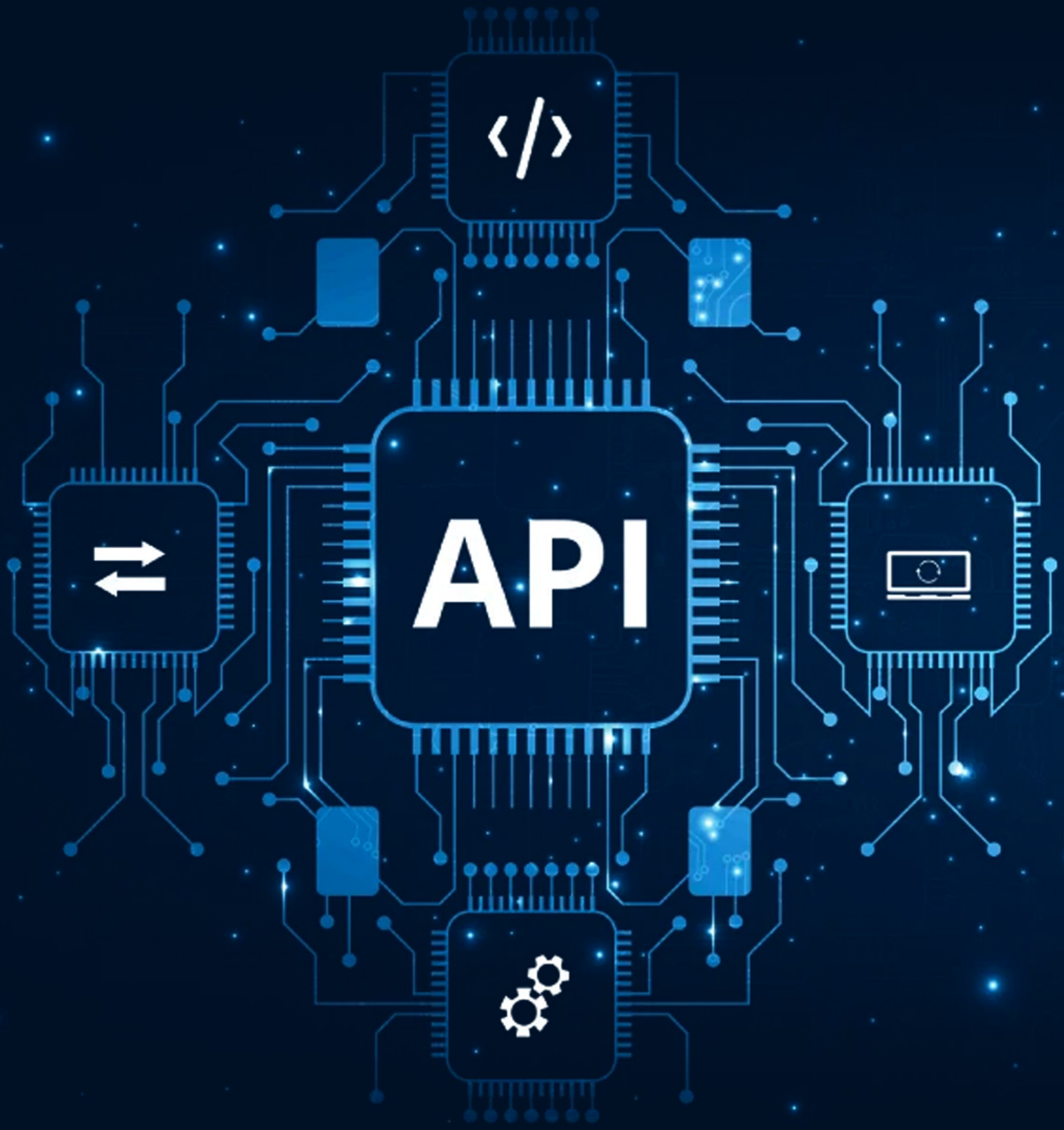
TÜBİTAK BİLGEM YTE | YAZILIM TEKNOLOJİLERİ ARAŞTIRMA ENSTİTÜSÜ

REST API VE gRPC MİMARİ STİLLERİNİN KARŞILAŞTIRILMASI

SAYI: 06



ARAŞTIRMA SERİSİ

Simge ve Kısaltmalar

Kısaltmalar	Açıklama
TÜBİTAK	Türkiye Bilimsel ve Teknolojik Araştırma Kurumu
BİLGEM	Bilişim ve Bilgi Güvenliği İleri Teknolojiler Araştırma Merkezi
YTE	Yazılım Teknolojileri Araştırma Enstitüsü
REST	Representational State Transfer (Temsili Durum Aktarımı)
HTTP	Hypertext Transfer Protocol (Metin Aktarım İletişim Protokolü)
gRPC	Google Remote Procedure Call (Google Uzaktan Prosedür Çağrısı)
API	Application Programming Interface (Uygulama Programlama Arayüzü)
URI	Uniform Resource Identifier (Tekdüzen Kaynak Tanımlayıcısı)
IDL	Interface Definition Language (Arayüz Tanımlama Dili)
TCP	Transmission Control Protocol (Gönderim Kontrol Protokolü)
IoT	Internet of Things (Nesnelerin İnterneti)

Yazar
Hüseyin ÇAMBAŞI

Yayın Koordinatörü
Elif ŞENYİĞİT

Editörler
Furkan TÜRK
Sevinç KARAKAŞ
Tuğçe YILMAZ

Tasarım
Şeyma KOÇER

İletişim: 0(312) 289 92 22 - yte.bilgi@tubitak.gov.tr

yte.bilgem.tubitak.gov.tr

İçindekiler

Önsöz

TÜBİTAK BİLGEM Yazılım Teknolojileri Araştırma Enstitüsü, 2012 yılından bu yana yazılım teknolojilerinde AR-GE faaliyetleri yürüten bir araştırma kuruluşudur. Araştırma faaliyetlerinde elde ettiği birikimini stratejik, hassas ve kritik projeler yürüterek kamu adına hayata geçirmekte; kurumlarımıza dijital dönüşüm, yazılım geliştirme teknolojileri ve kalite süreçleri konusunda danışmanlık vermektedir.

Araştırma Serisi ile TÜBİTAK BİLGEM YTE kurum içi çalışmaların yaygınlaştırılması ve sektörün erişimine açılması amaçlanmaktadır. Araştırma Serisi'nde yayınlanan çalışmalar TÜBİTAK BİLGEM YTE çalışanlarının projelerde elde ettiği bilgi birikimini paylaşmak adına derlenmiştir. Bu çalışmalar ile ülkemizin yazılım sektörüne katkı sağlanması hedeflenmektedir.

Giriş

gRPC ve REST API, modern uygulama geliştirme süreçlerinde yaygın olarak kullanılan iki farklı mimari stildir. Google tarafından geliştirilen gRPC, performans ve verimlilik odaklı bir yapı sunarken, REST API daha genel, kullanılabilirlik ve basitlik üzerine kurulmuştur. gRPC, Protobuf adı verilen serileştirme formatını kullanarak verileri binary formatına dönüştürür ve düşük bellek kullanımı ile yüksek hızlı iletişim sağlar. REST API ise genellikle JSON formatını kullanarak verilerin aktarımını gerçekleştirir.

gRPC'nin belirgin avantajlarından biri, güçlü bir tür denetimi sunmasıdır. Protobuf, veri yapılarını tanımlamak ve bu yapılar üzerinde sıkı denetim sağlamak için kullanılır. Bu, veri bütünlüğünü ve uyumluluğunu arttırır ve hataların erken tespit edilmesine olanak sağlar. Ayrıca gRPC, HTTP 2.0 tabanlı olduğu için tek bir bağlantı üzerinde birden fazla isteği aynı anda gönderebilir, istemci ve sunucu arasında verimli bir şekilde iletişim kurabilir.

REST API ise esneklik ve yaygın kullanılabilirlik konularında daha avantajlıdır. REST, HTTP protokolünü temel alır ve temel HTTP metotlarını kullanarak veri manipülasyonunu sağlar. Bu, farklı platformlar ve diller arasında kolay entegrasyon sağlar ve daha hafif istemci yapılarının kullanılmasına olanak verir. REST API'ler genellikle açık standartlara dayandığı için dokümantasyon, test etme ve anlaşılma konularında kolaylık sağlarlar.

Sonuç olarak gRPC ve REST API mimari stilleri, farklı avantajları ve kullanım senaryoları ile birlikte gelir. gRPC, performans ve tür denetimi gerektiren durumlarda tercih edilirken, REST API daha genel ve esnek kullanım gerektiren durumlarda daha uygundur. Bu çalışmanın amacı uygulama ihtiyaçları, performans, veri bütünlüğü, platform bağımsızlığı gibi faktörler göz önünde bulundurularak doğru mimari stilin seçilmesinde okuyucuya rehberlik etmektir.

REST API

REST (Representational State Transfer), API tasarlamak için kullanılan belirli kısıtlamaları içeren modern bir mimari yaklaşımdır. Günümüzde web ve mikroservis tabanlı uygulamaların birbirleri ile etkileşimlerinde kullanılan en popüler mimari stildir. REST, başlangıçta internet gibi karmaşık bir ağdaki iletişimi yönetmek için bir kılavuz olarak oluşturulmuştur. REST tabanlı mimari yüksek performanslı, güvenilir ve ölçeklenebilir API'ler oluşturmak için kullanılmaktadır. Kolay uygulayabilir, değiştirebilir yapısı, API'lerin sezgisel olarak kolaylıkla anlaşılabilir olmasını ve platformlar arası kolaylıkla taşınabilir olmasını sağlamaktadır. RESTful API'ler, durum bilgisiz, ölçeklenebilir ve güvenilir API'ler oluşturmak için modern web geliştirmede yaygın olarak kullanılmaktadır.

REST kısıtlamalarına uyan sistemler RESTful olarak isimlendirilir. RESTful API'ler HTTP protokolü üzerine kuruludur. Kaynaklara erişmek için bir URI (Uniform Resource Indentifier), istenen işlemi tanımlayan bir HTTP metodu (ör. GET, PUT, POST), mesaj gövdesi ve iletilen işleme özgü üst verileri içeren bir header alanlarına sahiptir. URI erişilmek istenen kaynağa ait önceden tanımlanmış adresine, HTTP metodu kaynak üzerinde gerçekleştirilmek istenen ekleme, okuma, güncelleme ve silme (CRUD) aksiyonlarına karşılık gelmektedir. Mesaj gövdesi alanında JSON, HTML, XML veya sunucu tarafından kabul edilen diğer bir veri biçiminde formatlanmış içeriği, header alanında ise isteği tanımlayan üst veriler yer almaktadır. JSON anahtar ve değer çiftleri içeren mesaj formatı ile basit, kullanımı kolay ve insanlar tarafından kolaylıkla okunabilirlik özellikleri sebebiyle en popüler mesaj içerik formatlarından biridir. Ancak mesaj içeriğinde bulunan özellik adlarının tekrarlanması mesajların gereğinden fazla büyük olmasına sebep olması sebebiyle en optimal mesaj formatı değildir.

Günümüzde REST API'lerin halka açık bir şekilde yayınlanması diğer üçüncü taraf uygulamalar tarafından bu API'lerin kullanılmasına olanak sağlamaktadır. REST için popüler API yönetim araçları arasında Postman, Amazon API Gateway, IBM API Connect, SAP Integration Suite vb. bulunmaktadır.

1. REST API Özellikleri

Bu bölümde REST API özellikleri; tek tip arayüz, durumsuzluk, istemci sunucu ayrımı, katmanlı sistem, ön belleğe alınabilirlik, istek üzerine kod başlıkları altında anlatılmıştır.

1.1. Tek Tip Arayüz

Sınıfları bağımlılıklarından ayırmak için arayüzlerin kullanılması oldukça eski bir yöntemdir. REST API de istemciyi REST hizmetinin uygulanmasından ayırmak için aynı konsepti kullanmaktadır. Böyle bir arayüzü tanımlamak için standartlar gereklidir. Tek tip arayüz, sunucunun bilgileri standart bir

formatta aktardığını belirtir. Her parçanın bağımsız olarak gelişmesine imkân sunar ve böylelikle mimariyi, dolayısıyla iletişim sürecini basitleştirir. Ayrıca yalnızca istenilen kaynakların, API kullanıcısına sunulması sağlar.

1.2. Durumsuzluk

REST mimarisinde durumsuzluk, sunucunun her isteği önceki tüm isteklerden bağımsız olarak tamamladığı bir iletişim yöntemini ifade eder. İstemciler, kaynakları istedikleri sırada talep edebilir ve her istek diğer isteklerden yalıtılmıştır. Sunucu gelen isteği işlemek için istemcinin durumunu bilmesine gerek duymaz ve aynı şekilde istemci de sunucudaki durum bilgisine ihtiyaç duymaz. Bu REST API tasarımı kısıtlaması, sunucunun isteği her seferinde tam olarak anlayabileceği ve yerine getirebileceği anlamına gelir. Durumsuzluğa ulaşmak için, her istek, önceki mesajları görmeden sunucunun kaynak ve işlem hakkında bilmesi gereken tüm bilgileri içermelidir. Sunucu ayrıca, önceki oturum paketlerinden gelen bağlamsal bilgiler olmadan istemcinin tamamen anlayabileceği bir mesajla yanıt vermelidir.

Sunucu tarafında oturum bilgileri tutulmaz ve sunucu istekler ilgili verileri kaydetmez. Sunucu tarafındaki servisin isteği işleyebilmesi için bir veri gerekiyorsa bu verinin kesinlikle her istek paketi içerisinde gönderilmesi gereklidir. Yani sunucu için istemciden gelen her istek durumsuz bir şekilde başlar ve istemcinin gönderdiği veriler ile geçici bir durum oluşur. İstemci durum verilerini kendi önbelleği aracılığıyla kaydeder. Durumsuzluk, önceki isteklerin ve yanıtların depolanmasından kaynaklanan sunucu yükünü ortadan kaldırarak REST API'leri güvenilir, hızlı ve ölçeklenebilir hale getirir.

1.3. İstemci Sunucu Ayrımı

REST API'lerde istemci ve sunucu bağımsız olarak çalışır. İstemci sunucu tarafındaki altyapı vs. hakkında bilgiye ihtiyaç duymaz. Sunucu tarafında yapılacak değişiklikler istemcileri etkilemeyeceği için bağımlılık azaltılmış olur. İstemci sunucuya kendisi ile paylaşılmış API dokümanları doğrultusunda geçerli istekler iletir ve geçerli yanıtlar alır.

1.4. Katmanlı Sistem

Katmanlı bir sistem mimarisinde istemci sunucuya doğrudan bağlanmak yerine arada bulunan yetkili aracılara bağlanarak sunucudan yanıt alır. İstemci yalnızca istek göndereceği sunucuyu bilir. İstek gönderdiği sunucu arka tarafta belki 10 farklı sunucuya istek gönderip verileri toparlayıp sunuyor olabilir ya da isteği yük dengeleyici sistemler ile işliyor olabilir. Aslında burada katmanlı olabilen taraf sunucu kısmıdır. Katmanlı mimari, sunucularda farklı bileşenlerin değiştirilebilir olması ve bakımına imkân sağlar.

1.5. Önbelleğe Alınabilirlik

RESTful API, sunucu yanıt süresini iyileştirmek için bazı yanıtları istemcide veya başka bir aracıda saklama süreci olan önbelleğe almayı destekler. Sunucu tarafından istemciye istek yanıtının önbelleğe alınıp/alınamayacağı ya da ne süre ile önbelleğe alınabileceği hakkında bilgi verilir. Bu sayede istemci sunucuya tekrar istek göndermeden önce varsa kendi tarafındaki önbelleğe bakar ve önbellek geçerli ise önbelleğe kaydedilen sonuç ile cevap üretilerek ağ (network) tarafında ve dolaylı olarak performans noktasında tasarruf sağlanır.

1.6. İstek Üzerine Kod

REST API'deki isteğe bağlı tek kısıtlamadır. Bağımlılık ilkesine aykırı olduğu için nadiren kullanılır ve kullanımı pek tercih edilmez. Sunucu istemciye çalıştırması için bir kod parçası iletir. Örnek olarak, herhangi bir web sitesindeki bir kayıt formu doldurulduğunda, tarayıcının yapılan hataları (ör. Yanlış telefon numarası) gönderilen kod parçasını kullanarak belirlemesi gösterilebilir.

2. REST API'lerin Avantajları ve Dezavantajları

REST API'lerin en önemli avantajı basitliktir. Servislerin oluşturulması, entegrasyonu, dokümantasyonu gibi noktalarda bu basitlik önemli avantajlar sağlamaktadır. Yaygın olarak kullanılması sebebiyle geliştirme aşamasında kullanılacak önemli araçlara sahiptir ve hataların çözümü için kolaylıkla destek alınabilir.

REST API mimarisinde istemci-sunucu etkileşimleri optimize edildiğinden, bu mimariyi uygulayan sistemler verimli şekilde ölçeklendirilebilir. Durumsuzluk; sunucunun, geçmiş istemci istek bilgilerini saklaması gerekmediğinden sunucu yükünü azaltır. İyi yönetilen önbelleğe alma süreci, bazı istemci-sunucu etkileşimlerini kısmen veya tamamen ortadan kaldırır. Tüm bu özellikler performansı azaltan iletişim sorunlarına neden olmadan ölçeklendirmeyi destekler.

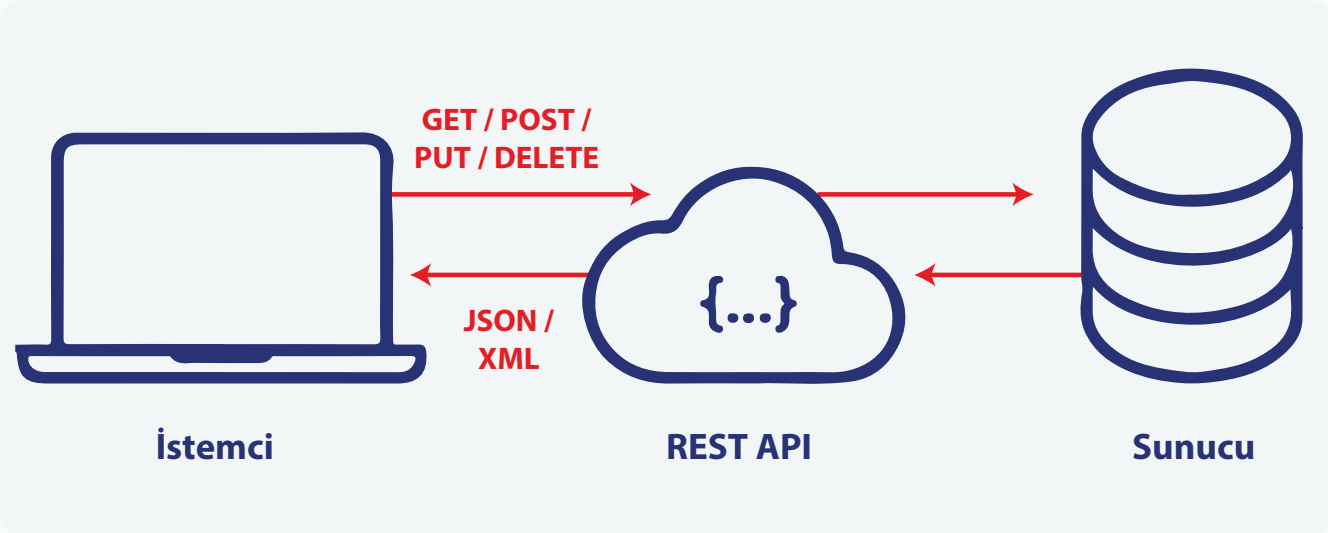
REST API, istemci-sunucu ayrımını destekler. Bu hizmetler, her bir parçanın bağımsız olarak gelişebilmesi için çeşitli sunucu bileşenlerini basitleştirir ve birbirinden ayırır. Sunucu uygulamasındaki platform veya teknoloji değişiklikleri, istemci uygulamasını etkilemez. Uygulama işlevlerini katmanlı yapıda olması özelliği, esnekliği daha da artırır.

REST API'ler, kullanılan teknolojiden bağımsızdır. API tasarımını etkilemeden hem istemci hem de sunucu uygulamalarını çeşitli programlama dillerinde geliştirme yapılabilir. Ayrıca, iletişimi etkilemeden her iki taraftaki temel teknoloji de değiştirilebilir.

REST API'lerin çeşitli dezavantajları da bulunmaktadır. HTTP protokolü istek ve yanıt mesajları üzerine eklediği katmanlar performansın en yüksek seviyede kullanılmasını kısıtlamaktadır. Ayrıca HTTP protokolünün kullanılması REST API'leri çeşitli güvenlik zafiyetlerine yol açmaktadır. Son olarak REST API'ler istek-yanıt temelli yaklaşımda çalıştığı için canlı veri akışını desteklememektedir.

3. REST API'ler Nasıl Çalışır?

Şekil 1'de de görüldüğü üzere istemci, bir kaynağa ihtiyaç duyduğunda REST API'yi kullanarak sunucuya istek gönderir. API geliştiricileri, istemcinin REST API'yi nasıl kullanması gerektiğini sunucu uygulamasının API dokümantasyonunda açıklar. Sonrasında sunucu, istemcinin kimliğini doğrular ve istemcinin bu istekte bulunma hakkı olduğunu kontrol eder. Daha sonra sunucu, isteği alır ve dahili olarak işler. Son olarak sunucu, istemciye bir yanıt verir. Yanıt, isteğin başarı durumu ve istemcinin talep ettiği bilgileri içerir.



Şekil 1. REST API Mimarisi

3.1. İstemci Talebi

REST API'lerde istemci isteği benzersiz bir kaynak tanımlayıcısı, HTTP metodu, mesaj gövdesi ve header bileşenlerini içerir. Sunucu, her bir kaynağı benzersiz kaynak tanımlayıcılarla tanımlar. URI, kaynağa giden yolu belirtir. HTTP metodu, sunucuya kaynağa yapılması istenen işlemi gösterir. Başlıca dört HTTP yöntemi şunlardır:

- **GET:** İstemciler, sunucuda belirtilen adreste yer alan kaynaklara okumak için erişmek istediğinde GET isteğini kullanır.
- **POST:** İstemciler, verileri sunucuya göndermek için POST isteğini kullanır. Aynı isteği birden fazla kez göndermek, aynı kaynağın birden fazla kez oluşturulmasına neden olur.
- **PUT:** İstemciler, sunucudaki kaynakları güncellemek için PUT yöntemini kullanır. POST'un aksine, RESTful web hizmetinde aynı PUT isteğini birden fazla kez göndermek aynı sonucu doğurur.
- **DELETE:** İstemciler, kaynağı kaldırmak için DELETE isteğini kullanır. Bir DELETE isteği, sunucunun durumunu değiştirebilir.

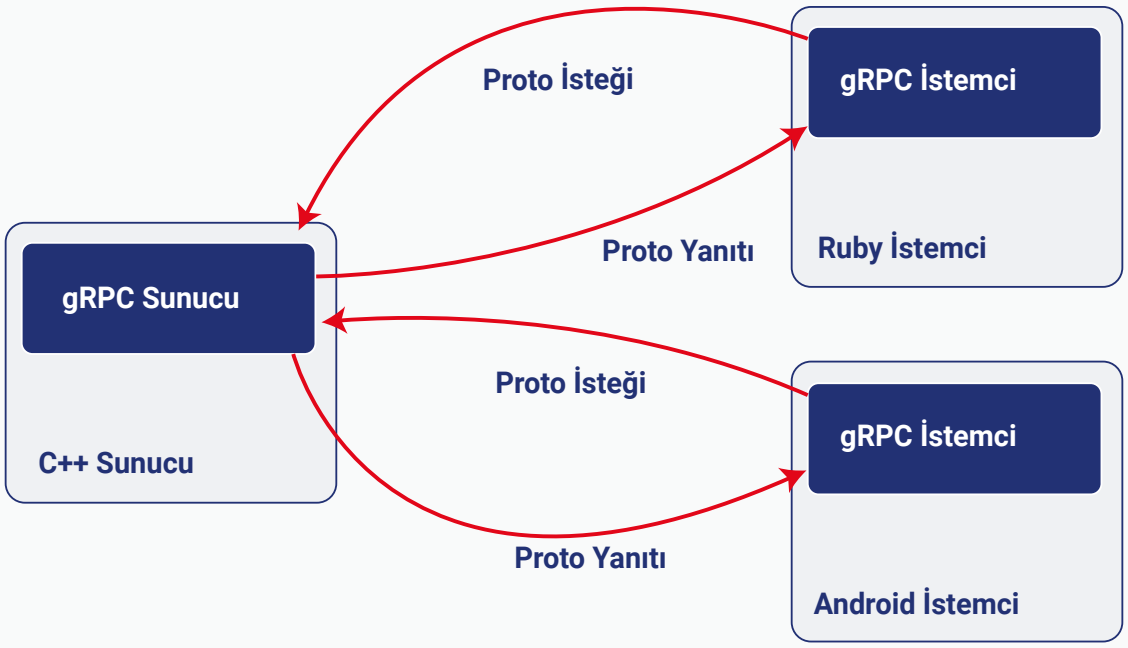
3.2. Sunucu Yanıtı

REST API'lerde sunucu yanıtı durum kodu, mesaj gövdesi ve header bileşenlerini içerir. Durum kodu isteğin başarılı veya başarısız olduğunu belirten üç haneli bir koddur. 2XX kodları başarıyı ifade ederken 4XX ve 5XX kodları hataları belirtir. 3XX kodları ise URL yönlendirmesini ifade eder. Mesaj gövdesi header kısmında gösterilen metin formatında içerik bilgisini içerir. Header yanıtla ilgili üst verileri içerir. Üst veriler, istek hakkında sunucu, kodlama, tarih ve içerik türü gibi çeşitli bilgileri içerir.

gRPC

RPC mimarisinin bir çeşidi olan gRPC (Google Remote Procedure Call), Google tarafından 2015 yılında geliştirilen ve mikroservisler arasında yüksek hızlı iletişime ve ölçeklenebilirliğe imkân veren açık kaynak kodlu ve sözleşme tabanlı bir platformlar arası bir iletişim protokolüdür. HTTP 2.0 protokolü üzerine inşa edilmiştir, çift yönlü veri alışverişine ve farklı dillerde geliştirilmiş servislere entegrasyona imkân verir. HTTP protokolü ve RPC etkileşimleri, karmaşıklığı azaltmak amacıyla API geliştiricileri ve sunuculardan soyutlanmıştır. gRPC, çağırılacak veri biçimleri ve işlemler üzerinde bir sözleşme/uzlaşı oluşturmak için Arayüz Tanımlama Dili'ni (IDL) kullanılır. IDL kullanılarak uzaktan çağrılabilecek yöntemler ve parametreler veri yapısı ve dönüş türleri tanımlanır.

gRPC, veri alışverişinde için oldukça verimli bir mesajlaşma formatı sağlayan protokol arabelleklerini (Protobuf) mesajlaşma formatı olarak kullanır. gRPC için popüler API yönetim araçları arasında omgRPC, gRPCox, Wombat, Milkman vb. bulunmaktadır.



Şekil 2. gRPC İstemci Sunucu Mimarisi

1. Binary Tabanlı Mesaj Formatları

Bu bölümde binary tabanlı mesaj formatları; protokol arabellekleri, avro, thrift başlıkları altında anlatılmıştır.

1.1. Protobuf (Protokol Arabellekleri)

Protobuf, istemci kütüphanelerinin otomatik olarak oluşturulmasını ve mikroservislerin basit bir şekilde tanımlanmasını sağlayan bir serileştirme protokolüdür. Google, Protokol arabelleklerini tasarlarken basitlik ve verimliliğe odaklanmıştır. Dil bağımsızdır ve herhangi bir dile kolaylıkla entegre edilebilir. API geliştiricileri, proto dosyaları ile istemciler ve sunucular arasındaki hizmetleri ve mesajları tanımlar. Protokol arabellekleri mesajlar üzerinden yapılandırılır. Her mesaj alan adı verilen bir dizi anahtar değer çifti ve alan türünü içerir. Örnek bir proto mesajı aşağıdaki gibidir:

```
message Person {
 string name = 1;
 int32 id = 2;
 bool has_ponycopter = 3;
}
```

Dosyalar, uzak hizmetlerle mesaj alışverişi yapmak için istemci ve sunucu kodu oluşturan protoc derleyici tarafından yüklenir. Protokol arabellekleri kullanılarak kodlanan mesajlar binary biçiminde temsil ederken, XML veya JSON temsillerinden çok daha küçüktür. Protokol arabellekleri insanlar tarafından okunulabilirlikten ödün vererek daha az CPU-yoğunluğu, daha az bant genişliğine gereksinim duyar. Bu durum iletişim gecikmesinin azalmasını sağlayarak ve daha yüksek performanslı sistemler oluşmasına imkân verir. En son sürüm olan proto3 kullanımı daha basittir ve gRPC için en yeni yetenekleri sunar.

1.2. Avro

Avro, Apache Software Foundation tarafından geliştirilen bir veri serileştirme sistemi ve mesajlaşma formatıdır. Temel olarak, Avro verileri şemalar kullanarak temsil eder ve bu sayede verilerin taşınması ve paylaşılması kolaylaşır. Şema, verinin yapısını ve veri türlerini belirleyen bir JSON formatında tanımlanır. Avro'nun en önemli özelliklerinden biri, şemaya dayalı olmasıdır. Şema, verinin doğru bir şekilde seri hale getirilmesi (serialization) ve ayrıştırılması (deserialization) için gereken talimatları sağlar. Bu sayede, veri gönderen ve alıcı arasında tutarlılık sağlanır. Ayrıca, Avro şemaları, dinamik tip kontrolü sağlar ve verilerin uyumlu olup olmadığını doğrulamak için kullanılabilir. Avro, dil bağımsız bir mesajlaşma formatıdır. Yani, farklı programlama dillerinde kullanılabilen kod üretebilir. Bu özellik, farklı dillerde yazılmış sistemler arasında veri alışverişini kolaylaştırır. Bununla birlikte, Avro veri sıkıştırma özelliği sunar. Verilerin sıkıştırılması, ağ trafiği ve depolama maliyetini azaltır.

Bu özellik özellikle büyük veri sistemleri ve dağıtık sistemler için önemlidir. Avro, birçok büyük veri teknolojisi tarafından desteklenmektedir ve genellikle Big Data sistemleri, veri entegrasyonu ve dağıtık sistemler gibi senaryolarda yaygın olarak kullanılır. Örneğin, Apache Kafka, Apache Hadoop ve Apache Spark gibi popüler veri işleme çözümleri Avro'yu desteklemektedir.

1.3. Thrift

Thrift, Facebook tarafından geliştirilen bir veri iletişim formatıdır. İstemci-sunucu iletişimi veya veri depolama gibi senaryolarda kullanılabilir. Thrift, platformlar arası bir çözüm sunar ve farklı dillerde yazılmış sistemler arasında veri alışverişini kolaylaştırır. Thrift, birleştirilmiş veri yapısına IDL'e dayanır. Bu IDL, veri türleri, yapılar, fonksiyonlar ve servisler gibi veri yapılarını tanımlamak için kullanılır. IDL, bir arayüz tanımı olarak düşünülebilir ve istemci ve sunucunun birbirleriyle nasıl iletişim kurduklarını belirler. Thrift mesajlaşma formatı, verilerin seri hale getirilmesi (serialization) ve ayrıştırılması (deserialization) için kullanılan bir protokol olarak çalışır. Bu sayede, farklı dillerde yazılmış istemci ve sunucuların birbirleriyle uyumlu bir şekilde veri alışverişi yapması sağlanır. Thrift, çeşitli programlama dilleri için kod üretebilir ve bu dillerdeki nesneleri Thrift mesajlarına dönüştürüp iletebilir. Thrift desteklediği diller arasında C++, Java, Python, Ruby, PHP ve daha birçok dil bulunmaktadır. Thrift, hafif bir yapıya sahip olması ve yüksek performans sunması nedeniyle özellikle dağıtık sistemlerde ve mikro hizmet mimarilerinde tercih edilen bir mesajlaşma formatıdır.

2. gRPC'nin Avantajları

Bu bölümde gRPC'nin avantajları; performans, otomatik kod oluşturma, standartlar, veri akışı, zaman aşımı, istek iptali başlıkları altında anlatılmıştır.

2.1. Performans

gRPC mesajları verimli bir binary mesaj formatı olan Protobuf kullanılarak serileştirilir. Binary formatında küçük boyutlu serileştirilmiş mesajlar HTTP 2.0 protokolünün hızı ile birleştirildiğinde performanslı bir iletişim altyapısı oluşturulmuş olur.

2.2. Otomatik Kod Oluşturma

gRPC servislerinin metotlarının ve mesajlarının kontratlarını tanımlayan şey proto dosyalarıdır. Bu dosyalar gRPC frameworkleri ile ihtiyaç duyulan sınıflar, metotlar ve mesajların oluşturulmasında kullanılabilir. Farklı diller kullanılsa bile tek bir proto dosyası üzerinden o dile ait kodların otomatik üretilmesi sağlanabilir.

2.3. Standartlar

gRPC iletişimi için standartlar belirlidir ve proto dosyaları üzerinden bu süreç keskin bir şekilde tanımlıdır. Standartların belirlenmesi karmaşıklığın azaltılmasına yardımcı olur.

2.4 Veri Akışı

gRPC HTTP 2.0 üzerinde çalışır ve HTTP 2.0'ın bütün veri akışı özelliklerini destekler.

- **Tek İstek:** İstemcinin sunucuya tek bir istek gönderdiği ve tek bir yanıt aldığı en basit gRPC türüdür.
- **Sunucu Veri Akışı:** Bir istemcinin çağrı yaptıktan sonra yanıt olarak sunucunun veri akışını başlatması ve çağrı yapılan tüm mesajların istemciye aktarılmasıdır.
- **İstemci Veri Akışı:** İstemcinin bir veri akışı oluşturarak tüm mesajları sunucuya ilettiği ve mesaj gönderimi tamamlandığında sunucunun tek bir mesaj halinde yanıt döndüğü türdür.
- **Çift Yönlü Veri Akışı:** İstemci ve sunucu birbirlerine tek bir mesaj gönderir ya da veri akışı oluşturur. İki veri akışı birbirinden bağımsız işler. Mesajlar herhangi bir sırayla okunabilir.

2.5. Zaman Aşımı

İstemciler üzerinde bir gRPC çağrısının tamamlanabilmesi için en fazla ne kadar süre beklenmesi gerektiği tanımlanabilmektedir. Sunucu da belirli bir çağrının zaman aşımına uğrayıp uğramadığını veya çağrıyı tamamlamak için ne kadar süre kaldığını sorgulayabilir. Zaman aşımı özelliği oluşabilecek yığılma ve aşırı kaynak tüketiminin önüne geçer.

2.6. İstek İptali

İstemci veya sunucu herhangi bir zamanda gRPC'yi iptal edebilir. İptal gRPC'yi derhal sonlandırır ve daha fazla işlem yapılmaz.

3. gRPC'nin Dezavantajları

Bu bölümde gRPC'nin dezavantajları; limit tarayıcı desteği, okunabilirlik, geliştirme zorluğu başlıkları altında anlatılmıştır.

3.1. Limitli Tarayıcı Desteği

gRPC HTTP 2.0 protokolünü kullanır, bu nedenle gRPC mimarisi doğrudan tarayıcıdan çağrılar yapmak

için kullanılamaz. Bu durum web uygulamalarında direk olarak kullanımın önüne geçmektedir. Tarayıcıdan gRPC çağrısı yapmak için proxy kullanılması gereklidir.

3.2. Okunabilirlik

gRPC mesajları varsayılan olarak Protobuf ile kodlanır. Protobuf insanlar tarafından okunamayan binary yapısını kullanmaktadır. Bu durum mesajların içeriğinin okunması gereken durumlarda dezavantaja sebep olur..

3.3. Geliştirme Zorluğu

REST API ile karşılaştırıldığında gRPC mimari yaklaşımı ile geliştirme yapmak daha fazla süre alır. Uygulaması ve öğrenmesi daha kolay olan JSON yerine Protobuf'ı kullanan gRPC daha karmaşıktır ve daha fazla uzmanlık gerektirir.

REST API ve gRPC MİMARİSİNİN KARŞILAŞTIRILMASI

Bu bölümde REST API ve gRPC mimari stillerinin aşağıda belirlenmiş olan özellikler için karşılaştırılmıştır.

1. HTTP Protokolü

REST API, istek-yanıt iletişim modeli kullanılarak HTTP 1.1 protokolü üzerine kurulmuştur. REST API mimarisinde sunucuya birden fazla istek ulaştığında, sunucu istekleri birbirinden bağımsız olarak tekil olarak işler. gRPC ise HTTP 2.0 protokolünün üzerine kurulmuştur. HTTP 2.0, HTTP 1.1 protokolünün sınırlamalarını ve verimsizliklerini gidermek ve web performansını artırmak için geliştirilen ikinci ana sürümdür. HTTP 2.0, çoklu işlem özelliği ile aynı bağlantı üzerinden birden fazla isteği aynı anda gönderme imkanı sunar. Ayrıca binary protokol kullanarak daha az veri iletişimi ve başlık sıkıştırmasıyla daha iyi performans sağlar. Sunucu zorlaması (server push) ile istemcinin istek yapmadan önce gerekli kaynakları proaktif olarak alabilmesi sağlanır. Önceliklendirme özelliği ile önemli kaynaklar öncelikli olarak iletilir. HTTP 2.0, geriye uyumlu olup web iletişimini hızlandırır, verimliliği artırır ve daha iyi bir kullanıcı deneyimi sunar. HTTP 2.0 üzerine inşa edilen gRPC, çift yönlü

veri akışını destekler ve aynı anda birden çok isteğe yanıt verebilir. Ek olarak gRPC, REST'e benzer tekli iletişimi de destekler.

2. Mesaj Formatı

REST API, veri alışverişinde genellikle JSON ve XML formatlarını kullanır. gRPC ise Protokol arabelleklerini (Protobuf) kullanarak serileştirilmiş binary formatında mesaj alışverişini kullanır. Belirtilen bütün formatlar platform ve dilden bağımsız olmakla birlikte birbirlerine üstünlük kurdukları çeşitli alanlar mevcuttur. JSON ve XML formatları insanlar tarafından daha kolaylıkla okunulabilirler ve daha basit ve esnek yapıya sahiptir. Protokol arabellekleri ise binary formatında olması sebebiyle insanlar tarafından okunamazken, mesaj boyutlarının daha küçük olması sebebiyle daha az bant genişliğine ihtiyaç duyar ve genellikle daha hızlı bir veri alışverişine imkân tanır.

3. Serileştirme ve Tip Kontrolü

REST, çoğu durumda hem istemci hem de sunucu için serileştirme ve hedef programlama diline dönüştürme gerektiren JSON veya XML kullanır, böylece yanıt süresini ve istek/yanıtı ayrıştırırken hata olasılığını artırır. Bununla birlikte gRPC, Protobuf değişim formatı kullanılarak seçilen programlama diline otomatik olarak dönüştürülen kesin olarak yazılmış mesajlar sağlar.

4. Ağ Gecikmesi

HTTP 1.1 protokolünü kullanan REST, her istek için bir TCP el sıkışması gerektirir. REST API'lerde bu sebeple bazı gecikme sorunları yaşanabilir. gRPC ise veri akışlarını destekleyen HTTP 2.0 protokolüne dayanır. İstemciler her istek için yeni bir TCP el sıkışmasına gerek duymadan birden çok isteği iletebilir. Bunun yanı sıra sunucu, kurulan bağlantı aracılığıyla istemcilere anlık bildirimler gönderebilir.

5 Tarayıcı Desteği

REST API, HTTP 1.1 protokolünü kullanması sebebiyle tarayıcıların büyük bir çoğunluğu tarafından desteklenmektedir. Ancak çoğu tarayıcının HTTP 2.0 için olgunlaştırılmış desteği olmadığı için gRPC, REST'e göre oldukça sınırlı bir tarayıcı desteğine sahiptir. HTTP 1.1 ve HTTP 2.0 arasındaki dönüşümü gerçekleştirmek için gRPC-web ve proxy katmanı kullanılır. Bu sebeple gRPC genellikle dışa dönük servisler yerine dahili servisler arasındaki işlemler için kullanılır.

6. Kod Oluşturma Araçları

REST API mimarisi yerleşik olarak kod oluşturma özelliği sağlamaz. REST API mimarisinde API istekleri için otomatik olarak kod oluşturmak için Swagger, Postman vb. üçüncü taraf araçların kullanılması gereklidir. gRPC ise yerleşik protoc derleyicini kullanarak otomatik kod oluşturma özelliğine sahiptir. Protoc derleyici, çeşitli programlama dilleri için geniş bir uyumluluk yelpazesiyle sahiptir ve üçüncü taraf uygulamaların kullanılması gereksinimini ortadan kaldırır.

7. Geliştirme Kolaylığı

REST API mimarisi basitliği ve sağladığı geliştirme kolaylığı sebebiyle daha popüler ve yaygın olarak kullanılmaktadır. gRPC ise daha karmaşık bir yapıya sahip olması ve sahip olduğu otomatik kod geliştirme araçlarına rağmen geliştirme sürelerinin daha uzun sürmesi sebebiyle geniş çapta benimsenmemiştir. Ayrıca REST API kolaylıkla oluşturulabilen düz metin biçimi olan JSON mesaj formatını desteklediği için geliştiricilerin REST API ile çalışması daha kolaydır. JSON formatı herhangi bir hata durumunda geliştiricilerin sorunu incelemesi ve çözmesi için kolaylık sağlar.

Sonuç ve Öneriler

Sonuç olarak her iki mimari yaklaşım da kendine özel farklı avantaj ve dezavantajlara sahiptir. En uygun mimari yaklaşımı seçmek için geliştirme sürecinde API gereksinimlerinin tespit edilmesi ve hangi mimari yaklaşımın daha doğru olacağına bu gereksinimlere göre karar verilmesi gereklidir.

REST API daha çok kullanılan ve daha popüler mimari stildir. Geliştirilmek istenen API eğer dışarıya açık, diğer uygulamaların entegrasyonuna izin veren, daha esnek, daha sezgisel ve ölçeklenebilir olması talep ediliyorsa tercih edilmelidir. Mikroservisler veya üçüncü taraf uygulamaların entegrasyonunda daha kullanışlıdır. Mesaj formatı olarak genellikle insanlar tarafından okunabilen JSON formatı kullanılır. Bu sayede geliştiriciler için hata ayıklama süreçlerinin daha kolay olmasını sağlar. Evrensel tarayıcı desteği sayesinde herhangi bir istemci kurulumu gerekmeksizin birçok platformda çalışabilir.

gRPC HTTP 2.0 ve protokol arabelleklerini kullanması sebebiyle REST ile kıyaslandığında daha yüksek performans gösterir ve daha hızlı mesaj iletilmesine imkân verir. Dağıtık sistemlerin haberleşmesinde daha kullanışlıdır. gRPC mimari yaklaşımı tercih edilirken iletilecek payload boyutu, ağ gecikme süresi, uygulamanın ölçeklenebilirliği ve izlenebilirliği göz önünde bulundurulmalıdır. Örneğin mesajlaşma trafiğinin loglanılması için okunabilir hale getirilmesi gerekmektedir. Daha verimli mesaj iletimi gerektiren IoT sistemleri, mobil uygulamalar ve veri akışına gereksinim duyan sistemlerde tercih edilebilir. gRPC ile çalışmak için daha fazla uzmanlık gereklidir. gRPC veri türlerini kontrol eder ve yanlış türde verilerin iletilmesine engel olur. gRPC mimarisi dışa açık sistemler yerine performans talep eden dahili sistemlerde kullanılması daha uygundur. gRPC farklı dillerin kullanıldığı sistemlerde ortak proto dosyası yapısı sayesinde rahatlıkla kullanılabilir. Ayrıca birçok dil için otomatik kod oluşturma imkânı vardır. Gerçek zamanlı veri akışı gerektiren video, ses, çevrimiçi oyun, dosya paylaşımı, sohbet uygulamaları gibi sistemlerde de gRPC kullanmak daha uygundur. Veri akışı özelliği sayesinde ağ gecikmesini azaltarak daha iyi bir kullanıcı deneyimi sağlar. En büyük dezavantajı ise tarayıcı uyumluluğunun sınırlı olmasıdır.

Kaynakça

1. Anshul Bansal. (2021) REST vs. gRPC, Erişim adresi https://www.baeldung.com/rest-vs-grpc

2. Mitul Makadia (2022) Key Differences Between REST and gRPC, Erişim Adresi https://marutitech.com/rest-vs-grpc

3. Mariana Berga, Andre Santos (2021) gRPC vs REST: comparing APIs architectural styles Erişim Adresi https://www.imaginarycloud.com/blog/grpc-vs-rest/#REST

4. Recep İnanç (2021) Benchmarking – REST vs. gRPC Erişim Adresi https://medium.com/sahibinden-technology/benchmarking-rest-vs-grpc-5d4b34360911

5. Ruwan Fernando (2019) Evaluating Performance of REST vs gRPC Erişim Adresi https://medium.com/@EmperorRXF/evaluating-performance-of-rest-vs-grpc-1b8bdf0b22da

6. Amazon Web Services, (t.y.) What is a restful API? https://www.aws.amazon.com/what-is/restful-api

İşçi Blokları Mahallesi Muhsin Yazıcıoğlu Caddesi No:51/C 06530 Çankaya/ANKARA

+90 (312) 289 92 22 - yte.bilgem@tubitak.gov.tr

TÜBİTAK - BİLGEM Yazılım Teknolojileri Araştırma Enstitüsü (YTE)

yte.bilgem.tubitak.gov.tr