# REST

## (Representation State Stransfer)

**Nedir?**

REST istemci ve sunucu arasında iletişim kurulmasını sağlayan bir servis yapısıdır. Servis yönelimi mimari üzerinedir ve HTTP üzerinde çalışan bir veri transferi yöntemidir.

**Nasıl Çalışır?**

Uygulamadaki haberleşmeleri sağlamak için, istemci-sunucu arasında XML ve JSON verilerini taşır. Durum bilgileri saklamaz. Yani ekstra başlık bilgilerini, istemci detaylarını taşımaz ve saklamaz.
REST servisler doğrudan URL çağırılarak çalışır. Çalışması için gerekli tek şey URL'dir.

**REST Mimarisinde Sınırlamalar;**

**Stateless;** Sadece ihtiyaç duyulduğu zaman, talep ile birlikte sunucuya bildirim gerçekleştirilir.

**Basit Arayüz;** Ortak arayüz sayesinde her parça bağımsız bir şekilde farklı evrimler geçirebiliyor.

**Code On Demand ;** Bu sınırlama ile birlikte sunucu bazı durumlarda kullanıcı tarafında daha fonksiyonel bir durum oluşturmak için executable script'ler göndermeyi seçebilir.

**Cach Edebilme;** HTTP üzerinden gelen response(etki), client tarafından cache edilebilir.

**Layered System;** Aracı sunucular ile birlikte balance değişiklikleri sağladığı için daha fazla ölçeklenebilirlik imkanı sunar.

**Client-Serer Mimarisi;** Client ve server arasında birbirinden bağımsız bir ilişki yer alır.

**Avantajlar;**

- Çok sade ve rahat kullanılabilir bir arayüzü bulunmaktadır. (Lightweight)
- Servis ajanlarınca yani hizmet vekilleri tarafından bileşenler arasındaki haberleşme görünürlüğü sağlamaktadır.
- Farklı ihtiyaçların doğru bir şekilde yerine getirilmesi amacıyla bileşenlerde değişiklikler gerçekleştirilebilir.
- Bir veri ile beraber program kodu hareket ettirildiğinde, istenilen bileşen kolaylıkla taşınabilir.
- Birçok farklı sayıda bileşende bileşenler arası etkileşim sayesinde de ölçeklenebilirlik elde edilebilir.
- Hataya karşı direnç güvenliği her zaman sistem seviyesindeki tüm veri, bağlantı veya bileşenlerde elde edilmektedir.

## RESTful

**Nedir;**

REST'e uygun olarak yazılan web servislerine RESTful servisleri denir.

**Nasıl Çalışır;**

RESTful servisler veri iletiminde farklı HTTP metotlarını kullanmaktadır (GET, POST, PUT, DELETE). HTTP talebi, çağırılan URL ile bu 5 metottan biri olarak seçilir ve sunucu sonucu buna göre belirler.

**HTTP Metotları;**

- GET, okuma işlemidir,
- POST, yeni bir kaynak oluşturur,
- PUT, kaynağı değiştirir veya var olan kaynağı günceller,
- DELETE, kayıt silme işlemi yapar,
- PATCH, ise bir kaynakta kısmi güncelleme yapmak için kullanılır.

**Avantajlar;**

Basit yapısı, kolay uygulanması, hızlı çalışması, esnek olması.

**Dezavantajlar;**

Güvenlik konusu geliştirilen yazılımın bir parçasıdır ve tokenler kullanılmaktadır.

**Kullanım Örnek;**

RESTful bir servisi çağırmak için karşınızdaki kurum size www.siteadi.com/product/12345 gibi bir URL verir ve bu adresi çağırdığınızda ID değeri 12345 olan ürünün detaylarının size JSON olarak döneceğini söyler. Size kalan tek iş bu URL'yi back-end'de WebRequest vb. sınıflarla veya client-side'da AJAX fonksiyonları vasıtasıyla çağırmak ve gelen JSON verisini uygun formatta görüntülemek olacaktır.

## DEPENDENCY INJECTIONS

**Nedir;**

Nesnenin bağımlı olduğu diğer nesneleri direkt olarak kullanmak yerine, bu nesnelerin dışarıdan verilmesiyle sistemdeki bağımlılığı azaltmaktır. Temelinde, oluşturacağımız bir sınıf içinde başka bir sınıfın nesnesini kullanacaksak, bunu "new" kelimesiyle oluşturmamamız gerektiğini söyleyen bir yaklaşımdır. Böylece bağımlılık bulunan sınıf üzerinde değişikliklerden korunmuş oluruz. Dependency Inversion problem çözmeye yarayan bir prensip iken Dependency Injection ise bu prensibi uygulayan bir Design Pattern'dir.

**Örnek;** Bir Staj sınıfı, rapor sınıfının bazı fonksiyonlarını kullanıyorsa, Staj Class'ı Rapor Class'ına bağımlıdır.

**Temel olarak üç tür bağımlılık enjeksiyonu vardır:**

***Yapıcı Enjeksiyonu:** Yapıcı enjeksiyonunda, enjektör hizmeti (bağımlılık) istemci sınıfı yapıcısı aracılığıyla sağlar.

***Özellik Enjeksiyonu:** Özellik enjeksiyonunda (diğer adıyla Setter Enjeksiyonu), enjektör, istemci sınıfının genel bir özelliği aracılığıyla bağımlılığı sağlar.

***Yöntem Enjeksiyonu:** Bu tür enjeksiyonda, istemci sınıfı, bağımlılığı sağlamak için yöntemleri bildiren bir arabirim uygular ve enjektör, istemci sınıfına bağımlılığı sağlamak için bu arabirimi kullanır.

**Bağımlılık enjeksiyonunun sorumlulukları şunlardır**:

- Nesneleri oluşturun.

- Hangi sınıfların bu nesneleri gerektirdiğini bilin.

- Ve onlara tüm bu nesneleri sağlayın.

**Avantajlar;**

- Nesnelerin yapımı ve kullanımı ile ilgili endişelerin ayrılmasını sağlamaktır. Bu, okunabilirliği ve kodun yeniden kullanımını artırabilir.

- Birim testinde yardımcı olur.

- Bağımlılıkların başlatılması enjektör bileşeni tarafından yapılır.

- Uygulama daha kolay hale gelir.

- Uygulama programlamasında önemli olan gevşek bağlantının sağlanmasına yardımcı olur.

**Dezavantajlar;**

-Öğrenmesi biraz karmaşıktır ve aşırı kullanılırsa yönetim sorunlarına ve başka sorunlara yol açabilir.

-Birçok derleme zamanı hatası çalışma zamanına itilir.

-Bağımlılık enjeksiyon çerçeveleri, yansıma veya dinamik programlama ile uygulanır. Bu, "referans bul", "çağrı hiyerarşisini göster" ve güvenli yeniden düzenleme gibi IDE otomasyonunun kullanımını engelleyebilir.

**Kullanım Alanları;**

- Kodlar esnek değildir. Rapor'da yapılacak bir değişiklik Staj üzerinde de değişikliğe neden olacaktır,
- Bağımlılıklar Unit Testi zorlaştırır. Bir sınıfın işlevlerini test etmek için diğerlerini de test etmek gerekir,
- Bağımlılık fazlaysa bir işlevi başka bir sınıfın yeniden kullanması zorlaşır. Çünkü bağımlı sınıflar birbirleri için geliştirilmiş olmalıdırlar,

gibi sorunlar Dependency Injection(DI) ile çözülebilinmektedir.

## HEXAGONAL ARCHITECTURE

**Nedir;**

Altıgen mimari veya bağlantı noktaları ve adaptörler mimarisi, yazılım tasarımında kullanılan bir mimari modeldir. Bağlantı noktaları ve adaptörler aracılığıyla yazılım ortamlarına kolayca bağlanabilen gevşek bağlı uygulama bileşenleri oluşturmayı amaçlar. Bu, bileşenleri her düzeyde değiştirilebilir hale getirir ve test otomasyonunu kolaylaştırır. Bu yaklaşım, iş katmanını çerçeve, kullanıcı arayüzü, veritabanı veya diğer harici bileşenlerden bağımsız hale getirmemize yardımcı olur.

Altıgen mimari, bir sistemi değiştirilebilir birkaç bileşene böler. Her bileşen, bir dizi açık "port" aracılığıyla diğerlerine bağlanır. Bu bağlantı noktaları aracılığıyla iletişim, amaçlarına bağlı olarak belirli bir protokolü takip eder. Bağlantı noktaları ve protokoller , herhangi bir uygun teknik yolla uygulanabilen soyut bir API tanımlar.

**Altıgen mimari üç ilke ve tekniğe dayanmaktadır:**

- Bunlar Kullanıcı Tarafı, İş Mantığı ve Sunucu Tarafıdır.
- Bağımlılıklar, Kullanıcı Tarafı ve Sunucu Tarafından, İş Mantığına doğru gitmektedir.
- Bağlantı Noktaları ve Adaptörler kullanarak sınırları izole edebiliriz.

**Bağlantı noktalarının ayrıntı düzeyi ve sayıları sınırlandırılmamıştır:**

- Yeterli olacaksa tek bir bağlantı noktası kullanılabilir,
- Olay kaynakları, veritabanı ve yönetim için bağlantı noktaları vardır,
- Gerekirse her durum için farklı bir bağlantı noktası kullanılabilir.

Kullanıcı Tarafı (User-Side), İş Mantığı (Business Logic) ve Sunucu Tarafı (Server Side) şeklinde soldan sağa doğru gidilen bir grafik düşünüldüğünde;

**KULLANICI TARAFI(SOL TARAF);**

Kullanıcı tarafı, kullanıcının ya da programın uygulama ile etkileşimde gireceği taraftır. Etkileşimlere izin veren kodu içermektedir (kullanıcı arabirim kodu, API için HTTP'ler).

**İŞ MANTIĞI MERKEZİ(ORTA TARAF);**

Bu kısım iki tarafa da izole olmalıdır. İş mantığını ilgilendiren tüm kodları içerir ve uygular. Sorunları çözen, uygulamayı zengin kılan kısımdır. (Kodlamayı bilmeyen bir alan uzmanı bu bölümdeki bir kod parçasını okuyabilir ve sizi bir tutarsızlığa işaret edebilir.)

**SUNUCU TARAFI(SAĞ TARAF)**

Uygulamanın neye ihtiyaç duyduğunu, neyin işe yaradığını bu kısım içerir. Veritabanıyla etkileşime giren, dosya sistemine çağrıda bulunan, HTTP çağrılarını işleyen kod gibi temel altyapı ayrıntılarını içerir.

Bu ayrımlar ile sorunları tek bir tarafa bakarak çözebiliriz. Ayrıca bu şekilde iş mantığı kodlarını ön plana çıkarılmaktadır. Bu kodlar tüm geliştiricilere açık hale getirmek için bir dizinde veya modülde izole edilebilir. Programın geri kalanının bilişsel yükünü yüklenmeden tanımlanabilir, iyileştirilebilir ve test edilebilir.

**Etki Alanı Nesnesi:**

Çekirdek etki alanı modelini temsil eder ve uygulamanın çekirdeğidir. Devlet ve iş davranışlarına sahip olabilir. Etki alanı nesnesinin diğer bileşenlere herhangi bir bağımlılığı yoktur.

**Giriş ve Çıkış Portları:**

Uygulama çekirdeği, dış dünyayla iletişim kurmak için "bağlantı noktaları" adı verilen özel arabirimler kullanır. Uygulamaya veri girişine veya uygulamadan veri çıkışına izin verirler.

- Bir giriş bağlantı noktası ( sürücü bağlantı noktası ), uygulama çekirdeğinin işlevselliği dünyanın dışına göstermesini sağlar.

- Çıkış bağlantı noktası ( yönlendirilen bağlantı noktası ), uygulama çekirdeği tarafından kendi dışındaki şeylere ulaşmak için kullanılan başka bir arabirim türüdür (veritabanından bazı verileri almak gibi).

**Adaptörler;**

İki tip adaptör vardır;

Birincil bağdaştırıcılar, kullanım senaryolarının yürütülmesini tetiklemek için giriş bağlantı noktalarını kullanır.

**Örneğin:** Bir mobil uygulamada, bir kullanıcı bir düğmeye tıkladığında, ilgili adaptör, ilgili kullanım senaryosunu istemek için giriş adaptörünü çağırmalıdır.

İkincil adaptörler, kullanım durumları tarafından çağrılır.

**Örneğin:** Bir veritabanından belirli bir veriye erişmek için kullanım durumu tarafından ikincil bir bağdaştırıcı kullanılabilir.

**Uygulama;**

Uygulama, sistemin özüdür, işlevselliği veya kullanım durumlarını düzenleyen Uygulama Hizmetlerini içerir. Bağlantı Noktalarından komutlar veya sorgular alan ve ayrıca veritabanları gibi diğer harici aktörlere de Bağlantı Noktaları aracılığıyla istekler gönderen bir altıgen ile temsil edilir.

**Avantajlar;**

- Uygulamayı daha test edilebilir, yönetilebilir ve bakımı kolay hale getirecektir.
- Uygulamanın temel mantığını dış kaygılardan ayırmaya izin verir.
- Diğer sistemlere herhangi bir bağımlılık olmaksızın iş mantığını test edilebilir hale getirir.( Bu, iş kurallarının UI, Veritabanı, Web Sunucusu veya başka herhangi bir harici öğe olmadan test edilebileceği anlamına gelir.)
- Adaptörlerde kolayca değişiklik yapma esnekliği sağlar