**API Nedir? REST API Kavramı**

Öncelikle ‘API nedir, nasıl çalışır?’ tanımlayacak olursak; ***bir API veya uygulama programlama arabirimi (Application Programming Interface), uygulamaların veya cihazların birbirine nasıl bağlanabileceğini ve birbirleriyle iletişim kurabileceğini tanımlayan bir dizi kuraldır.*** API entegrasyonu, veri alışverişi yapmak ve ortak bir işlev gerçekleştirmek için API’leri aracılığıyla birbirine bağlanan ve böylece uygulamalar arasında etkileşimi sağlayan birkaç uygulamayı (iki veya daha fazla) ifade eder.

**REST, client-server arasındaki haberleşmeyi sağlayan HTTP protokolü üzerinden çalışan bir mimaridir. İstemci ve sunucu arasında XML ve JSON verilerini taşıyarak uygulamanın haberleşmesini sağlar. REST mimarisini kullanan servislere ise RESTful servis (RESTful API) denir.**

*Amazon, Google, Facebook, LinkedIn ve Twitter* gibi çeşitli web siteleri, kullanıcıların bu bulut hizmetleriyle iletişim kurmasını sağlayan REST tabanlı API’leri kullanır.

REST ile yazılmış bir servisle çalışmak için *ihtiyacımız olan tek şey URL*. *Bir URL’e istek attığımızda, URL size JSON veya XML formatında bir cevap döndürür, dönen cevap parse edilir ve servis entegrasyonunuz tamamlanır.* Yani client uygulama, REST bir servisin yapısını ve detaylarını bilmek zorunda değildir. Rest servisler; client ve server arasındaki ayrım sayesinde, REST protokolü, bir projenin farklı alanlarındaki geliştirmelerin bağımsız olarak gerçekleşmesini kolaylaştırır.

REST API, operasyonel sözdizimine ve platforma göre ayarlanabilir ve geliştirme sırasında çok sayıda ortamı test etme olanağı sunar. Kullanıcılar, REST client-server farklı sunucularda barındırılsa bile kolayca iletişim kurabilir , bu da yönetim açısından önemli bir fayda sağlar.

**REST TASARIM İLKELERİ**

RESTful API’ler oluşturmak için dikkat edilmesi gereken kısıtlama ve kurallara geçmeden önce, 3 temel kavramdan bahsedelim:

*1.****Client(İstemci)*:** Client, API’yi kullanan kişi veya yazılımdır. Bir geliştirici olabilir, örneğin bir geliştirici olarak siz, Twitter API’sini Twitter’dan veri okumak ve yazmak, yeni bir tweet oluşturmak ve yazdığınız bir programda daha fazla işlem yapmak için kullanabilirsiniz. Programınız Twitter’ın API’sine istek atacak. İstemci ayrıca bir web tarayıcısı olabilir. Twitter web sitesine gittiğinizde, tarayıcınız Twitter API’sine istek atan ve döndürülen verileri ekranda bilgi işlemek için kullanan istemcidir.

*2.****Server (Sunucu)*:**Tüm API ve işlevsellikleri üzerinde barındıran sistemdir. Client’den gelen istekleri işlemek ve gerekli cevapları yine istenen formatta dönmekle sorumludur.

*3.****Resource (Kaynak):*** Bir kaynak, API’nin bilgi sağlayabileceği herhangi bir nesne olabilir. Örneğin; *Instagram’ın API’sinde bir kaynak: kullanıcı, fotoğraf ya da hashtag olabilir.* Her kaynağın benzersiz bir tanımlayıcısı (unique identifier) vardır. Tanımlayıcı bir isim veya bir numara olabilir.

Rest Api’nin tasarım ilkelerine gelecek olursak, aslında bunlara kısıttan daha çok REST mimarisinin hangi sınırlar içerisinde yer alması gerektiğini belirleyen prensipler diyebiliriz.

**Stateless**

*REST’in stateless olması server’ın client hakkında****session gibi bilgileri tutmaması*** *demektir.* Bu gibi bilgileri yalnızca *client*tutar. Dolayısıyla *server,*istek yapan*client*’ındaha önce kaç istek yaptığı veya hangi istekleri yaptığı gibi bilgileri tutmaz. *Client*ise yaptığı istekte *server*’ın ihtiyaç duyduğu tüm bilgileri verir.

Fakat aynı zamanda *server*, *client*’a ilişkin veri tutmadığı için *client*’ın ***her istekte bazı bilgileri göndermesi maliyeti artırır***. Bu da *stateless*oluşunun **dezavantajı** olarak sayılabilir.

**Uniform Interface**

Aynı kaynak için tüm API istekleri, isteğin nereden geldiğine bakılmaksızın aynı görünmelidir. REST API, bir kullanıcının adı veya e-posta adresi gibi ***aynı veri parçasının yalnızca tek bir tek tip kaynak tanımlayıcısına (URI) ait olmasını sağlamalıdır***. Kaynaklar çok büyük olmamalı, ancak karşı tarafın ihtiyaç duyabileceği her türlü bilgiyi içermelidir.

**Cacheable**

Durum bilgisi olmayan bir API, çok sayıda gelen ve giden çağrıyı yöneterek istek yükünü artırabileceğinden, bir REST API tasarımı önbelleğe alınabilir verileri depolayabilmelidir. Bu API tasarım ilkesine göre, bir yanıttaki veriler dolaylı veya açık bir şekilde cacheable veya uncacheable olarak sınıflandırılmalıdır.

Bir yanıt önbelleğe alınabilirse, istemci önbelleğine gelecekte benzer istekler için bu yanıt verilerini geri dönüştürme hakkı verilir.

**Client-Server**

Client ve server bağımsız hareket eder, her biri kendi başına ve aralarındaki etkileşim yalnızca client tarafından başlatılan istekler ve server’ın yalnızca bir isteğe tepki olarak client’a gönderdiği yanıtlar biçimindedir. *Server yalnızca client’dan gelen istekleri bekler*. Server, bazı kaynakların durumu hakkında kendi başına bilgi göndermeye başlamaz. Bunun sonucunda *client* ve *server*birbirinden bağımsız olarak geliştirilir, *server*tarafında geliştirme basit ve ölçeklenebilirlik yüksek olur ve *client*tarafında ise kodun taşınabilirliği yüksek olur.

**Layered System**

*Client-Server*mimarisinden bahsederken kastımız her zaman bir *client*’ın doğrudan bir *server*’a istek göndermesi ve ondan doğrudan cevap alması şeklinde değildir. *Aralarda güvenlik katmanı, cache katmanı gibi katmanlar olabilir. Böyle bir sistemde aralardaki katmanlar request ve response’a etki etmemeli. Her katman yalnızca iletişime geçtiği katmanları bilmeli.*

**Code On Demand**

Zorunlu olmayan tek kısıt *code on demand*’dir.

Client, serverdan kod talep edebilir ve daha sonra, yanıt HTML biçiminde olduğunda, serverden gelen yanıt, genellikle bir komut dosyası biçiminde bazı kodlar içerecektir. Client daha sonra bu kodu çalıştırabilir.

**Rest Api Nasıl Çalışır?**

Bir Api geliştirmek için aşağıdaki tasarım modellerinden birini izlemelisiniz:

***· Rest***

***· SOAP***

***· gRPC***

***· XML-RPC***

Genellikle modern yazılım geliştirme süreçlerinde REST kullanılmaktadır. gRPC ise yeni yeni hayatımıza giren bir teknolojidir.

REST mimarisinde işlemler resource kavramıyla yapılır. Resource URI ile tanımlanır ve bir metod tanımlaması veya bir değişken olabilir. Oluşturulan metodu çağıracak URIler ile web servise HTTP protokolüyle istek yapılır. Böylece REST için işlemler tamamen HTTP metodları üzerinden yapılır. Örneğin, satışta olan bir ürüne ait tüm bilgileri çekmek için “*/product/15*” URI’sine bir GET isteği atmamız yeterli olacaktır. Burada ‘15’ rakamı ilgilini ürünün benzersiz tanımlayıcısıdır. Ayrıca REST **JSON**, **XML**, **TXT**, **HTML** gibi istenen veri türünde değer döndürülebilir. Fakat genellikle **JSON**tercih edilmektedir.

REST’de iki tip URI vardır. Bunlardan biri *Collection* URI diğeri *Element* URI’dir.

·**Collection URI**; array, list gibi veri yapıları için kullanılır. Örnek kullanım şekli; *http://example.com/resources*

**· Element URI**ise değişkenler üzerinden işlem yapmak için kullanılır. Örnek kullanım şekli; *http://example.com/resources/item/17*

**REST’te HTTP Metotlarının Kullanımı**

REST tabanlı web servislerde HTTP metodlarına özel anlamlar yüklenir ve böylece web servise bir HTTP isteği geldiği anda metod çalıştırılmış olur. Bu durumda HTTP metodlarının REST ile nasıl kullanılacağı önemlidir. Şimdi REST’te HTTP metotlarının kullanımıyla alakalı kurallara geçelim.

Test etmek amacıyla birçok ücretsiz test araçları bulunmakta, bunlardan biri testini gerçekleştirebilirsiniz. (***Postman***, Swagger UI, Insomnia gibi )

**GET**

GET metodu URI’da belirtilen *resource*’u getirmek için kullanılır. Bu istekte *body*olmaz. Yalnızca *header*olur.



*Header* veya *body*eklemeden gönderdiğimiz isteğe cevap olarak aşağıdaki JSON *representation’*ı alıyoruz.



**HEAD**

HEAD metodunun GET metodundan tek farkı *response*’ta *body*göndermemesidir. Bu isteği, *response*’ta gönderilecek olan *header*’lar için yaparız. Örneğin bir *resource*’un belirtilen URI’da olup olmadığını kontrol etmek için veya *resource*’un en son ne zaman değiştirildiğini görmek için kullanılabilir *(Last-Modified)*.

GET metodu ile istek gönderdiğimiz adrese şimdi de HEAD ile istek gönderelim:



HEAD metodu ile istek gönderdiğimiz için gelen *response’*ta *body*olmayacak. Fakat ***header***’ları görebiliriz.



**POST**

POST metodunu istek gönderdiğimiz *server*’da yeni bir *resource*oluşturmak için, *controller resource*’ları çalıştırmak için ve *form input*’larını göndermek için kullanırız. Tüm bu işlemler için veri göndereceğimiz için POST metodunu kullanırken *body*de göndeririz.

Yine aynı sitede yeni bir post oluşturmak için POST isteği ile *title* ve *body*değişkenlerini gönderiyoruz.



Cevap olarak ise oluşturduğumuz Resource’un bilgilerini alıyoruz.



**PUT**

PUT metodu, POST metoduna benzeyen bir kullanım alanına sahiptir. *Body* ile bir *resource*gönderilir ve eğer URI var olan bir *resource*’a aitse o *resource g*üncellenir.

**DELETE**

Adından da anlaşılacağı üzere URI’da belirtilen *resource*’u silmek için gönderilen istektir. Yalnızca ilgili *resource*’un silinmesi ve bir daha hiçbir şekilde ulaşılmaması için kullanılır. Eğer *soft-delete* yapılacaksa bunun için *application-spesific*bir *controller* tanımlanır.

**OPTIONS**

İstek gönderdiğimiz URI’ın hangi metotları desteklediğini öğrenmek için kullanırız. Bu bilgi bize *header*’da *allow*ile verilir.

**PATCH**

PATCH metodu PUT’a benzer fakat *resource*’da kısmı güncelleme yapılacaksa kullanılır. Örneğin bir blog postunun yalnızca başlığı güncellenecekse PATCH metodu ile *body*’de başlığı göndermek maliyet açısından daha doğru bir tercih olur.

**Response Code’lar**

RFC 2616'da belirlenmiş olan *status code*’lar gönderilen isteğin durumunu *client*’a standart bir şekilde ifade eder. Eğer bir hata varsa hatayı, gönderilen istek başarılı bir şekilde işlem gördüyse ona ilişkin durumu aktarır. *Status code*’lar 5 kategoriye ayrılmıştır.

***· 1xx Bilgi***

***· 2xx Başarılı***

***· 3xx Yönlendirme***

***· 4xx Client Hatası***

***· 5xx Server Hatası***

Şimdi bu kategorilerde en çok kullanılan *status code*’ları inceleyelim.

**· 200 (OK) :**GET, PUT veya POST işleminde başarılı oldu. Başarılı bir yanıt için döndürüldü

**· 201 (Created) :**PUT veya POST işleminde başarı. Nesne başarıyla oluşturuldu veya güncelleştirildi.

**· 204 (No Content) :**PUT veya POST işleminde başarı. Dizin veya belgeler başarıyla karşıya yüklendi.

**· 400 (Bad Request ): i**stek URI ‘SI, üst bilgiler veya gövdede bir hata olduğunda döndürülür.

**· 401 (Unauthorized ):**Korumalı bir *resource*’a gerekli *authorization*sağlanmadan erişilmeye çalışıldığını ifade eder.

**· 403 (Forbidden ):**Geçersiz bir API anahtarı geçirdiğinizde döndürüldü.

**· 404 (Not Found ):***Client*’ın istediği *resource*’un bulunamadığını ifade eder.

**· 405 (Method Not Allowed ):**İstek yapılan URI’ın ilgili metodu desteklemediğini belirtir

**· 406 (Not Acceptable):** *Server*’ın, *client*’ın gönderdiği istekte *header*’*da Accept*alanında yazdığı medya tiplerinden herhangi bir formatta çıktı veremediğini belirtir.

**· 409 (Conflict ):**Yazma işlemleri çakışması olduğunda döndürülür.

**· 415 (Unsupported Media Type ):** *Server’*a, desteklemediği tipte bir *resource*gönderildiğinde 415 döner.

**· 500 (Internal Server Error ):** *Server*’ın hatası yüzünden istek yerine getirilemiyorsa kullanılır.