🎯 Request-Response Pattern Nedir?

- İstemci (Client, örneğin tarayıcı) bir istek gönderir (Request).
- Sunucu (Server, yani WebAPI) bu isteği alır, işler ve bir cevap döner (Response).
- İstek ve cevap bir standart (JSON gibi) üzerinden iletişir.
- Özellikle Web API'lerde (REST API) bu desen çok kullanılır.

📝 Örnek:

- Client: "GET /api/users" der.
- Server: User listesini JSON olarak döner.

🎯 Neden IAsyncRepository<T> diye bir Interface yazdık?

- Çünkü **bütün veritabanı işlemlerimizin kurallarını** bir yerde tanımlamak istedik.
- Yani her Add, Delete, Update işlemi için **ortak bir sözleşme** oluşturduk.
- Başka bir sınıf (örneğin EfRepositoryBase) bu interface'i uygularsa, **bu kurallara uymak zorunda kalacak**.
- Böylece **standart ve düzenli** bir yapı oluşuyor. (Her entity için baştan Add yazmak zorunda kalmıyoruz.)

🎯 Neden EfRepositoryBase<T> diye bir Base Class yazdık?

- Çünkü **CRUD işlemleri** (ekle, sil, güncelle, listele) **her entity için aynıdır**.
- Yani User da eklenecek, Product da eklenecek — mantık aynı.
- Her entity için aynı kodu tekrar tekrar yazmamak için bir **base sınıf** yaptık.
- Artık **UserRepository**, **ProductRepository** gibi sınıflar sadece EfRepositoryBase'i miras alacak, ekstra uğraşmayacağız.

🎯 Neden hepsi async yazıldı?

- Çünkü **veritabanı işlemleri yavaş** olabilir.
- Asenkron (async) çalışarak uygulamayı **donmadan**, **hızlı** çalıştırmak istiyoruz.
- Bu da kullanıcı deneyimini artırıyor, performansı artırıyor.

📋 Çok basit bir özet:

Neden?	Açıklama
Interface	Ortak kuralları yazdık. (Standartlaşma)
Base Class	Tekrarlanan kodları tek yerde topladık. (Kod tekrarı yok)
Async	Uygulama donmasın, daha hızlı çalışsın diye.