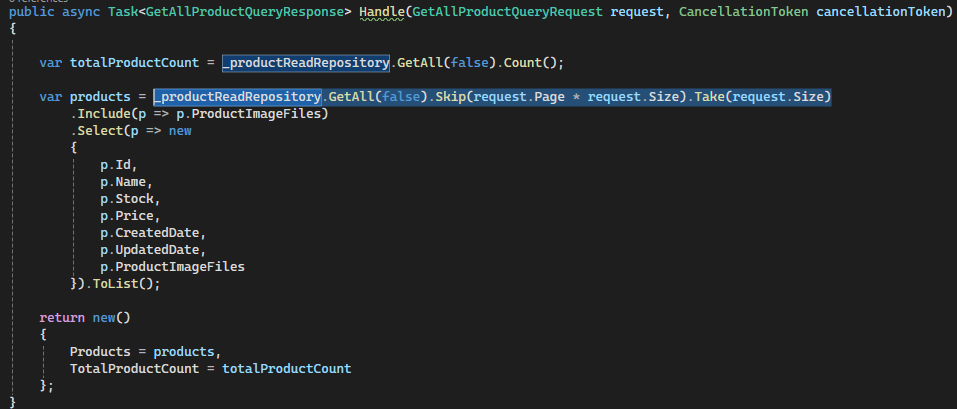
Sayfalama İşleminin Gerçekleştirilmesi: Veri kümesi, sayfalama algoritması kullanılarak belirli bir sayfa boyutuna ve belirli bir sayfa numarasına göre bölünür. Bu, sayfalanmış veri kümesini elde etmek için kullanılır.

Sayfalama Bilgilerinin Kullanıcıya Sunulması: Sayfalanmış veri kümesi, kullanıcı arayüzünde sayfa numaralandırması, sayfa kontrolleri ve sayfa boyutu seçenekleri ile birlikte gösterilir. Kullanıcılar, istedikleri sayfaya gitmek veya sayfa boyutunu değiştirmek için bu kontrolleri kullanabilirler.

Kullanıcı Etkileşimi ile Sayfa Değişikliklerinin Yönetilmesi: Kullanıcılar, sayfa numaralarına tıklayarak veya sayfa boyutunu değiştirerek farklı sayfalara geçebilirler. Bu değişiklikler, kullanıcı tarafından yapıldığında, ilgili sayfanın yeniden yüklenmesi veya AJAX isteği gönderilerek güncellenmesi gerekebilir.

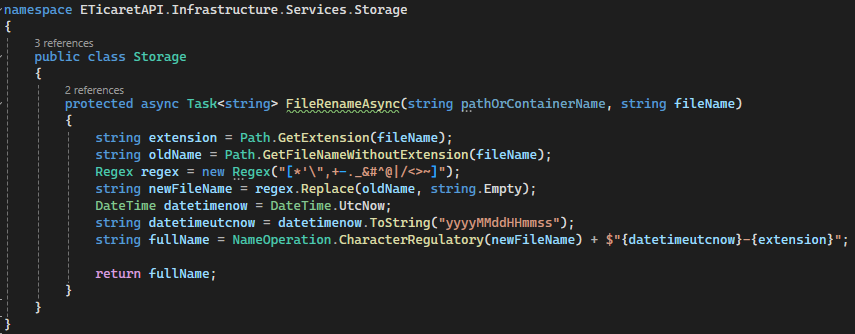
Ürünler için yapılmış bir Pagination query requesti

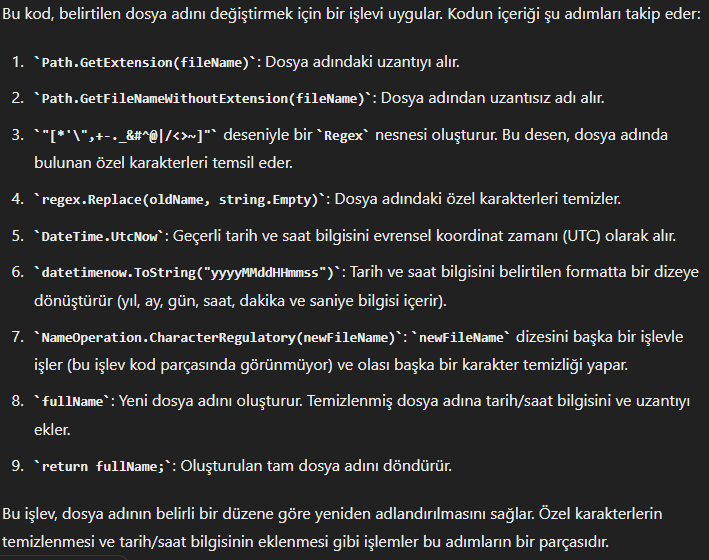




Bu kod, belirli bir sayfa boyutuyla birlikte belirli bir sayfadan başlayarak bir liste almak için kullanılabilir. \_productReadRepository.GetAll(false) ile tüm ürünleri getirirsiniz, Skip() metoduyla belirtilen sayfa numarasının başında bulunan kayıtları atlar ve Take() metoduyla belirli bir sayıda kayıt alırsınız. Bu genellikle bir sayfalama işlemi için kullanılır, böylece büyük veri kümesi olan sonuçlar kullanıcıya küçük parçalar halinde gösterilir.

## **Dosya isim Seo Optimizasyonu**





## **Table Per Hierarhy**

Table Per Hierarchy (TPH), .NET Entity Framework gibi ORM araçlarında ve ilişkisel veritabanlarında nesne mirasının veri tabanı tablolarına nasıl yansıtılacağını belirleyen bir kalıptır. TPH, nesne tabanlı modelleme yaklaşımında sıklıkla kullanılan bir kalıptır ve tek tablo içinde bir hiyerarşiyi temsil etmek için kullanılır.

### **Table Per Hierarchy'nin Özellikleri**

Tek Tablo Kullanımı: TPH kalıbında, bir hiyerarşinin tüm alt sınıfları tek bir veritabanı tablosunda temsil edilir. Bu tablo, tüm alt sınıfların özelliklerini içerir.

Diskriminatör Sütun: TPH, bir diskriminatör sütun kullanır. Bu sütun, tablodaki her bir satırın hangi alt sınıfa ait olduğunu belirtir. Bu sayede farklı alt sınıflar arasındaki farklar tabloda tutulabilir.

Polimorfik İlişkilerin Desteklenmesi: TPH, farklı alt sınıflar arasındaki polimorfik ilişkileri (örneğin, bir üst sınıf referansı aracılığıyla alt sınıflara erişim) destekler.

Performans Avantajı: TPH, tek bir tabloda tüm verilerin saklanmasını sağladığı için performans avantajı sağlar. Tek bir tabloda verilerin saklanması, veritabanı sorgularını daha basitleştirir ve veritabanı performansını artırabilir.

### **.NET Entity Framework ve TPH**

.NET Entity Framework, Table Per Hierarchy kalıbını destekler ve hiyerarşik nesne modellemesini veritabanı tablolarına nasıl yansıtacağınızı belirlemek için kullanışlı bir araçtır. TPH, .NET projelerinde genellikle şu şekilde uygulanır:

Ana Sınıfın Tanımlanması: TPH'de temel sınıf (base class) tanımlanır. Bu sınıf, tüm alt sınıfların paylaştığı ortak özellikleri ve davranışları içerir.

Alt Sınıfların Tanımlanması: TPH'de farklı alt sınıflar, ana sınıfın miras alındığı sınıflar olarak tanımlanır. Her bir alt sınıf, özel özelliklerini ve davranışlarını içerir.

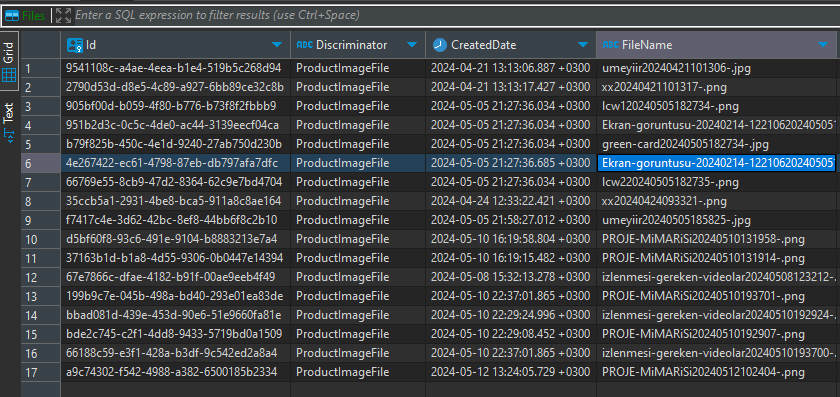
TPH İlişkisinin Belirlenmesi: Entity Framework Fluent API veya veri tabanı kodlama yaklaşımı kullanılarak TPH ilişkisi belirlenir. Bu, Entity Framework'e hangi sınıfların bir TPH hiyerarşisinde olduğunu ve hangi sınıfın hangi tabloya karşılık geldiğini bildirir.

Veritabanı Oluşturma ve Yönetme: Entity Framework Code First yaklaşımı kullanılarak veritabanı şeması oluşturulur. TPH ilişkisi doğru şekilde yapılandırıldığında, Entity Framework otomatik olarak doğru tabloları ve ilişkileri oluşturur.

Örnek:

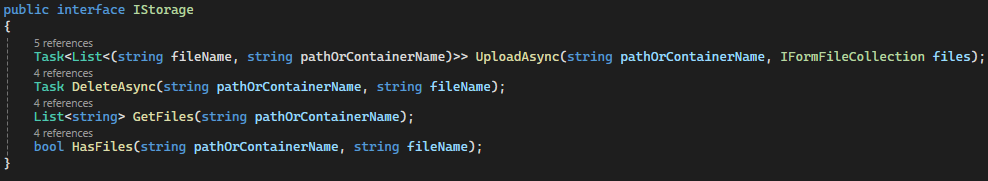
Örnek olarak, bir hayvan türleri hiyerarşisi düşünelim. Bu hiyerarşi köpekler, kediler ve kuşlar gibi farklı alt türleri içerebilir. TPH kullanarak, tüm hayvan türlerini temsil etmek için tek bir Animals tablosu oluşturulabilir. Bu tablo, bir Type sütunu ile her bir hayvan türünün türünü belirtir. Köpekler için veri Type sütununda "Dog" olarak, kediler için "Cat" olarak ve kuşlar için "Bird" olarak saklanabilir.

Files tablosu için yapılmış bit TPH

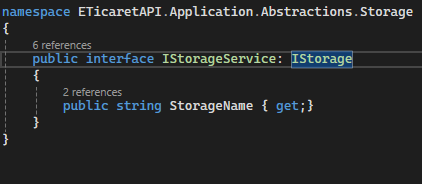


## **Dosya Yönetimi İçin Storage Altyapısı // azure-local-aws**

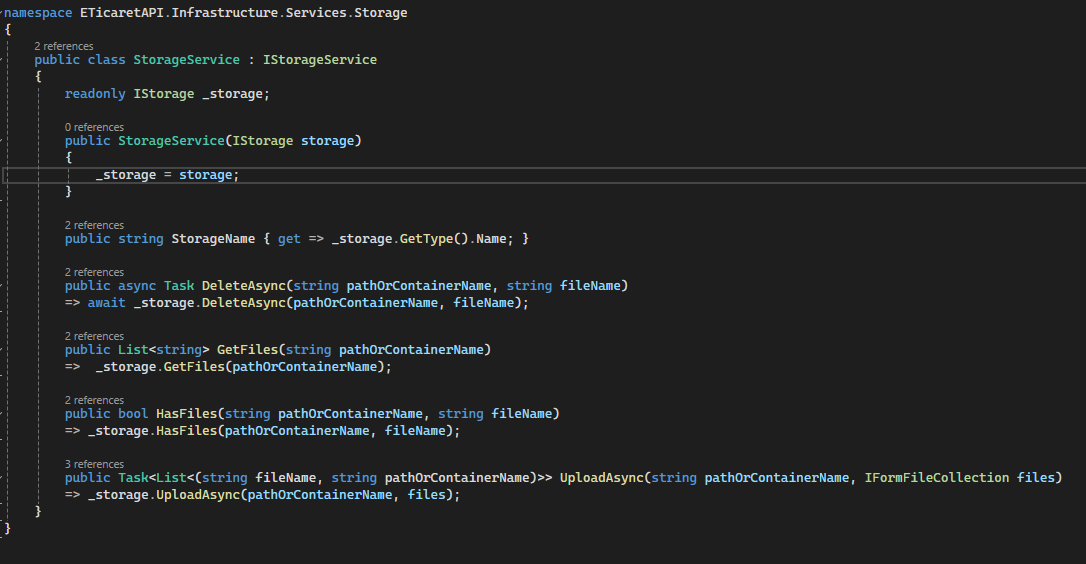
Temel Storage fonksiyonların arayüzleri

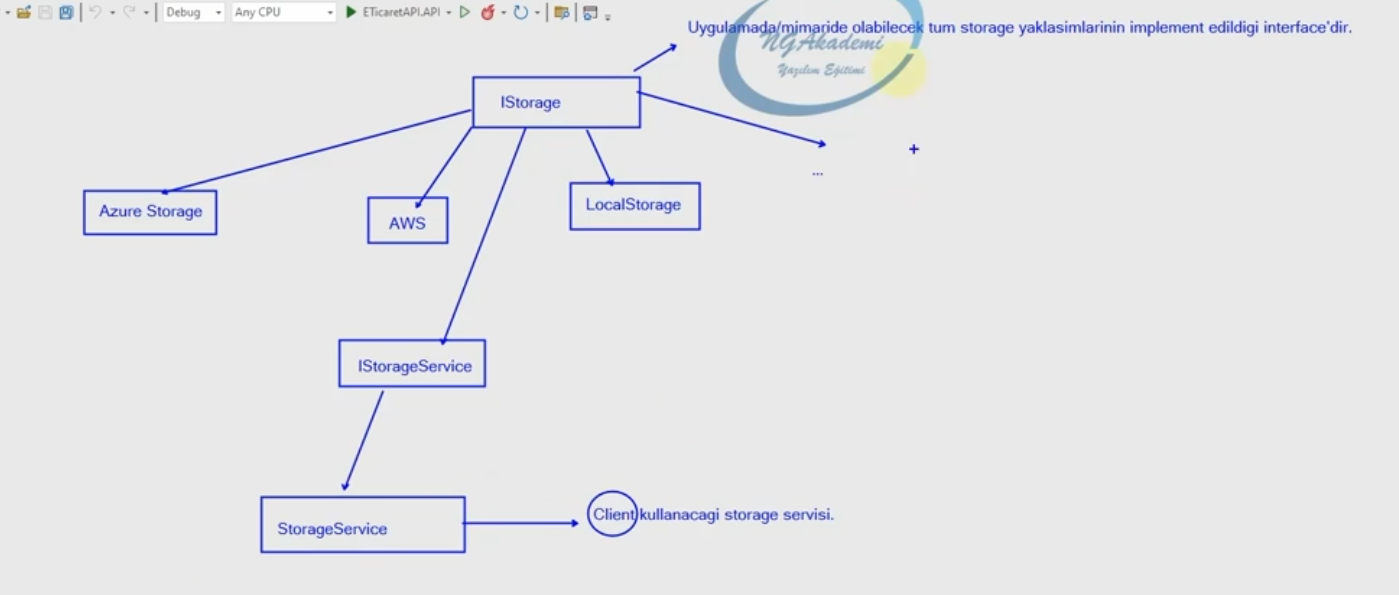


Çoklu depolama kullanımı için name arayüzü oluşturma

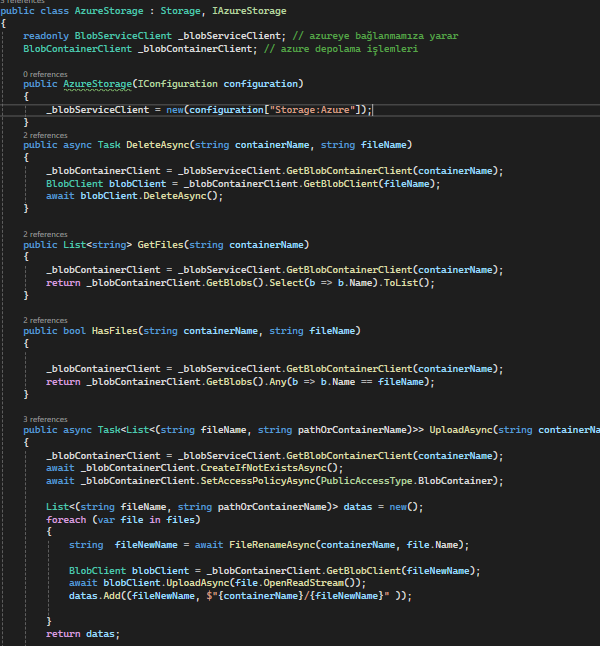


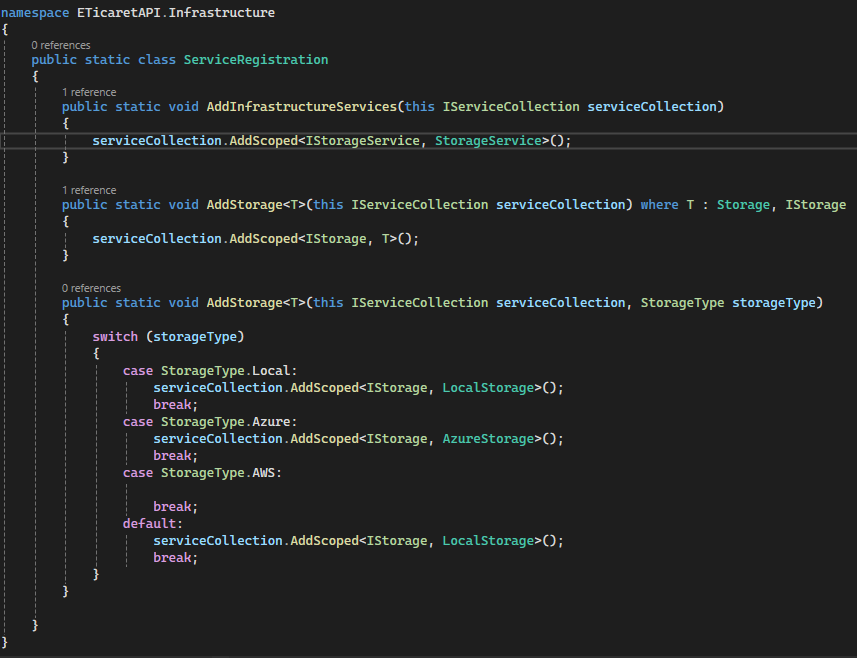
Temel Depolama metodlarının oluşumu





Örnek Azure için yapılmış bir storage kodlaması, Local içinde yapılır AWS içinde yapılır hangisinin kullanacağı bir sonraki adımda gösterilecek.



Burada hazırlanan depolama yapılarının program.cs’e registration yapılması

Hangi storagenin belirtileceği program.cs de bu şekilde belirtiliyor.



## **MediatR - MediatR.DependencyInjection**

MediatR, .NET platformunda sıklıkla kullanılan bir kütüphanedir ve özellikle CQRS (Command Query Responsibility Segregation) tasarım desenini uygularken ve uygulama içindeki olayları işlerken yardımcı olur. Bu kütüphane, istemci kodunu, iş mantığını ve veri erişim kodunu birbirinden ayırmaya yardımcı olur ve birçok .NET uygulamasında temiz bir mimari oluşturmak için kullanılır.

MediatR.DependencyInjection, MediatR kütüphanesinin bağımlılıklarının ve Mediator (Aracı) desenini kullanarak uygulamanın içindeki sınıflar arasındaki iletişimi kolaylaştıran bir pakettir. Bu paket, .NET uygulamalarında kolaylıkla MediatR özelliklerini entegre etmek için kullanılır.

### **MediatR'in Temel İşlevleri**

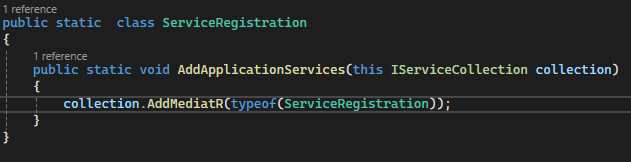
Komut ve Sorgu İşlemleri: MediatR, komut (Command) ve sorgu (Query) işlemlerini kolaylaştırır. Komutlar, uygulamada bir değişiklik yapmayı amaçlayan işlemlerdir (örneğin, bir kayıt ekleme işlemi). Sorgular, uygulamadan veri almayı amaçlayan işlemlerdir.

Aracı Deseni (Mediator Pattern): MediatR, aracı desenini uygular. Bu desen, istemci ve işleyici arasındaki ilişkiyi ayırır ve istemci kodunun karmaşıklığını azaltır. Mediator, istemci tarafından belirli bir işlem başlatıldığında ilgili işleyiciyi çağırır.

İşleyici (Handler) Sınıfları: MediatR, işlemi gerçekleştirecek işleyici sınıflarını temsil eder. Her komut veya sorgu için bir işleyici sınıfı oluşturulur ve bu sınıf, işlemi gerçekleştiren kodu içerir.

Bağımlılık Enjeksiyonu (Dependency Injection): MediatR.DependencyInjection, .NET Core ve .NET 5 gibi platformlarda bağımlılık enjeksiyonunu destekler. Bu, MediatR özelliklerini kullanırken bağımlılıkları yönetmeyi kolaylaştırır.

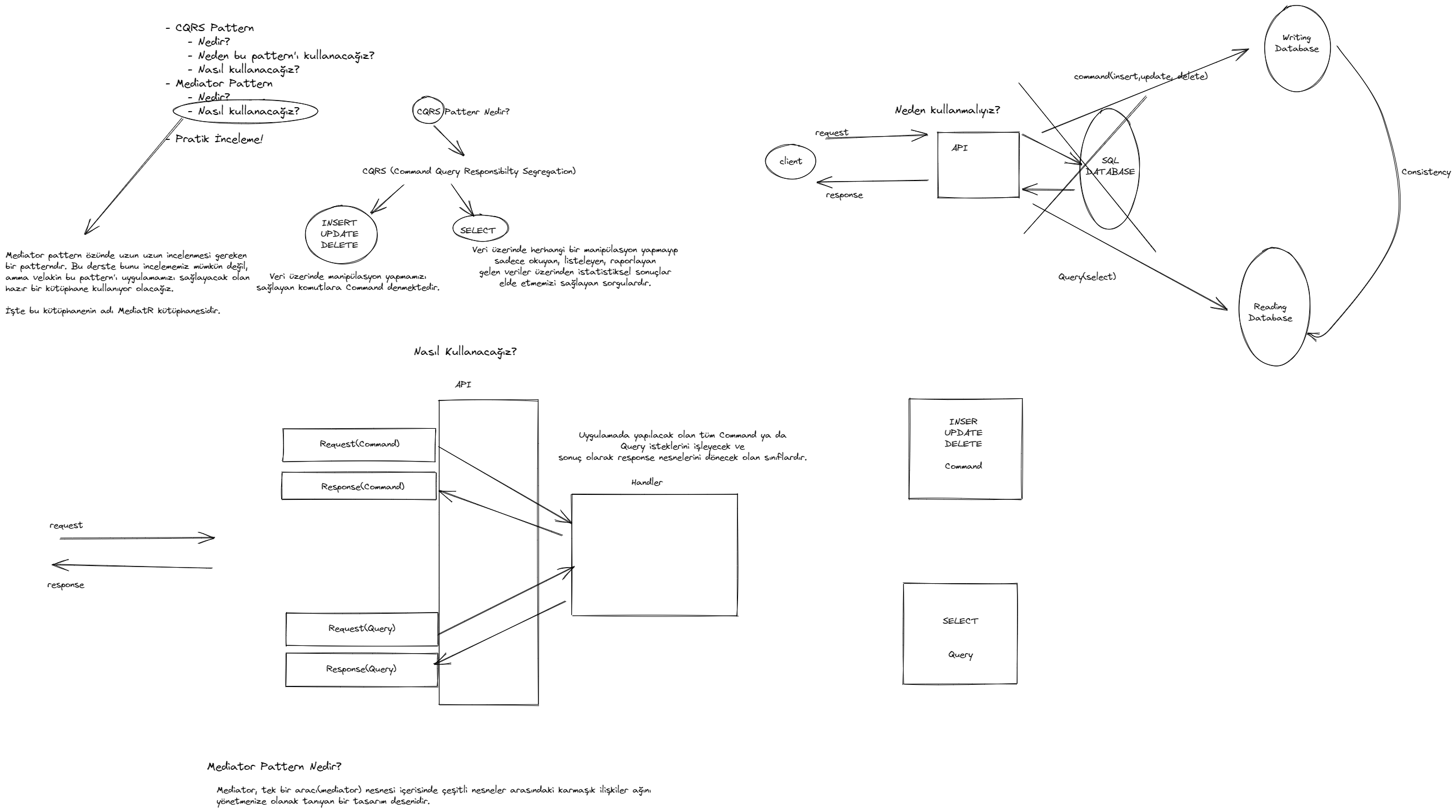
Application katmanının registration edilmesi

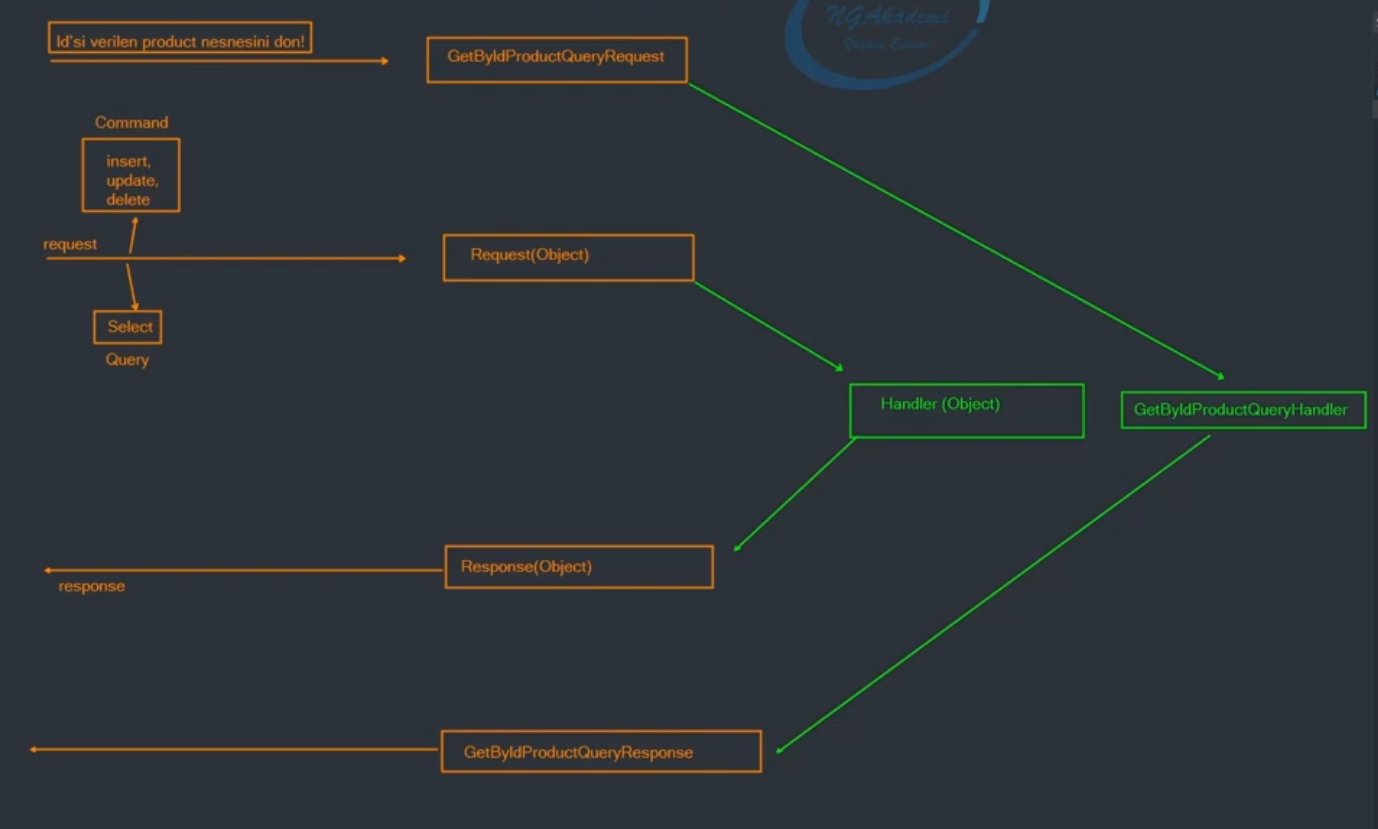


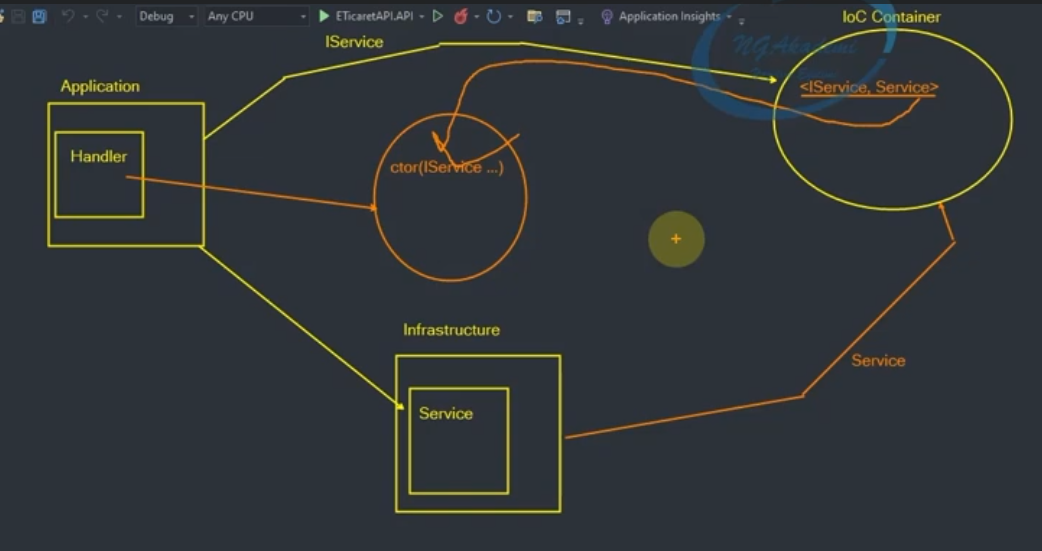
Program.cs eklenmesi



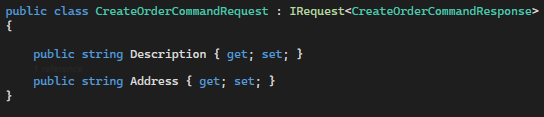
MediatR.DependencyInjection, .NET uygulamalarında MediatR kütüphanesini kullanmayı kolaylaştırır ve bağımlılıkları yönetir. Bu paket, işlemleri komut ve sorgulara ayırarak, kod karmaşıklığını azaltır ve temiz bir mimari oluşturmayı sağlar. MediatR, .NET uygulamalarında CQRS ve aracı desenini kullanmak isteyen geliştiriciler için güçlü bir ar

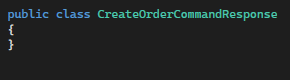




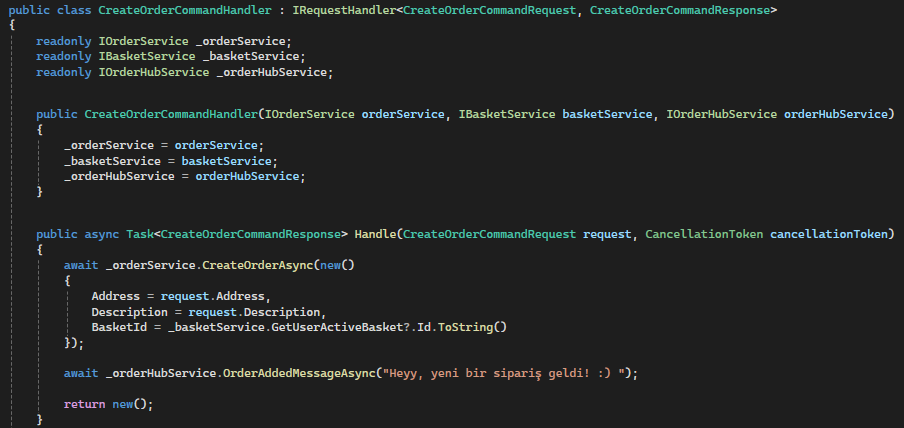


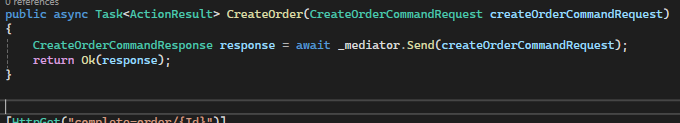
Örnek MediatR kullanımı- Order Command için





Concrete yapıları handler sınıfında işleriz





## **Identity Mekanizması**

"Identity Mekanizması", .NET platformunda kullanıcı kimliklerini yönetmek ve doğrulamak için sağlanan bir dizi araç ve yapılar bütünüdür. Bu mekanizma, kullanıcıların kimliklerini belirlemek, doğrulamak ve yetkilendirmek için kullanılan bir dizi sınıf, arabirim ve hizmetlerden oluşur. .NET'te Identity Mekanizması, özellikle web uygulamaları için kullanıcı yönetimi ve yetkilendirme süreçlerini kolaylaştırır.

### **Identity Mekanizması'nın temel bileşenleri**

Identity Sınıfı: Bu sınıf, bir kullanıcının kimliğini temsil eder. Bir Identity nesnesi, kullanıcı adı, e-posta, roller gibi kullanıcıya ilişkin temel bilgileri içerir.

User Manager Sınıfı: Bu sınıf, kullanıcı hesaplarını yönetmek için kullanılır. Kullanıcı oluşturma, düzenleme, silme gibi işlemleri gerçekleştirmek için kullanılır. ASP.NET Identity tarafından sağlanan bir User Manager, kullanıcı hesaplarıyla ilgili temel işlevleri sağlar.

Role Manager Sınıfı: Bu sınıf, kullanıcı rollerini yönetmek için kullanılır. Bir rol, bir kullanıcı grubunu temsil eder ve uygulamanın farklı parçalarına erişim düzeyini belirler. Role Manager, rolleri oluşturma, düzenleme ve silme gibi işlevleri sağlar.

Authentication Middleware: ASP.NET Core'da, kimlik doğrulama işlemleri için kullanılan bir dizi ara yazılım bulunur. Bu middleware, kullanıcıların kimliklerini doğrular ve kimlik bilgilerini oluşturur. Örneğin, Cookie Authentication Middleware, kullanıcıların oturum açmalarını ve oturumlarını yönetmelerini sağlar.

Authorization Middleware: Bu middleware, kullanıcıların uygulamanın belirli bölümlerine erişimini denetler. Kullanıcının belirli bir kaynağa veya işleme erişip erişemeyeceğini belirlemek için rolleri ve politikaları kullanır.

IdentityDbContext: Bu sınıf, kullanıcı ve rol verilerini saklamak için kullanılan Entity Framework tablosunu temsil eder. IdentityDbContext, kullanıcı hesapları ve rolleri için ilişkilendirilmiş veritabanı tablolarını yönetir.

Identity Razor Sayfaları: ASP.NET Core Identity, kimlik yönetimi işlemlerini gerçekleştirmek için bir dizi Razor sayfası sağlar. Bu sayfalar, kayıt olma, giriş yapma, şifre sıfırlama gibi kullanıcı işlemlerini kolaylaştırır.

Identity Mekanizması, .NET geliştiricilerine kullanıcı kimliklerini, yetkilendirmeyi ve rol yönetimini uygulamak için kapsamlı bir çözüm sunar. Bu sayede geliştiriciler, güvenlik gereksinimlerini karşılayacak ve kullanıcı yönetimi işlemlerini daha kolay yönetebilecekleri bir çerçeve üzerinde uygulama geliştirebilirler.

## **JWT Teknolojisi**

JWT (JSON Web Token), web uygulamalarında kimlik doğrulama ve yetkilendirme için kullanılan bir açık standarttır. .NET platformunda JWT'yi kullanmak için çeşitli kütüphaneler ve araçlar bulunmaktadır. JWT'ler, JSON formatında bilgiyi güvenli bir şekilde taşıyabilen, imzalı ve şifreli bir veri yapısına sahiptirler. Bu yapı, istemci (genellikle tarayıcı) ve sunucu arasında kullanıcı kimliğini doğrulama ve güvenli bir şekilde bilgi taşıma sürecini sağlar.

### **JWT'ler üç bölümden oluşur**

Header (Başlık): JWT'nin türünü (typ) ve kullandığı algoritmayı (alg) içeren JSON formatındaki verilerdir. Bu bölüm, JWT'nin nasıl işleneceğini belirler.

Payload (Taşıma): Kullanıcı hakkındaki bilgilerin (claims) ve ekstra verilerin (payload) bulunduğu bölümdür. JSON formatında bu bilgiler tutulur ve JWT'nin temel işlevselliğini sağlar. Taşıma, kullanıcı kimliği, roller, izinler gibi bilgileri içerebilir.

Signature (İmza): Başlık ve taşıma bölümlerinin birleştirilip belirli bir anahtarla (secret key) imzalanması sonucu oluşturulan veridir. İmza, JWT'nin doğrulanmasını sağlar ve verinin güvenilirliğini sağlar. Bu sayede, JWT'nin manipüle edilip edilmediği veya değiştirilip değiştirilmediği kontrol edilebilir.

.NET platformunda JWT'leri oluşturmak ve doğrulamak için çeşitli kütüphaneler bulunur. Bu kütüphaneler genellikle JWT oluşturma, doğrulama, işaretleme (signing) ve şifreleme işlemlerini kolaylaştıran API'ler sunar. Bunlar arasında popüler olanlar arasında Microsoft.IdentityModel.Tokens ve System.IdentityModel.Tokens.Jwt kütüphaneleri bulunmaktadır. Bu kütüphaneler, JWT'leri oluşturmak için gerekli algoritma desteğini sağlar ve doğrulama süreçlerini kolaylaştırır.

JWT'ler, token tabanlı kimlik doğrulama sistemlerinde yaygın olarak kullanılmaktadır. Özellikle, tek noktadan giriş (single sign-on) sistemlerinde, API tabanlı kimlik doğrulama süreçlerinde ve mikro servis mimarilerinde JWT'lerin kullanımı oldukça yaygındır. Bu tokenler, hafif, taşınabilir ve güvenli bir kimlik doğrulama mekanizması sunarlar, bu da onları modern web uygulamalarında tercih edilen bir çözüm haline getirir.

## **Guard Yapılandırılması**

NET'te "Guard Yapılandırılması" terimi genellikle güvenlik ayarlarının, izinlerin veya erişim kontrollerinin yapılandırılması sürecini ifade eder. Bu, uygulamaların belirli kaynaklara erişimini sınırlama, kimlik doğrulama ve yetkilendirme politikalarını belirleme veya genel güvenlik ayarlarını yapılandırma sürecini içerir. Guard Yapılandırılması, bir uygulamanın güvenlik ve izin kontrollerini sağlamak için gereken adımları içerir ve bu adımlar genellikle aşağıdaki şekillerde gerçekleştirilir:

Kimlik Doğrulama ve Yetkilendirme Ayarları: Bir uygulama genellikle belirli bir kaynağa erişim izninin olup olmadığını belirlemek için kimlik doğrulama ve yetkilendirme işlemlerini gerçekleştirir. Guard yapılandırması, kullanıcı kimliklerini doğrulamak ve bu kimliklerin belirli roller veya izinlere sahip olup olmadığını belirlemek için gerekli ayarları içerir. Bu, kullanıcıların uygulamanın belirli bölümlerine erişimini kontrol etmek için rol tabanlı veya politika tabanlı yetkilendirme mekanizmalarının yapılandırılmasını içerir.

Güvenlik Politikalarının Belirlenmesi: Uygulama için belirli güvenlik politikalarının yapılandırılması da bir Guard Yapılandırması unsuru olabilir. Bu politikalar, güvenli iletişim kuralları, güvenlik duvarı ayarları, giriş güvenliği kontrolleri gibi genel güvenlik standartlarını içerebilir. Bu politikalar genellikle uygulamanın belirli bir endüstri standardına veya yasal düzenlemelere uygunluğunu sağlamak için belirlenir.

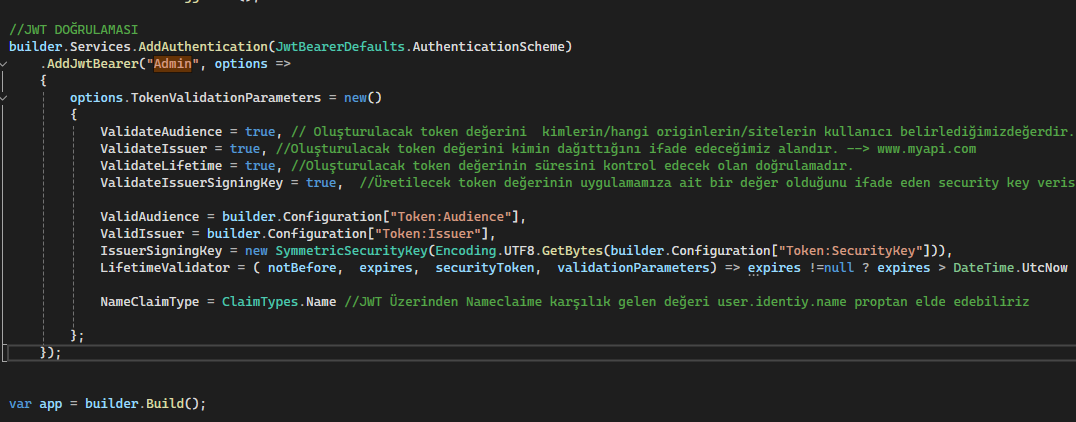
API Erişim Kontrolleri: Bir API'nin erişimini sınırlamak veya korumak için yapılandırılan kontroller de Guard Yapılandırması içinde yer alır. Bu, API'ye yapılan isteklerin yetkilendirme, kimlik doğrulama, IP kısıtlamaları, anahtar tabanlı erişim kontrolleri gibi kriterlere göre kontrol edilmesini sağlar.

Veri Güvenliği Ayarları: Uygulamanın veri güvenliği ile ilgili ayarlar da Guard Yapılandırması kapsamına girer. Bu, veritabanı erişim yetkileri, veri şifreleme politikaları, güvenli veri iletim protokolleri gibi konuları içerir.

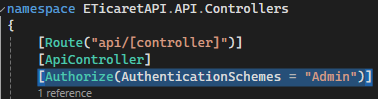
Güvenlik Denetimleri ve İzleme: Guard Yapılandırması, uygulamanın güvenlik durumunu izlemek ve güvenlik denetimlerini yapılandırmak için de kullanılabilir. Bu, güvenlik olaylarının günlüğe kaydedilmesi, izlenmesi ve gerektiğinde alarm verilmesini sağlayan ayarları içerir.

Guard Yapılandırması, bir uygulamanın güvenlik gereksinimlerini karşılamak ve uygun güvenlik önlemlerini uygulamak için gereken adımları belirleme ve yapılandırma sürecidir. Bu süreç, uygulamanın güvenliğini sağlamak ve verilerini korumak için kritik öneme sahiptir.

Program.cs de jwt tanımlanması



Tanımlanan jwt’nin controllerde kullanılması sadece metodlarda kullanılacaksa metod üstüne yerleştirilir.



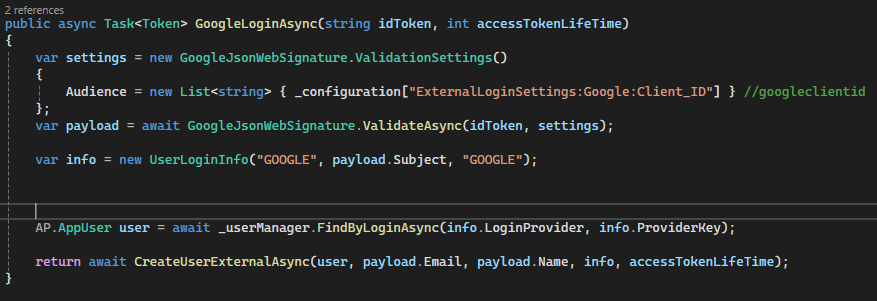
## **Login İşlemleri**

Login işlemleri için token oluşturma



### **Google Login**

Google ile login yapımı

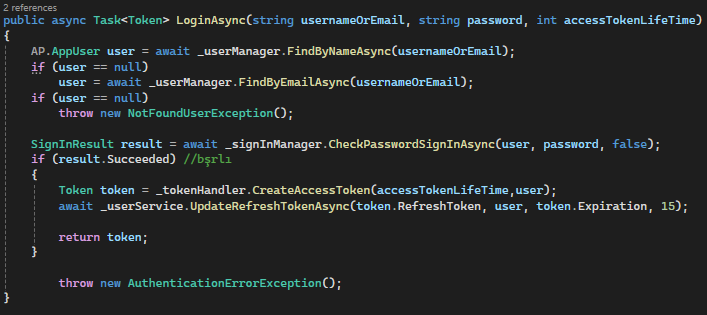


### **Facebook Login**

Facebook ile login yapımı



### **Normal Login**



## **Serilog Sinks PostgreSQL**

Serilog.Sinks.PostgreSQL, Serilog'un bir eklentisi veya "sink"i olarak adlandırılan bir bileşendir ve .NET uygulamalarında PostgreSQL veritabanına logları yazmak için kullanılır. Serilog, yapılandırılabilir, genişletilebilir ve performans odaklı bir loglama kütüphanesidir ve .NET Core ve .NET Framework gibi platformlarda yaygın olarak kullanılır.

Serilog.Sinks.PostgreSQL, Serilog'un desteklediği loglama yapılarını kullanarak .NET uygulamalarının çalışma zamanında oluşan olayları PostgreSQL veritabanına kaydetmeye olanak tanır. Bu, uygulamaların loglarını merkezi bir veritabanında saklamak ve analiz etmek için kullanılabilir, bu da geliştirme, hata ayıklama ve performans izleme süreçlerini kolaylaştırır.

Bu sink, Serilog tarafından desteklenen farklı loglama yapılarını (log event'leri) PostgreSQL veritabanına kaydetmek için kullanılabilir. Bu yapılar arasında mesajlar, hata izleri, debug bilgileri gibi çeşitli log türleri bulunur. Sink, belirli bir PostgreSQL veritabanına bağlanmak için gerekli yapılandırma seçeneklerini sağlar ve logları bu veritabanına belirli bir tabloya veya şema içine yazabilir.

Serilog.Sinks.PostgreSQL kullanırken, aşağıdaki gibi **temel özellikler ve yapılandırma** seçenekleri sunar:

Connection String Ayarları: PostgreSQL veritabanına bağlanmak için gerekli olan bağlantı dizesi (connection string) yapılandırılabilir. Bu, veritabanının konumu, kimlik doğrulama bilgileri ve diğer bağlantı parametrelerini içerir.

Tablo Yapılandırması: Logların hangi PostgreSQL tablosuna yazılacağı yapılandırılabilir. Varsayılan olarak, loglar "Logs" adında bir tabloya yazılır, ancak bu yapılandırma seçeneğiyle değiştirilebilir.

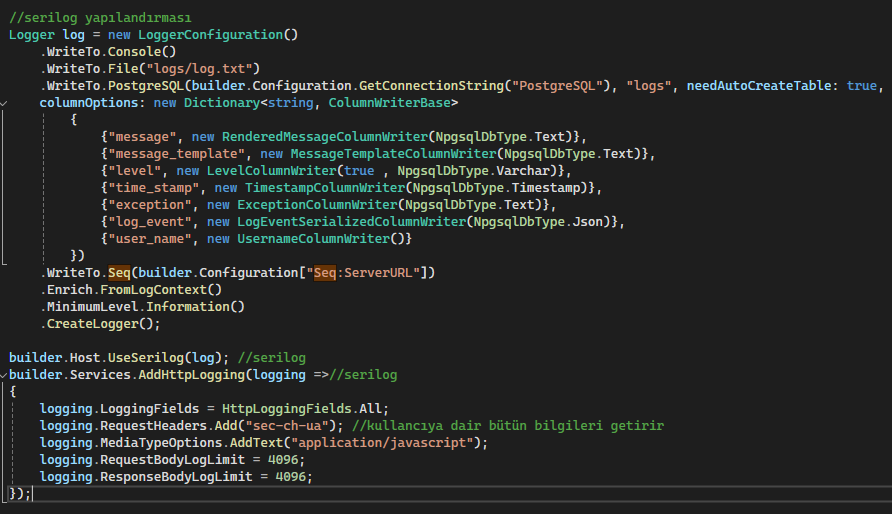
Log Yapısı: Logların PostgreSQL tablosuna hangi sütunlara ve hangi veri tiplerine karşılık geldiği yapılandırılabilir. Bu, logların veritabanında nasıl saklanacağını kontrol etmek için önemlidir.

Arazi Seviyesi Kontrolleri: Hangi log seviyelerinin (debug, info, warning, error, vb.) PostgreSQL'a yazılacağı belirlenebilir. Bu, gereksiz logların veritabanına yazılmasını engellemek için kullanışlıdır.

Performans ve Paralellik Kontrolleri: Logların veritabanına yazılması sırasında performans ve paralellik ayarları yapılandırılabilir. Bu, yoğun loglama senaryolarında performansı optimize etmek için önemlidir.



Serilog.Sinks.PostgreSQL, .NET uygulamalarında güvenilir ve etkili bir loglama çözümü sağlar ve PostgreSQL gibi yaygın kullanılan bir veritabanı yönetim sistemini log verileri için bir depolama aracı olarak kullanmanın kolay bir yolunu sunar. Bu sayede, geliştiriciler uygulama loglarını merkezi bir konumda tutabilir, hata ayıklama süreçlerini iyileştirebilir ve performans izleme amacıyla log verilerini analiz edebilirler.



## **Serilog’u Seq ile Günlük Görselleştirme**

Seq, Serilog tarafından oluşturulan günlükleri toplayan, işleyen ve görselleştiren merkezi bir günlük sunucusudur. Seq, günlüklerinizi tek bir yerde görselleştirmenizi, filtrelemenizi, aramanızı ve analiz etmenizi sağlayarak hata ayıklama ve sorun giderme sürecini büyük ölçüde kolaylaştırır.

### **Seq'in temel özelliklerinden bazıları**

Gerçek zamanlı günlük toplama: Seq, Serilog'dan gelen günlükleri gerçek zamanlı olarak toplayabilir ve işleyebilir.

Zengin görselleştirmeler: Seq, çizelgeler, grafikler, ısı haritaları ve daha fazlası dahil olmak üzere çeşitli görselleştirmeler sunarak günlüklerinizi anlamanıza yardımcı olur.

Güçlü filtreleme ve arama: Seq, seviyeye, mesaj içeriğine, meta verilere, IP adreslerine ve daha fazlasına göre filtreleme ve arama yetenekleri sunar.

Uyarı ve bildirimler: Seq, belirli koşullara uyan günlükler için uyarılar ve bildirimler oluşturabilir.

İşbirliği ve ekip çalışması: Seq, birden fazla kullanıcının günlükleri görüntülemesine, analiz etmesine ve yorum yapmasına olanak tanır.

## **SignalR**

SignalR, .NET platformu için geliştirilmiş olan bir teknolojidir ve gerçek zamanlı web uygulamaları oluşturmak için kullanılır. Bu teknoloji, sunucu ve istemci arasında iki yönlü iletişim sağlar, yani sunucudan istemciye ve istemciden sunucuya veri iletimini destekler. Geleneksel web uygulamalarında, istemci (tarayıcı) sunucuya bir istekte bulunur ve sunucu bu isteği işler ve bir yanıt gönderir. Ancak SignalR ile, sunucu istemcilere anlık güncellemeler gönderebilir ve istemciler de sunucuya anlık veri gönderebilir.

SignalR, WebSocket, Server-Sent Events (SSE), Long Polling gibi farklı iletişim protokolleri üzerinden iletişim kurabilir. Bu sayede, kullanıcıların çeşitli tarayıcılar veya cihazlar üzerinden de aynı deneyimi yaşamalarını sağlar.

### **SignalR Nasıl Çalışır?**

SignalR, istemciler ve sunucu arasında sürekli bir bağlantı kurarak çalışır. Bu bağlantı, istemcilerin sunucuya anında veri göndermesine ve almasına ve sunucunun istemcilere anında veri göndermesine olanak tanır. SignalR, bu bağlantıyı kurmak ve yönetmek için çeşitli taşıma protokolleri kullanır, bunlara şunlar dahildir:

WebSocket: En yaygın ve en verimli protokol olan WebSocket, düşük gecikme süresi ve yüksek bant genişliği sunar.

Server-Sent Events (SSE): WebSocket'in mevcut olmadığı durumlarda kullanılabilen bir alternatif protokoldür.

HTTP Uzun İstekleri: Daha eski tarayıcılarla uyumlu olan ancak daha az verimli bir protokoldür.

### **SignalR'ın temel özellikleri şunlardır**

Gerçek Zamanlı İletişim: SignalR, sunucu ve istemci arasında gerçek zamanlı iletişimi destekler. Bu, anlık güncellemelerin hızlı bir şekilde iletilmesini ve kullanıcıların uygulamada gerçek zamanlı etkileşimlerde bulunmasını sağlar.

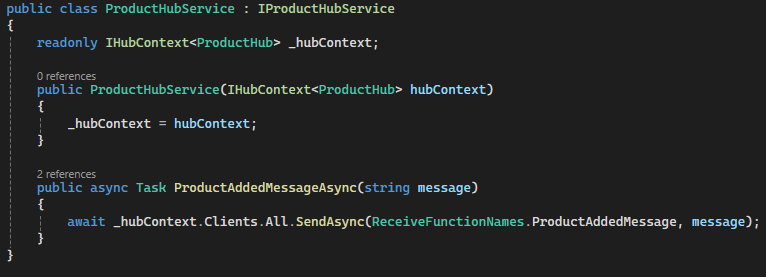
İstemci ve Sunucu API'leri: SignalR, .NET sunucu tarafında ASP.NET Core veya ASP.NET Framework ile çalışabilen bir sunucu API'sine sahiptir. Ayrıca, JavaScript istemci tarafında kullanılabilen bir istemci API'si de bulunmaktadır.

Bağlantı Yönetimi: SignalR, istemci ve sunucu arasındaki bağlantıyı otomatik olarak yönetir. Bağlantı kopmaları veya yeniden bağlanma gibi senaryolarla başa çıkmak için gelişmiş mekanizmalar sağlar.

Farklı Platformlarla Uyumlu: SignalR, .NET platformuyla uyumlu olmasının yanı sıra, JavaScript, Java, Python gibi farklı platformlarla da kullanılabilir.

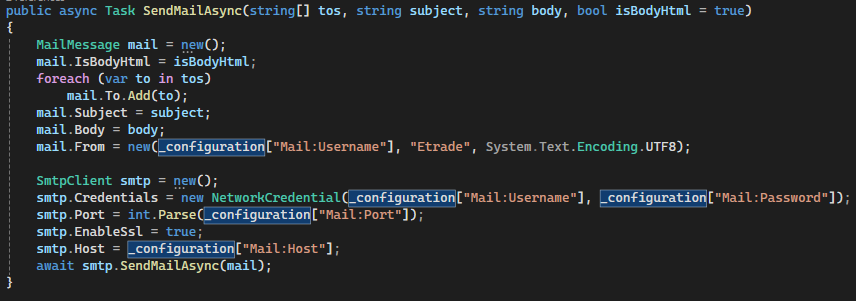
Ölçeklenebilirlik: SignalR, yüksek ölçeklenebilirlik sağlayacak şekilde tasarlanmıştır. Büyük miktarda eşzamanlı bağlantıları yönetebilir ve sunucu yükünü dağıtmak için farklı yöntemler kullanabilir.

SignalR, çeşitli uygulama senaryolarında kullanılabilir, örneğin canlı sohbet uygulamaları, çevrimiçi oyunlar, canlı pazarlama panoları, hisse senedi takip sistemleri gibi uygulamalarda yaygın olarak kullanılmaktadır. Bu, kullanıcıların gerçek zamanlı olarak veri alışverişinde bulunmasını sağlar ve uygulamaları daha etkileşimli hale getirir.

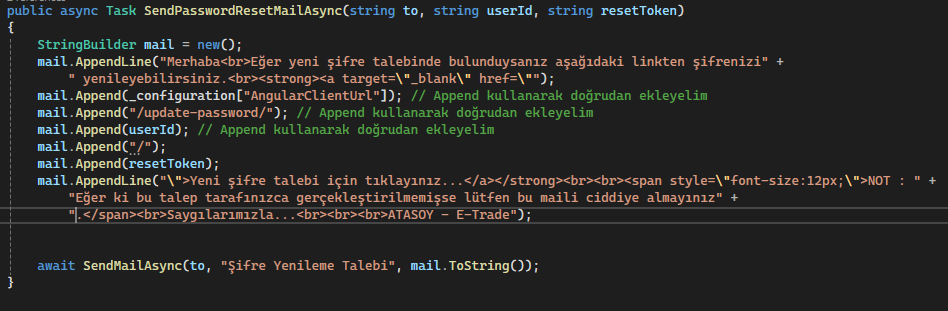


## **Mail Servisi**

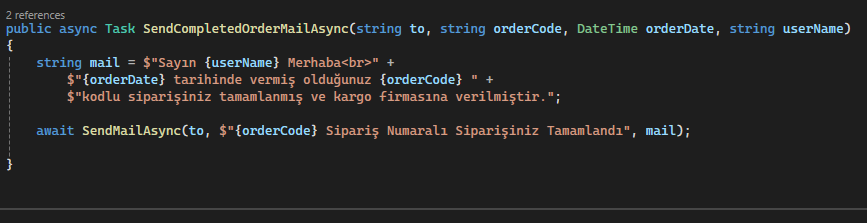
Mail servisinin oluşturulması



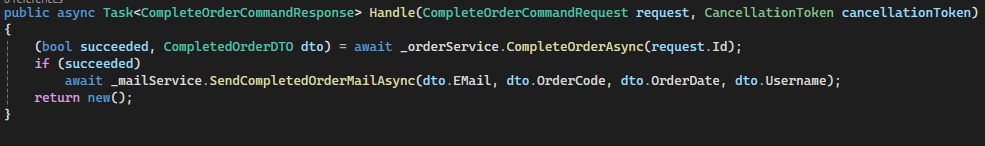
Örnek şifre yenileme mailini atılma kodu



Örnek sipariş tamamlandı mailinin gönderilmesi



Order işlemi başarılıysa eğer başarılı mailinin gönderilmesi



## **Rol Yapılandırılması Mimari**

.NET platformunda menü rol yapılandırması yapmak için genellikle ASP.NET MVC veya ASP.NET Core gibi web uygulama çerçeveleri kullanılır. Bu çerçeveler, kullanıcı yetkilendirme ve rol yönetimi için hazır özellikler sunar. Aşağıdaki adımları takip ederek menü rol yapılandırmasını gerçekleştirebilirsiniz:

Rolleri Tanımlama: İlk olarak, uygulamanızda kullanmak istediğiniz rolleri tanımlayın. Bu roller genellikle "Admin", "Editör", "Standart Kullanıcı" gibi şeyler olabilir. Bu rolleri genellikle veritabanında veya bir yapılandırma dosyasında tutarsınız.

Menü Oluşturma: Kullanıcıların erişebileceği menüleri oluşturun veya mevcut olanları kullanın. Bu, genellikle bir Layout dosyasında yapılır ve tüm sayfalarda kullanılacak bir üst gezinme menüsü içerir.

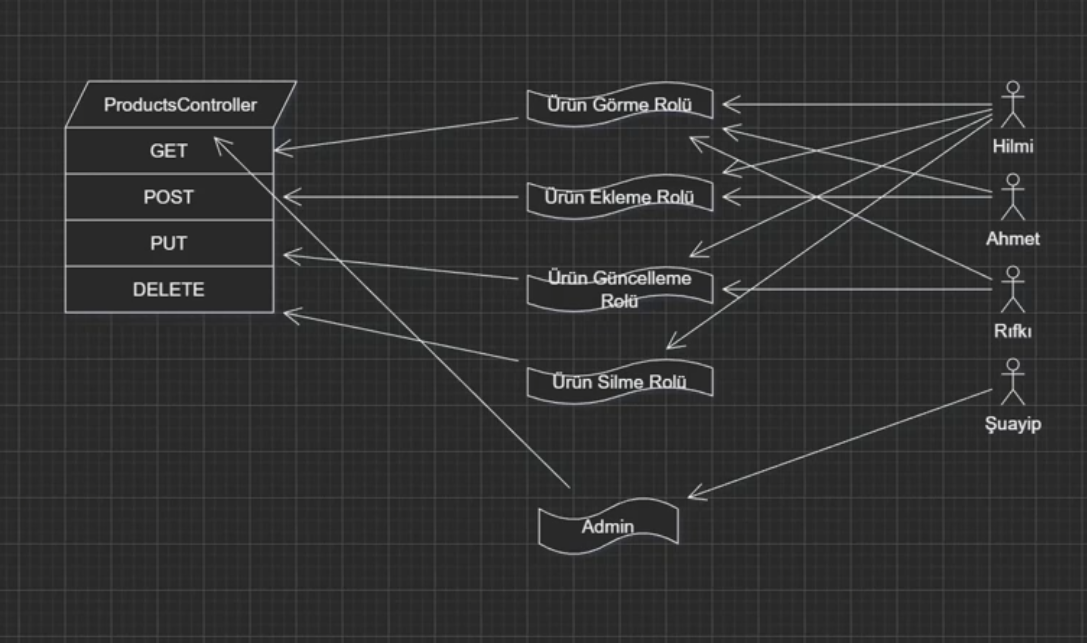
Erişim Kontrolü: Her menü öğesi için, o öğeye erişebilecek rolleri belirleyin. Bu bilgileri genellikle bir yapılandırma dosyasında tutarsınız.

Yetkilendirme Kontrolü: Kullanıcının oturum açtığı zaman veya sayfaları gezindiği zaman, kullanıcının rolünü kontrol edin ve kullanıcının mevcut rolüne göre menü öğelerini filtreleyin. .NET platformunda, bu genellikle bir Controller veya Razor Pages üzerinde yapılır.

Arayüz Güncellemesi: Kullanıcı rolü değiştiğinde veya oturum açtığında, menü öğelerini güncelleyin. Bu, kullanıcının mevcut rolüne göre doğru menü öğelerini görmesini sağlar.

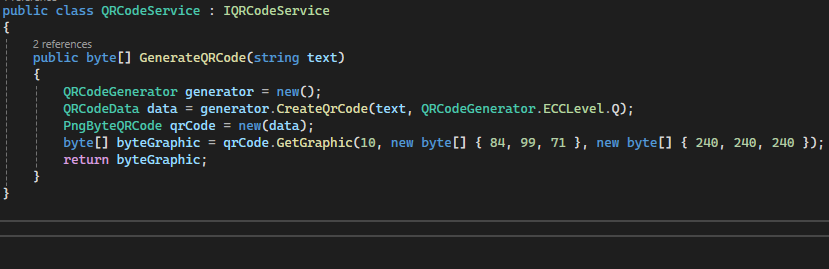
Örneğin, ASP.NET Core kullanarak, menü rol yapılandırmasını sağlamak için Authorization Middleware ve Razor Pages veya Razor Views kullanabilirsiniz. Bunun için, kullanıcının rolüne göre menü öğelerinin görünüp görünmeyeceğini kontrol eden bir Razor Pages filtresi oluşturabilirsiniz.

Bu filtre, kullanıcının mevcut rolüne göre menü öğelerini dinamik olarak filtreleyebilir ve sadece o kullanıcının erişimine izin verilen öğeleri gösterir. Bu şekilde, kullanıcılar yalnızca yetkilerine uygun menü öğelerini görürler ve kullanıcı deneyimi daha tutarlı hale gelir.

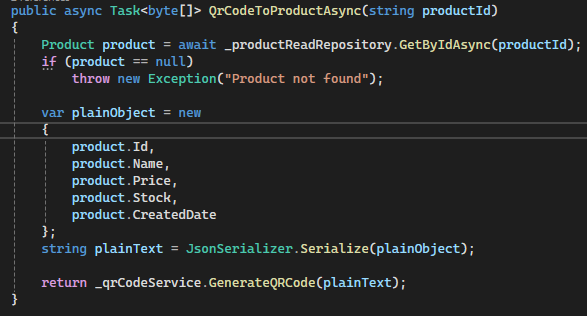


## **QR Coder**

QR Code oluşturulması



Ürünlere qr Code atanması



## **Backend için Generics Repositories yapılandırılması**

Jenerik yapılar (generics), .NET framework'ünde tür (type) güvenliğini koruyarak, kodun yeniden kullanılabilirliğini ve performansını artıran güçlü bir özelliktir. Generics, sınıfların, yapıların, arayüzlerin ve yöntemlerin (method) belirli bir türle çalışacak şekilde tanımlanmasına olanak tanır ve bu türü çalışma zamanında belirleyerek kodun daha esnek ve yeniden kullanılabilir olmasını sağlar.

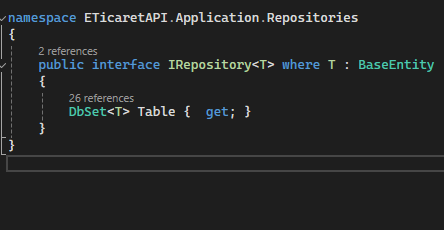
## **Generics Kullanmanın Avantajları**

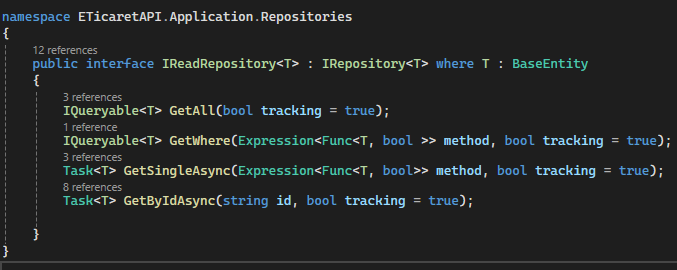
Tür Güvenliği (Type Safety): Hataları derleme (compile) zamanında yakalar ve tür dönüşümü hatalarını (type casting) önler.

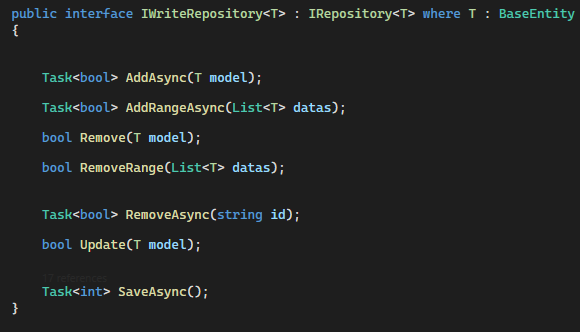
Yeniden Kullanılabilirlik (Reusability): Aynı kodu farklı veri türleriyle kullanmanızı sağlar.

Performans: Kutulama (boxing) ve kutudan çıkarma (unboxing) işlemlerini azaltarak performansı artırır.

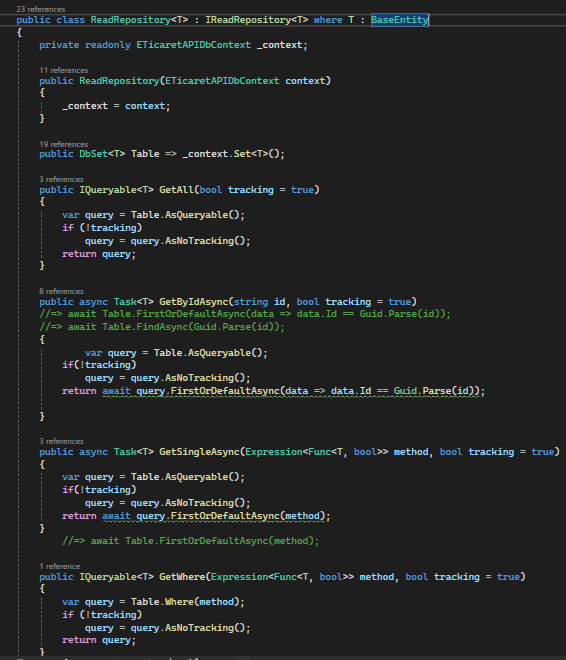
Read ve Write Repository’lerin interfacelerin oluşturulması

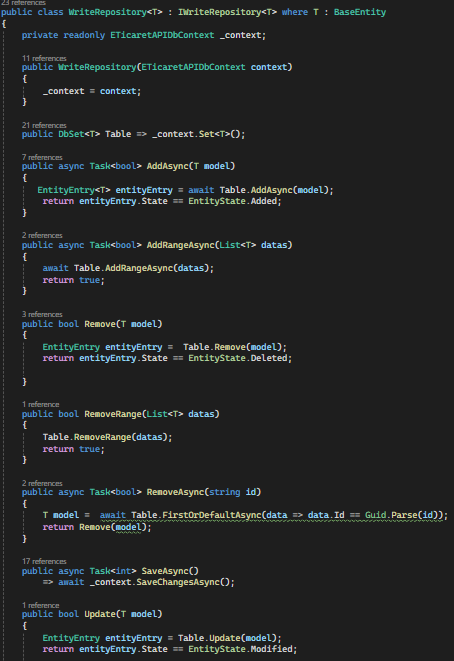




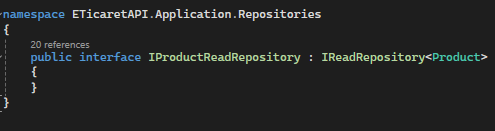


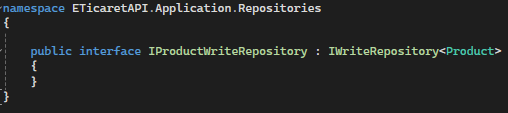
Read ve Write Repositorylerin Concrete’lerin oluşturulması



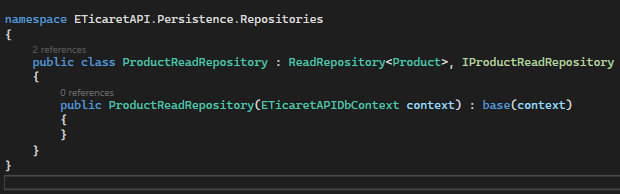


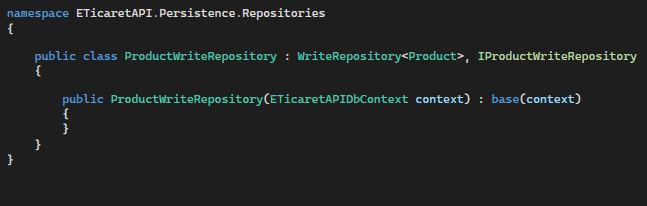
Örnek Product repository Read ve Write interface kullanımı





Örnek Product repository Read ve Write concrete kullanımı





# **Frontend yazılımı için kullanılan teknolojiler**

## **Angular Moduler Yapı**

Angular'da modül yapısı, uygulamanızı parçalara bölmek ve her bir parçayı yönetilebilir, bağımsız bileşenlere ayırmak için kullanılan bir yapıdır. Modül, genellikle benzer özelliklere veya işlevselliğe sahip bileşenleri, direktifleri, hizmetleri ve diğer Angular öğelerini gruplamak için kullanılır. Bu, büyük ölçekli uygulamaların geliştirilmesini kolaylaştırır ve bakımını daha yönetilebilir hale getirir.

Angular modüllerinin temel amacı, uygulamanızı parçalara ayırmak ve her bir parçayı bağımsız olarak geliştirmek, test etmek ve sürdürmek için izole bir ortam sağlamaktır. Her modül, kendi kapsamını oluşturur ve dışarıdaki diğer modüllerle etkileşim kurabilir. Bu sayede, uygulamanın farklı parçaları arasında sıkı bağlantılar olmadan çalışabilirsiniz.

Bir Angular uygulaması genellikle bir ana modül (root module) ile başlar ve bu modül, genellikle uygulamanın başlangıç noktasıdır. Ana modül, diğer modülleri içerebilir ve uygulamanın genel bileşenlerini ve hizmetlerini tanımlar. Ayrıca, uygulamanın yükleme işlemi sırasında Angular'a hangi bileşenlerin yükleneceğini bildirir.

Angular modülleri, @NgModule dekoratörü ile işaretlenir. Bu dekoratör, bir JavaScript sınıfına eklenir ve modülün özelliklerini tanımlamak için kullanılır. @NgModule dekoratörü, genellikle aşağıdaki özellikleri içeren bir nesne alır:

declarations: Modülün bileşenlerini, direktiflerini ve boru hatlarını tanımlar.

imports: Modülün bağımlı olduğu diğer modülleri içerir.

exports: Bu modül tarafından tanımlanan bileşenleri ve direktifleri dışarıya açar, böylece başka modüllerde kullanılabilir hale gelir.

providers: Modülde kullanılacak olan hizmetleri tanımlar.

bootstrap: Uygulamanın başlangıç bileşenlerini belirtir. Sadece ana modülde kullanılır.

Örneğin, bir alışveriş uygulaması için ürünleri listeleyen bir modül oluşturmayı düşünelim. Bu modülün bir ProductListComponent gibi bir bileşeni olabilir. Bu bileşenin bulunduğu modül, bu bileşeni declarations içinde tanımlayacaktır. Ayrıca, bu modülün Angular'ın CommonModule gibi ön tanımlı modülleri kullanması gerekebilir, bu nedenle bu modülleri imports içine ekleyecektir. Son olarak, başka yerlerde bu bileşeni kullanmak isteyebiliriz, bu nedenle exports içinde bu bileşeni belirtiriz.

Angular modül yapısı, büyük ölçekli uygulamaları geliştirirken düzeni, bakımı ve geliştirilebilirliği artıran güçlü bir araçtır. Bu yapı, kodu organize etmeyi ve farklı ekiplerin aynı uygulama üzerinde çalışmasını kolaylaştırır.

## **Bootstrap(UI) ve Angular Material(Admin) – Dizayn/Tasarım**

Bootstrap ve Angular Material, web uygulamaları oluşturmak için popüler CSS ve UI bileşen kütüphaneleridir. Her ikisi de geliştiricilere hızlı ve şık kullanıcı arayüzleri oluşturma imkanı sunar, ancak farklı özelliklere ve yaklaşımlara sahiptirler. İşte her birinin geniş ve detaylı bir açıklaması:

### **Bootstrap(UI)**

Bootstrap, Twitter tarafından geliştirilen ve yayınlanan açık kaynaklı bir CSS ve JavaScript kütüphanesidir. Bootstrap, web uygulamaları ve mobil uygulamalar için kullanıcı arayüzü bileşenlerini hızlı bir şekilde oluşturmayı amaçlar.

Özellikler:

Grid Sistemi: Bootstrap, duyarlı bir grid sistemine sahiptir. Bu, kolayca sütunlar ve satırlar oluşturmanıza ve uygulamanızın farklı cihazlara uyumlu olmasını sağlamanıza olanak tanır.

Hazır Bileşenler: Bootstrap, düğmeler, formlar, navigasyon menüleri, kartlar, modaller ve daha birçok yaygın kullanılan UI bileşenini içerir. Bu bileşenlerin birçoğu önceden tasarlanmış ve önceden stilize edilmiştir.

Özelleştirme Kolaylığı: Bootstrap, temalar ve stil özellikleriyle özelleştirilebilir. Özel CSS kuralları eklemek veya Bootstrap'in varsayılan stillerini geçersiz kılmak için kolay bir yol sunar.

Dokümantasyon ve Topluluk: Bootstrap, kapsamlı bir dokümantasyona ve büyük bir topluluğa sahiptir. Bu, geliştiricilerin ihtiyaç duydukları bilgilere ve desteklere kolayca erişmelerini sağlar.

### **Angular Material(Admin)**

Angular Material, Google tarafından geliştirilen ve Angular uygulamaları için kullanıcı arayüzü bileşenlerini içeren bir kütüphanedir. Material Design ilkelerine dayanır ve modern ve şık bir görünüm sunar.

Özellikler:

Material Tasarım İlkeleri: Angular Material, Google'ın Material Design ilkelerine dayanır. Bu, net, basit ve sezgisel bir kullanıcı deneyimi sunar.

Hazır Bileşenler: Angular Material, düğmeler, form kontrolleri, kartlar, menüler, tablolar, ilerleme çubukları ve diğer birçok UI bileşenini içerir. Bu bileşenler, Material Design özelliklerine uygun olarak tasarlanmıştır.

Gelişmiş Animasyonlar: Angular Material, animasyonları ve geçişleri kolayca entegre edilebilen gelişmiş bir animasyon API'si sunar. Bu, uygulamanıza daha etkileyici bir görünüm kazandırabilir.

Dokümantasyon ve Topluluk: Angular Material, kapsamlı bir dokümantasyona ve geniş bir topluluğa sahiptir. Bu, geliştiricilerin bileşenlerin kullanımı ve özelleştirilmesi konusunda bilgi edinmelerini kolaylaştırır.

## **Angular Toastr(UI) ve Angular Alertify(Admin) -- Bildiri**

Angular Toastr ve Angular Alertify, Angular uygulamalarında kullanılan pop-up bildirimler ve uyarılar için iki popüler kütüphanedir. İkisi de kullanıcıya bilgilendirme, uyarı veya hata mesajları göstermek için kullanılır, ancak her birinin özellikleri ve kullanımı biraz farklıdır. İşte her ikisinin geniş ve detaylı bir açıklaması:

### **Angular Toastr(UI)**

Angular Toastr, Angular uygulamalarında kullanıcıya pop-up bildirimler göstermek için kullanılan bir kütüphanedir. Toastr, Bootstrap dikkat çekici bildirimlerinin Angular uygulamalarına uyarlanmış bir sürümüdür.

Özellikler:

Gösterişli Bildirimler: Toastr, kullanıcıya çekici ve dikkat çekici bildirimler göstermek için kullanılabilir. Bu bildirimler, kullanıcının dikkatini çekmek için animasyonlar ve renkler kullanır.

Özelleştirme Seçenekleri: Toastr, bildirimlerin görünümünü ve davranışını özelleştirmek için geniş bir yelpaze sunar. Renkler, animasyonlar, pozisyonlar ve bildirim süreleri gibi birçok özellik özelleştirilebilir.

Farklı Bildirim Türleri: Toastr, farklı bildirim türlerini destekler, bunlar arasında success (başarı), error (hata), warning (uyarı) ve info (bilgi) gibi yaygın türler bulunur.

Dinamik Kullanım: Toastr, Angular bileşenlerinin veya hizmetlerinin herhangi bir yerinde çağrılabilir. Bu, uygulamanızın herhangi bir noktasından bildirim gösterebilmenizi sağlar.

### **Angular Alertify(Admin)**

Angular Alertify, Angular uygulamalarında kullanıcıya pop-up bildirimler ve uyarılar göstermek için kullanılan bir diğer kütüphanedir. AlertifyJS'nin Angular için bir sarmalayıcısıdır.

Özellikler:

Basit Kullanım: Alertify, basit ve kullanımı kolay bir API sunar. Bildirim göstermek için tek bir satır kod yeterlidir.

Özelleştirme Seçenekleri: Alertify, bildirimlerin görünümünü ve davranışını özelleştirmek için seçenekler sunar. Renkler, butonlar, başlık ve metinler gibi birçok özellik özelleştirilebilir.

Farklı Bildirim Türleri: Alertify, farklı bildirim türlerini destekler, bunlar arasında success (başarı), error (hata), warning (uyarı) ve info (bilgi) gibi yaygın türler bulunur.

Hızlı ve Hafif: Alertify, hafif bir kütüphanedir ve Angular projenize eklemek için minimum çaba gerektirir.

## **Spinner - Loading**

Spinner, kullanıcıya bir işlemin devam ettiğini göstermek için kullanılan dairesel bir ilerleme göstergesidir. Spinner, genellikle web sayfalarında veya uygulamalarda bir içerik yüklenirken veya bir işlem gerçekleşirken kullanılır.

Özellikler:

İlerleme Göstergesi: Spinner, dairesel bir ilerleme göstergesi olarak kullanılır. Bu, kullanıcının bir işlemin devam ettiğini ve tamamlanmak üzere olduğunu gösterir.

Görsel Değişkenlik: Spinner, farklı boyutlarda, renklerde ve stillerde olabilir. Bu, tasarımın ve kullanıcı deneyiminin gereksinimlerine uyacak şekilde özelleştirilebilir.

Duyarlı Tasarım: Spinner, duyarlı tasarım ilkelerine uygun olarak oluşturulabilir. Bu, farklı cihazlarda ve ekran boyutlarında tutarlı bir deneyim sağlar.

İşlem Durumunu Belirtme: Spinner, kullanıcıya bir işlemin devam ettiğini ve sayfanın veya uygulamanın yanıt verdiğini belirtmek için kullanılır. Bu, kullanıcıya beklediği bir geri bildirim sağlar.

## **Openbase – Dosya Yükleme**

Openbase dosya yükleme Angular için oldukça yaygın bir gereksinimdir ve genellikle kullanıcıların web uygulamalarına dosya eklemelerine olanak tanır. Angular'da dosya yükleme işlemi genellikle çeşitli kütüphaneler veya doğrudan Angular'ın modülleri kullanılarak gerçekleştirilir. İşte bu konuda geniş ve detaylı bir açıklama:

Dosya Seçme Arayüzü Oluşturma:

Dosya yükleme işlemi genellikle bir dosya seçme arayüzüyle başlar. Kullanıcılar, bilgisayarlarından veya diğer cihazlarından dosyaları seçmek için bir düğme veya alan arayüzü kullanır. Bu, HTML <input type="file"> etiketiyle kolayca yapılabilir.

Angular Component ve Template Oluşturma:

Angular'da dosya yükleme işlemini gerçekleştirmek için bir bileşen ve bir şablon oluşturmanız gerekir. Bu bileşen, dosya yükleme işlemini yönetmek için gereken iş mantığını içerir.

Dosya Yükleme Servisi Oluşturma:

Dosya yükleme işlemlerini yönetmek için genellikle bir servis oluşturulur. Bu servis, dosyaların sunucuya yüklenmesi ve dosya işlemleri için gerekli olan HTTP isteklerini yönetir.

HTTP İstekleriyle Sunucuya Dosya Gönderme:

Angular'da dosya yükleme genellikle HTTP POST istekleri kullanılarak gerçekleştirilir. Bu istekler, seçilen dosyaları sunucuya göndermek için kullanılır. İsteğin gövdesinde dosya verisi genellikle bir FormData nesnesi olarak taşınır.

Sunucu Tarafı İşlemler:

Sunucu tarafında, alınan dosyaların işlenmesi ve sunucuya kaydedilmesi gerekmektedir. Bu işlemler, genellikle sunucunun hangi teknolojiyi kullandığına bağlı olarak değişir. Örneğin, Node.js için Express, Python için Django veya Flask kullanılabilir.

Geri Bildirim ve Hata İşleme:

Dosya yükleme işlemi sırasında kullanıcıya geri bildirim sağlanması önemlidir. Başarılı yükleme durumunda kullanıcıya bir başarı mesajı gösterilmeli, hatalar durumunda ise uygun hata mesajları verilmelidir. Bu, kullanıcı deneyimini artırmak ve kullanıcıların sorunsuz bir şekilde dosya yükleyebilmelerini sağlamak için önemlidir.

İlerleme Göstergesi Ekleme:

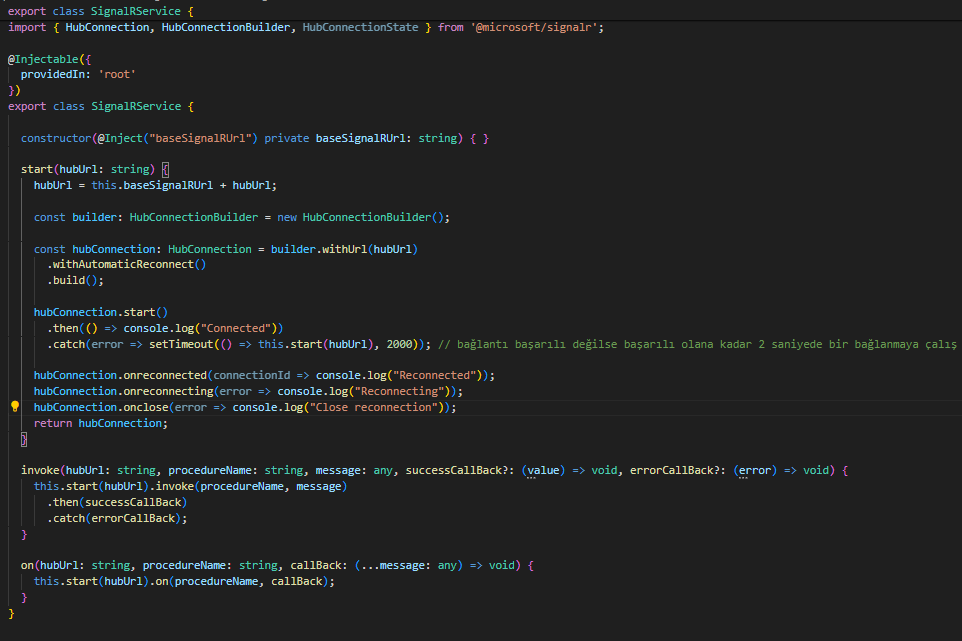
Büyük dosyaların yüklenmesi durumunda, kullanıcıya dosyanın yüklendiğini ve işlemin devam ettiğini göstermek için bir ilerleme göstergesi eklenmesi önemlidir. Bu, kullanıcının işlemin ne kadarının tamamlandığını görmesine olanak tanır.

Güvenlik Dikkate Alınması:

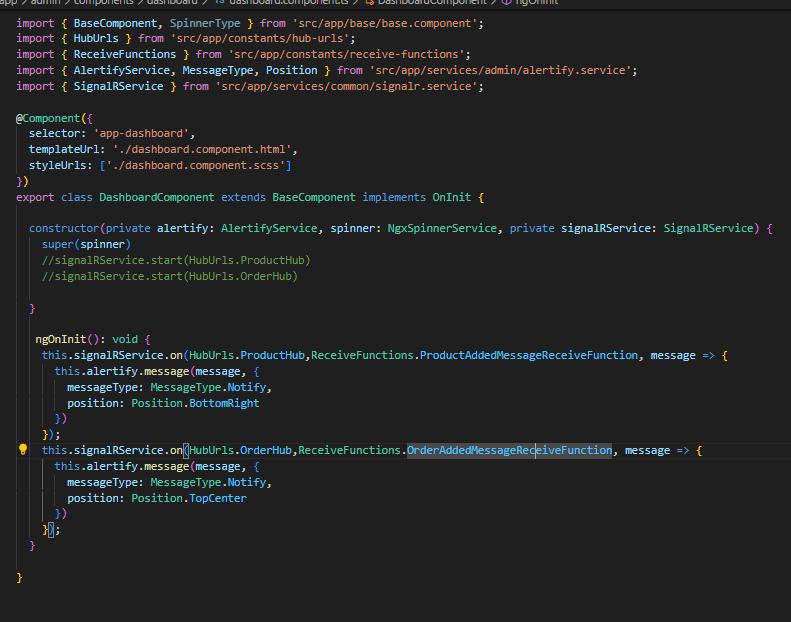
Dosya yükleme işlemleri sırasında güvenlik önlemlerinin alınması çok önemlidir. Kötü niyetli kullanıcıların kötü amaçlı dosyalar yüklemelerini engellemek için dosya türü doğrulaması ve dosya boyutu sınırlamaları gibi güvenlik kontrolleri yapılmalıdır.

## **SignalR - Angular**

Signal servisinin oluşturulması



Signal servisinin kullanılması başarılı bir orderin veya productun eklenmesi sonrasında eş zamanlı bilginin dashboard componentinde gösterilmesi



## **Dynamic Component Loading**

### **Dynamic Component Loading Nedir?**

Dynamic Component Loading, bir Angular uygulamasında, çalışma zamanında (runtime) belirli bir bileşeni (component) dinamik olarak yükleme ve oluşturma sürecidir. Bu, özellikle belirli koşullara göre bileşenleri yüklemek, büyük uygulamalarda performansı iyileştirmek ve belirli özellikleri modüler hale getirmek için kullanışlıdır.

### **Dynamic Component Loading'in Avantajları**

Performans: Yalnızca ihtiyaç duyulan bileşenlerin yüklenmesi, başlangıç yükleme süresini ve kaynak kullanımını azaltır.

Esneklik: Kullanıcı etkileşimlerine veya uygulama durumuna göre farklı bileşenlerin yüklenebilmesi, uygulamanın daha esnek ve kullanıcı dostu olmasını sağlar.

Modülerlik: Bileşenlerin bağımsız olarak yüklenmesi, kodun daha modüler ve yönetilebilir olmasına yardımcı olur.

Dynamic Component Loading angular için servisin oluşturulması ve fonksiyonlarn açıklanması

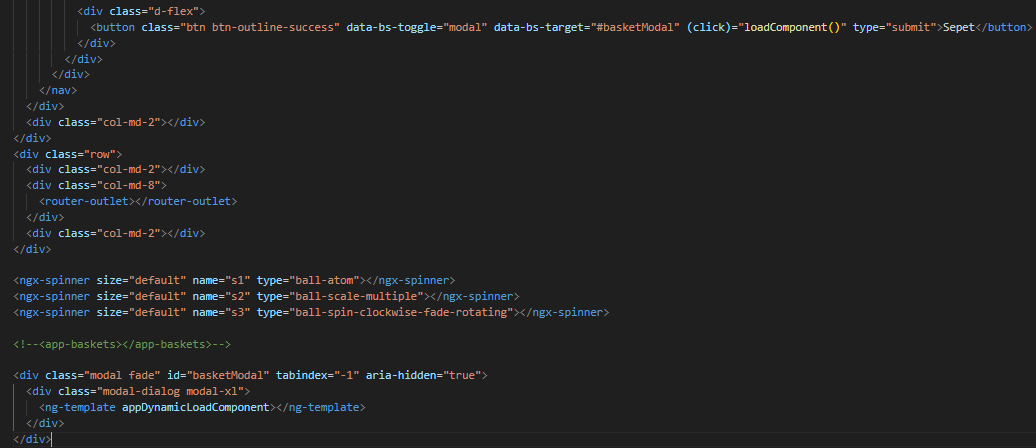


//ViewContainerRef : Dinamik olarak yüklenecek componenti içerisinde barındıran container'dır. (Her dinamik yükleme sürecinde önceki view'leri clear etmemiz gerekmektedir.)

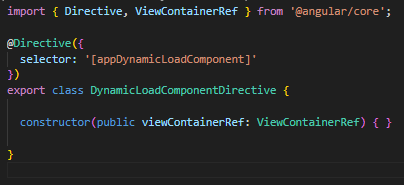
//ComponentFactory : COmponent'lerin instance'larını oluşturmak için kullanılan nesnedir.

//ComponentFactoryResolver : Belirli bir component için ComponentFactory'i resolve eden sınıftır. İçerisindeki resolveComponentFactory fonksiyonu aracılığıyla ilgili componente dair bir ComponentFactory nesnesi oluşturup, döner.

App.componentte html kullanılması



Directiv oluşturularak genel bir kullanım sağlandı



## **Delete Directive**

### **Angular Directive Nedir?**

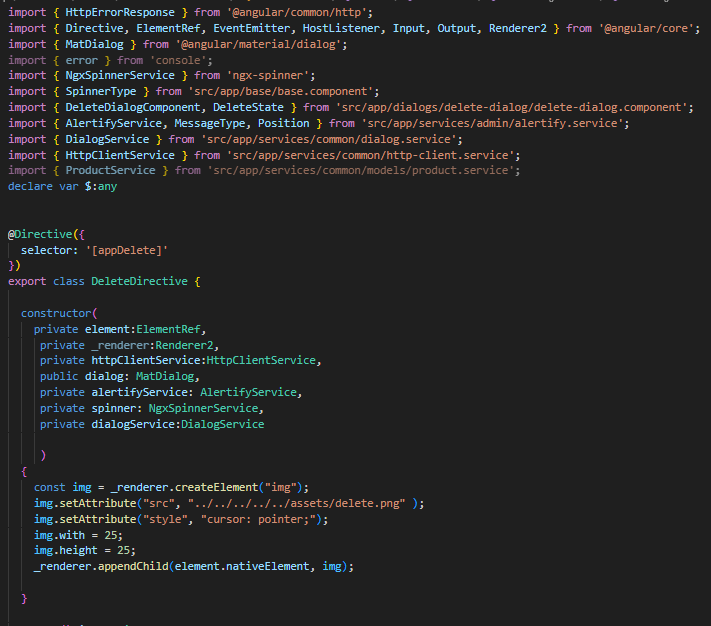
Angular directive'ler, Angular framework'ünde bir bileşenin DOM (Document Object Model) yapısını ve davranışını değiştiren özel talimatlardır. Directive'ler, HTML öğelerine yeni davranışlar ekleyebilir, öğelerin görünümünü değiştirebilir veya tamamen yeni DOM yapıları oluşturabilir. Angular'da üç ana tür directive bulunmaktadır:

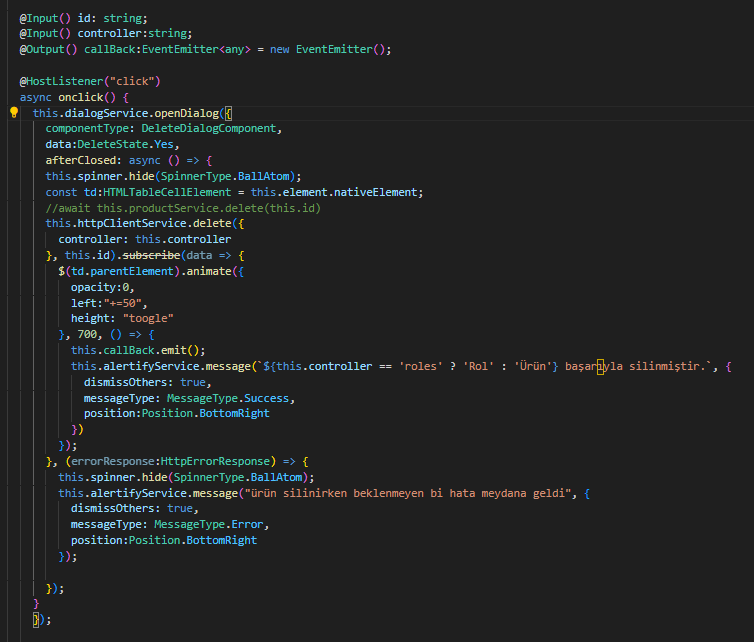
Component Directives (Bileşen Yönergeleri): Bu, Angular bileşenleridir ve @Component dekoratörü ile tanımlanır. Bir template, stil ve davranış içeren directive'lerdir.

Attribute Directives (Özellik Yönergeleri): HTML elementlerinin görünümünü veya davranışını değiştiren directive'lerdir. @Directive dekoratörü ile tanımlanır.

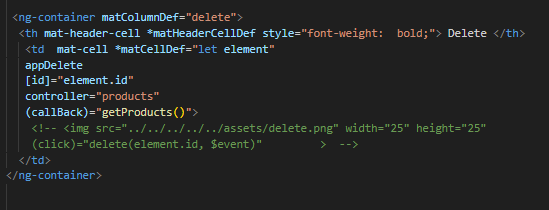
Structural Directives (Yapısal Yönergeler): DOM yapısını değiştiren directive'lerdir. Elementleri ekleyebilir veya kaldırabilir. \* sembolü ile belirtilir ve @Directive dekoratörü ile tanımlanır.

Delete Directivinin oluşturulması





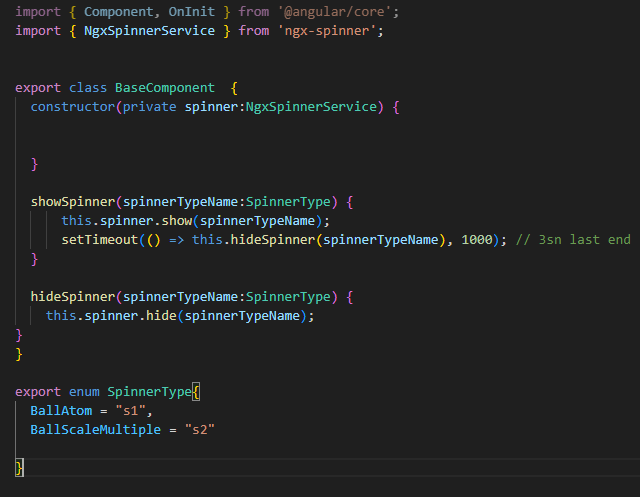
Delete Directivinin kullanılması



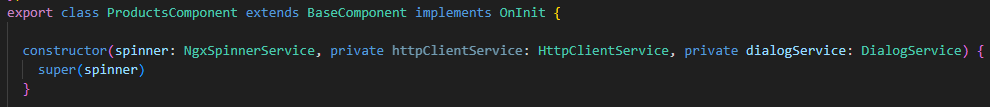
## **Base Yapılar**

Jenerik yapıya benzer ama extends edilerek kullanılan yapılardır. Örnek için hazırlanmış base spinner:

### **Spinner Base Yapısı**



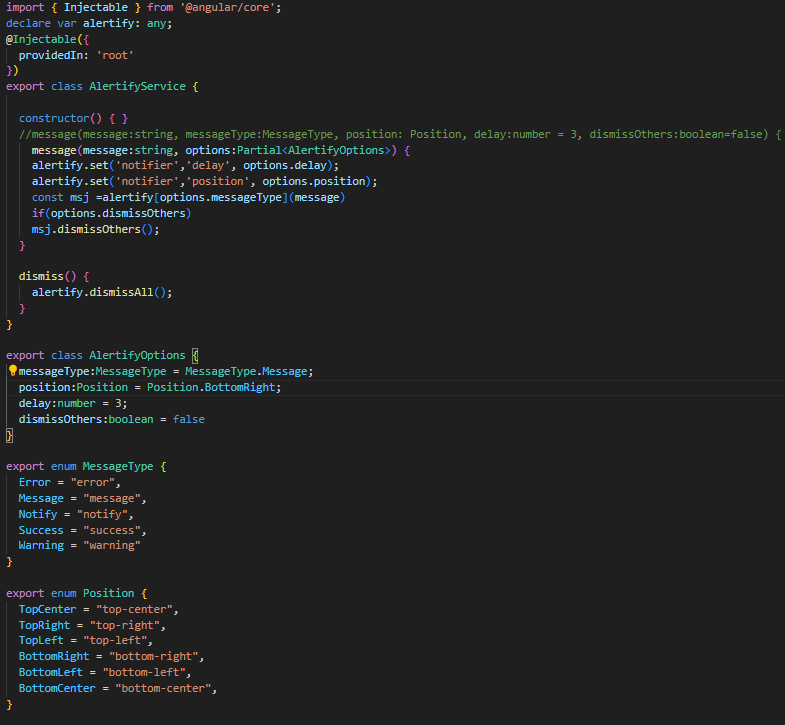
Products için constructorda kullanımı basespinnein



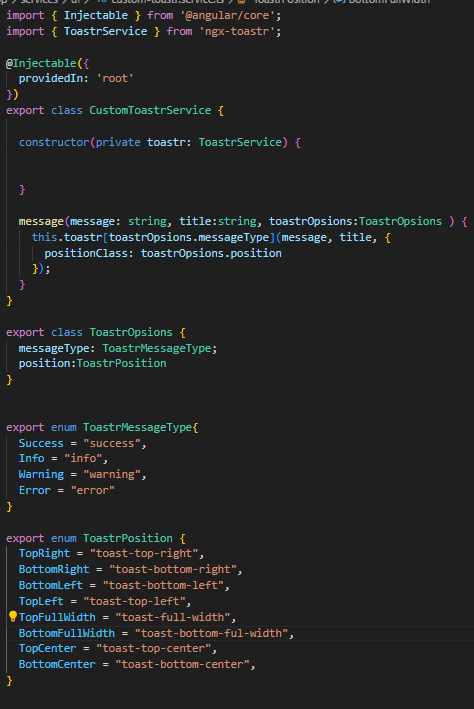
Products için kullanımı spinner,



### **Alertify Base Yapısı**

Alertfiy base yapısının kullanımı 

### **Toastr Base Yapısı**



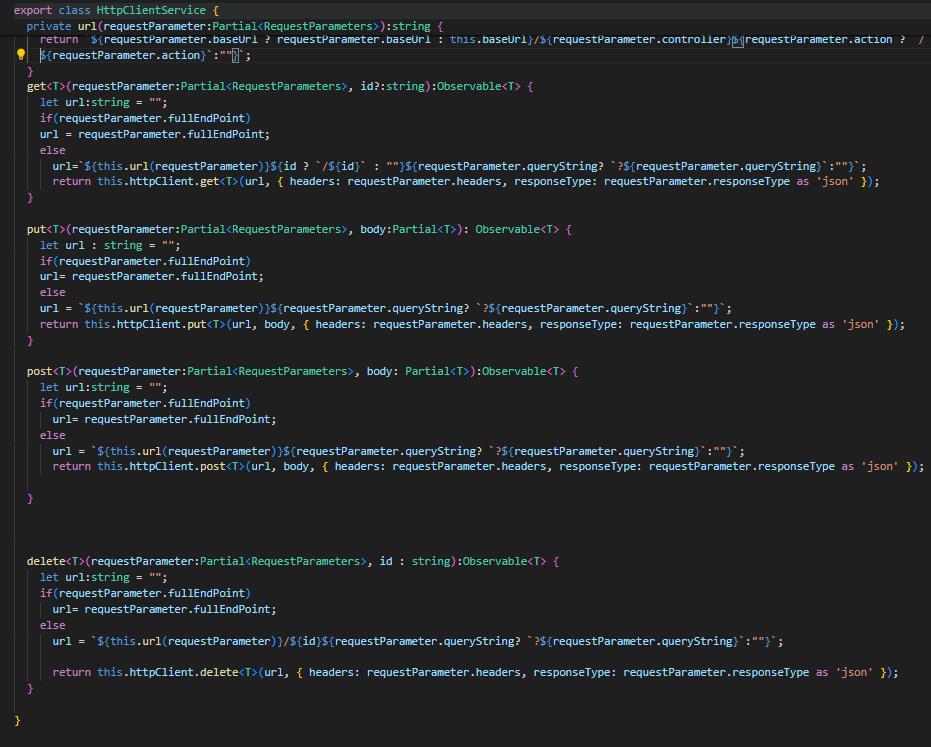
Toastr servisinin kullanımı – constructor a toastrService(customtoastrservice) eklendikten sonra

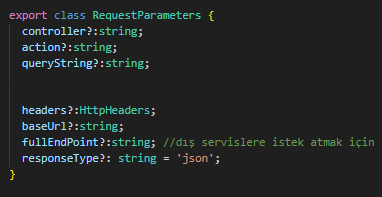


## **Service Yapılar**

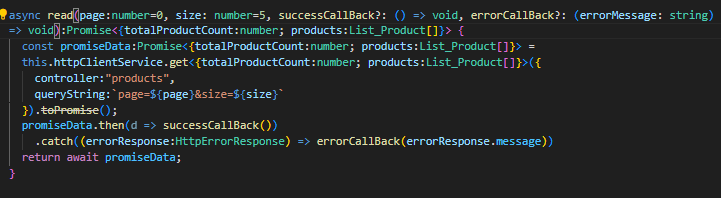
Http requestleri için jenerik base tasarımı

### **HttpClientService**





Örnek Http servisinin kullanımı:

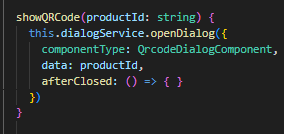


Dialog Penceresi nin Servisinin hazırlanması

### **Dialog Penceresi**



Constructor dialog servisinin eklendikten sonra kullanımı



# **KAYNAKÇA**

BACKEND YAZILIM KODLARI: <https://github.com/umeyiratasoy/E-Trade-Asp.NETCore>

FRONTEND YAZILIM KODLARI: <https://github.com/umeyiratasoy/E-Trade-Angular>

## **Teknolojilerin Kaynakçaları**

## **Backend**

<https://www.gencayyildiz.com/blog/nedir-bu-onion-architecture-tam-teferruatli-inceleyelim/>

<https://learn.microsoft.com/en-us/ef/>

<https://www.gencayyildiz.com/blog/asp-net-core-uygulamalarinda-corscross-origin-resource-sharing-politikasi-ayarlama/>

<https://docs.fluentvalidation.net/en/latest/aspnet.html>

<https://www.gencayyildiz.com/blog/entity-framework-core-kalitimsal-durumlardaki-table-per-typetpt-ve-table-per-hierarchytph-davranis-modellemeleri/>

<https://www.gencayyildiz.com/blog/azure-storage-serisi-7-azure-blob-storage-ve-asp-net-core-ile-kullanimi/>

<https://www.gencayyildiz.com/blog/cqrs-pattern-nedir-mediatr-kutuphanesi-ile-nasil-uygulanir/>

<https://www.gencayyildiz.com/blog/asp-net-core-identity-yazi-dizisi/>

<https://tr.wikipedia.org/wiki/JSON_Web_Token#:~:text=JSON%20Web%20Token%20(JWT)%2C,olu%C5%9Fturabilir%20ve%20bunu%20kullan%C4%B1c%C4%B1ya%20g%C3%B6nderebilir.>

<https://www.gencayyildiz.com/blog/asp-net-core-identity-angular-14-esliginde-google-login/>

<https://www.gencayyildiz.com/blog/asp-net-core-identity-facebook-login/>

<https://github.com/serilog/serilog>

<https://www.gencayyildiz.com/blog/asp-net-core-signalr-yazi-serisi/>

<https://www.gencayyildiz.com/blog/asp-net-mail-islemleri/>

<https://github.com/codebude/QRCoder>

## **Frontend**

<https://www.gencayyildiz.com/blog/angularjs-modul-yapisi-nedir-ve-nasil-kullanilir/>

<https://update.angular.io/>

<https://getbootstrap.com/>

<https://material.angular.io/>

<https://alertifyjs.com/>

<https://www.npmjs.com/package/ngx-toastr>

<https://www.npmjs.com/package/ngx-spinner>

<https://www.npmjs.com/package/ngx-file-drop>

<https://www.npmjs.com/package/@abacritt/angularx-social-login>

<https://www.npmjs.com/package/ngx-signalr>

<https://www.npmjs.com/package/@angular/platform-browser-dynamic>

<https://www.npmjs.com/package/@techiediaries/ngx-qrcode>