İçindekiler Tablosu

[**HTTP Metotları** 2](#_Toc163580513)

[1. Idempotent Metot 2](#_Toc163580514)

[2. GET 2](#_Toc163580515)

[3. POST 2](#_Toc163580516)

[4. PUT 2](#_Toc163580517)

[5. DELETE 2](#_Toc163580518)

[6. PATCH 2](#_Toc163580519)

[7. OPTIONS 2](#_Toc163580520)

[8. HEAD 2](#_Toc163580521)

[9. TRACE 2](#_Toc163580522)

[10. SEARCH 2](#_Toc163580523)

[11. CONNECT 2](#_Toc163580524)

[**API** 3](#_Toc163580525)

[REST 3](#_Toc163580526)

[REST Tasarım İlkeleri 3](#_Toc163580527)

[Client (İstemci) 3](#_Toc163580528)

[Server (Sunucu) 3](#_Toc163580529)

[Resource (Kaynak) 3](#_Toc163580530)

[**Stateless (Statüsüz)** 3](#_Toc163580531)

[**Uniform Interface (Tekdüze Arayüz)** 3](#_Toc163580532)

[**Cacheable (Önbelleklenebilir)** 3](#_Toc163580533)

[**Client-Server** 3](#_Toc163580534)

[**Layered System (Katmanlı Sistem)** 3](#_Toc163580535)

[**Code On Demand (Talep Üzerine Kod)** 3](#_Toc163580536)

[REST – RESTful Arasındaki Fark 4](#_Toc163580537)

# **HTTP Metotları**

Veri iletimi için kullanılan **http,** protokollerinden bir tanesidir. WEB tarayıcılar ve sunucular arasındaki iletişime yönelik bazı metotları bulunmaktadır.

## Idempotent Metot

Bir isteğin birden fazla kez gerçekleşmesi sonucu değiştirmemesi prensibine dayanır. HTTP metotlarında birden fazla kez gönderilen isteğin kaynak üzerinde bir değişikliğe sebebiyet vermemesi durumudur.

## GET

Belirtilen bir URI (Uniform Resource Identifier) -Tekdüzen Kaynak Tanımlayıcısı- üzerinden bilgi almak için kullanılan en yayın yöntemlerden biridir. Sunucudan veriyi çekmek için gönderilen tarayıcı isteğidir. Genellikle URL ile birlikte kullanılır ve sorgu metinleri URL içinde gönderilebilir.

metin, ekran görüntüsü, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

GET metotu bir kaynağı alma isteğidir. idempotent metottur. Aynı kaynağı birden fazla almak kaynak üzerinde bir değişikliğe sebep olmaz.

## POST

Sunucuya yeni bir veri kaynağı yazmada kullanılır. Genellikle HTTP body kısmında gönderilir. Bu gönderim ile genellikle form gönderimi ve veri tabanı işlemlerine uygundur. Güvenlik olarak daha sıkı bir metottur.

## PUT

Belirtilen URL üzerindeki kaynağın bir kısmını veya tamamını güncellemek için kullanılabilir. Yani sunucu tarafından kaynağın tamamen değiştirilmesi anlamına gelir. RESTful API’de kullanılabilir. Güncellenecek kaynağın ID’si gönderilmek zorundadır. Idempotent metottur. Belirtilen kaynak her zaman istenilen duruma gelerek değişmezliğini koruyacaktır.

## DELETE

Belirtilen kaynağı kaldırmak için kullanılır. Idempotent metottur. Belirtilen kaynak silinmişse metot kaynak üzerindeki etkisi aynı olarak kalacaktır.

## PATCH

Belirtilen kaynağın bir kısmını değiştirmeyi sağlar. PUT ile farklı olarak sadece değişim istenen alanlar gönderilir, tamamı değiştirilmez.

## OPTIONS

Sunucunun belirli bir kaynağa erişim izinlerini ve desteklediği HTTP metotlarını bildirmek için kullanılır. İstemcinin kaynağa ulaşmadan önce hangi HTTP metotlarını kullanabileceğini ve ek özel başlıkları kullanabileceği sorgulanır. CORS uygulamalarında sıklıkla kullanılır.

CORS uygulamaları web tarayıcıları arasında güvenli bir şekilde kaynak paylaşımını sağlayan mekanizmadır. İki farklı web sayfası arasında tarayıcı güvenlik politikaları nedeniyle doğrudan kaynak paylaşımı engellenebilir. CORS ile bu engel kaldırılabilir ve site tarafından izin verilebilir.

## HEAD

Belirtilen bir kaynağın sadece header kısmını alıp gerçek içeriği almaz. GET metodu ile aynı özellikleri taşır fakat GET metodundan önce kaynağın varlığını kontrol eder.

## TRACE

Sunucuya gelen istek mesajının içeriği kontrol edilir. Aralarında başka bir sunucu var mı? Var ise mesaj içeriğinde nasıl değişiklikler yaptı? Sorularını cevaplar. Teşhis metodudur.

## SEARCH

Bir dizinin altındaki kaynakları sorgulama işleminde kullanılır.

## CONNECT

Bir Proxy sunucusu üzerinden başka bir sunucuya bağlanmada ve Proxy sunucusunu tünel olarak kullanmada kullanılır.

# **API**

API (Application Programming Interface) uygulamaların veya cihazların birbirine nasıl bağlanacağını ve birbirleriyle nasıl iletişim kurabileceğini tanımlayan bir dizi kuraldır.

API entegrasyonu veri işlemleri ve ortak işlev gerçekleştirmek için API’leri aracılığı ile birbirine bağlanan ve iki veya daha fazla uygulama arasında etkileşimi sağlamayı ifade eder.

## REST

REST bir sistemde çalışmak için gerekli olan tek şey URL’dir. Bir URL’e istek atıldığında geri dönüş olarak JSON veya XML formatında cevap alınır. Bu cevap parse edilir ve entegrasyon tamamlanır.

Client uygulama, REST bir servisin detaylarını ve yapısını bilmek zorunda değildir. REST servisler, client-server arasındaki ayrımı baz alır. Bu sayede farklı sunucular ve clientler ile iletişim kurmayı kolaylaştırır.

## REST Tasarım İlkeleri

REST API’nın tasarım ilkeleri kısıttan daha fazla olarak REST mimarisinin hangi sınırlar arasında olduğuna yönelik ilkelerdir.

### Client (İstemci)

API’yi kullanan yazılım veya kişidir. Bir sayfanın API’sine istek atarak oradan istenilen verileri alan ve yansıtan uygulamalara denir.

### Server (Sunucu)

Tüm API ve işlevsellikleri üzerinde barındıran sistemdir. İstemciden gelen istekleri işlemek ve istenilen formatta döndürmekle sorumludur.

### Resource (Kaynak)

Kaynak, API’nin herhangi bir bilgi sağlayacağı nesnedir. Bu nesneler belirsiz bir tanımlayıcısı (Unique Identifier) veya numarası vardır.

### **Stateless (Statüsüz)**

Sunucunun istekler hakkında bilgi tutmamasıdır. Sunucu istek gönderen client’in kaç istek veya neler istediği durumları hakkında bilgileri tutmaz. Client ise sunucunun ihtiyaç duyduğu tüm bilgileri verir.

Sunucu, istemci bilgileri tutmadığı için her istekte istemcinin bilgileri vermesi maliyeti arttırır.

### **Uniform Interface (Tekdüze Arayüz)**

Client, kaynakları tanımlamak ve etkileşim kurmak için genel bir arayüz kullanmalıdır. Bu, servislerin değişikliklere karşı daha esnek ve ölçeklenebilir olmasını sağlar.

### **Cacheable (Önbelleklenebilir)**

Durum bilgisine sahip olmayan bir API, gelen giden istekleri yöneterek istek yükünü arttırabilir. Bunun önüne geçebilmek için önbelleğe isteklerin alınmasıyla daha sonra gelen benzer isteklerin yanıtlanması sağlanır. Bu ilkeye göre yanıttaki veriler dolaylı veya direkt olarak **cacheable** veya **uncacheable** olarak tanımlanır.

### **Client-Server**

Client ve server bağımsız hareket eder. Server sadece clientten istek gelmesini bekler. Kendi başına bilgi gönderimi yapmaz. Bunun sonucunda client ve server ayrı olarak geliştirilir.

Server tarafından geliştirme bağımsız ve ölçeklenebilirlik, client tarafında ise kodun taşınılabilirliği yüksek olur.

### **Layered System (Katmanlı Sistem)**

Client-System mimarisinde bahsedilen doğrudan iletişim olması değil, arada güvenlik katmanları oluşturulmasıdır. Bu katmanlar **Request-Response** etki etmemelidir. Her katman iletişime geçtiği katmanı bilmelidir.

Ağdaki katmanlı yapı sonucunda, sistemin ölçeklenebilirliği ve esnekliğini arttırır.

### **Code On Demand (Talep Üzerine Kod)**

Client, serverdan dinamik olarak kod alabilmesini sağlar. İlke sonucunda, sunucudan istemciye belirli bir bölümün işlenebilmesi için gerekli kod parçacığı veya senaryoları gönderimine izin verir.

Bu ilke, istemcilerin daha az kod ile daha zengin bir içerik içermesini sağlar. Ancak RESTful servisler genellikle statik kaynaklar döndürür ve stateless olması gerektiği için bu prensip çok yayın olarak kullanılmaz. RESTful servislerde ek işlevsellik veya özel durumlar için kullanılır.

## REST – RESTful Arasındaki Fark

REST, web tabanlı bir tasarım mimarisidir. Kaynaklara benzersiz URI’ler ile erişime ve kaynakların temsilini kullanarak etkileşimde bulunma fikridir. Bu etkileşimler genellikle **HTTP** metotları kullanılarak gerçekleştirilir.

RESTful, bir uygulamanın veya servisin REST prensiplerini takip etmesi durumudur.

Kaynaklara benzersiz URI’ler üzerinden erişilebilir, HTTP metotlarının kullanıldığı, temsilin istemcilerle paylaşıldığı ve sunucunun durum bilgisinin **saklanmadığı** anlamına gelir.