

# Web Programlama I

---

## Ders 07 - RESTful API Teorisi

23.11.2021

Erciyes Üniversitesi

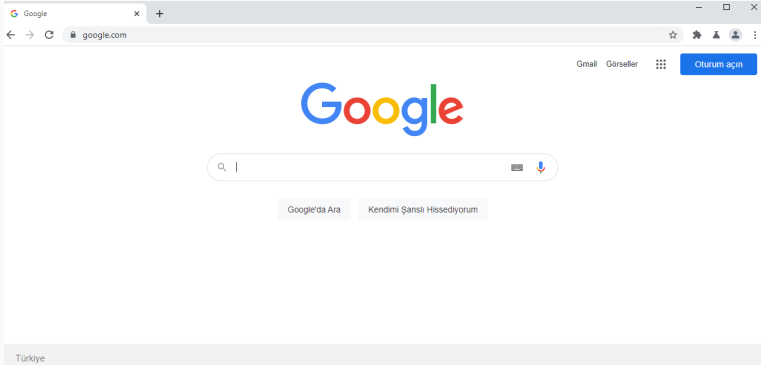
Bilgisayar Mühendisliği Bölümü

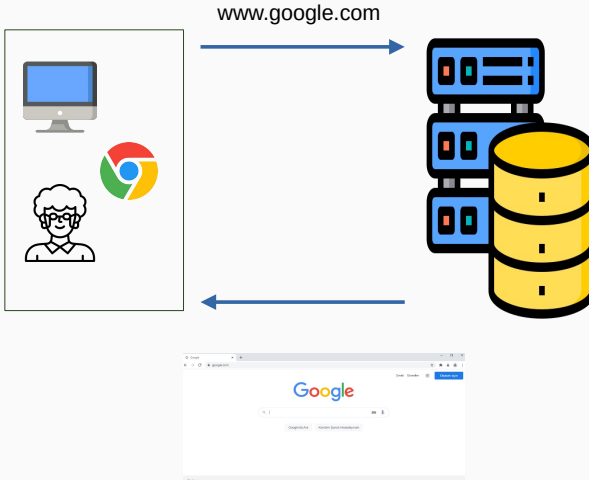
**Eğitmen:** Ömür ŞAHİN

- REST web servislerin genel konseptini anlamak
- Statik kaynaklar ve dinamik içeriklerin benzerlikleri ve farklılıklarını anlamak.
  - HTML/CSS vs. JSON

# 1-HTTP

---





- HTTP isteği TCP üzerinden iletilir ve HTML içeriği geri döndürülür. Bu içerik tarayıcı tarafından görselleştirilir.

# URL (Uniform Resource Locator)

- Bir web kaynağını göstermektedir ve bu web kaynağının nasıl çekileceğini belirtir.

scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]

https://en.wikipedia.org:443/wiki/URL#Syntax



scheme



host



port



path



fragment

- Scheme: Kaynağa nasıl erişileceğini gösterir.
  - http, https, ftp vs.
- Host: Sunucu adı veya doğrudan IP adresidir.
- Port: Uzak sunucu ile bağlantı kurabilmek için dinlenen sunucu portu.
- Path: Kaynak tanımlayıcı. Çoğu zaman bir hiyerarşik düzen içerisindedir.
  - /a/b/c gibi.
- Query: "?" ile başlar ve <key>=<value> şeklinde erişilir. "&" ile ayrılır.
  - <https://github.com/search?q=reach&type=repositories>
- Fragment: Ayrıntılı kaynak tanımlayıcı. Genellikle ana isteğin içerisindeki bir parçadır.
  - Örnek: HTML sayfa içerisindeki bir bölüm.

- Mesaj yapısını tanımlamada kullanılan protokoldür.
- 1989 yılında CERN'de tanımlanmıştır.
- 1995: versiyon 0.9
- 1996: versiyon 1.1
- 1999: güncellemelerle 1.1
- 2014: daha fazla güncellemeler gerçekleşti yine 1.1
- 2015: versiyon 2.0



- HTTP versiyonlaması kötü versiyonlamaya örneklerdendir.
- Bütün değişikliklere rağmen 18 yıl boyunca versiyon 1.1 olarak anılmıştır.
- 2014 yılındaki güncellemeyi pek çok insan fark etmemiştir. Hala pek çok kütüphane eski sürümü kullanmaktadır.

# RFC (Request for Comments)

Teknik olarak RFC bir standart değildir ancak uygulamada öyle kullanılmaktadır.

- RFC 7230, HTTP/1.1: Mesaj yapısı ve yönlendirme
- RFC 7231, HTTP/1.1: Semantik ve içerik
- RFC 7232, HTTP/1.1: Şartlı istekler
- RFC 7233, HTTP/1.1: Kısmi istekler (genellikle büyük dosya transferi için)
- RFC 7234, HTTP/1.1: Önbellekleme
- RFC 7235, HTTP/1.1: Kimliklendirme
- RFC 7540, HTTP/2
- vs.
- Web servislerle çalışırken, HTTP'nin temellerini anlamak gereklidir.

- v2 2015 yılında hayatımıza girmiştir ancak hala çok yaygın değildir.
- Aksi belirtilmedikçe v1.1'den söz ediyor olacağız.
- Kullanıcı açısından bakıldığında v2 ile v1.1 birbirine benzerdir.
  - Aynı metod/fiil'lere sahiptir yalnızca daha iyi optimize edilmiş ve daha performanslıdır.
  - Daha fazla işlevsellik eklenmiştir ancak var olan değiştirilmemiştir.
- Temel farklılık: v1.1 "text" tabanlı iken v2'nin kendi byte formatı (daha az yer kaplar ancak insanlar tarafından okunup parse edilmesi zordur) vardır.

Bir HTTP mesajının 3 temel bileşeni vardır.

- Yapılmak istenen eylemi belirten ilk satır (GET gibi)
- Ekstra meta-bilgi içeren Header
  - Cevabı hangi formatta istiyorsunuz? JSON, Text, XML gibi.
  - Hangi dilde cevap istiyorsunuz? Türkçe, İngilizce vs.
- (Opsiyonel) Body: Her şey olabilir.
  - Request: Genellikle kullanıcı bilgileri (login/şifre, form bilgileri vs.)
  - Response: Çekilen mevcut kaynak (HTML sayfası gibi)

- `<METOT><KAYNAK><PROTOKOL> \r\n`
- Örnek: `GET / HTTP/1.1 \r\n`
  - `<METOT>`: **GET**
  - `<KAYNAK>`: `/`
  - `<PROTOKOL>`: **HTTP/1.1**
- Kaynak her şey olabilir.
  - html, jpeg, json, xml, pdf, vs.
- Bir kaynak yol ile tanımlanır.
  - kök dizin `"/` dosya sisteminde nerede?

- **GET:** Bir kaynağı çekmek için
- **POST:** Veri gönderme (HTTP body içerisinde) ve/veya bir kaynak yaratmak için
- **PUT:** Var olan bir kaynağı yenisi ile değiştirmek için
- **PATCH:** Var olan bir kaynağı kısmi güncelleme için
- **DELETE:** Bir kaynağı silmek için
- **HEAD:** GET gibi çalışır ancak kaynak verisini göndermeden yalnızca header bilgisi döner.
- **OPTION:** Bir kaynak için hangi metotların kullanılabilir olduğuna bakmak için
- **TRACE:** Debug etmek için
- **CONNECT:** Proxy aracılığı ile bağlantı kurmak için

- Her bir metot oldukça açık bir semantiğe sahiptir.
  - GET bir kaynak getirmeye yararken, DELETE kaynağı silmelidir.
- Ancak sunucunun bunları nasıl ele aldığı tamamiyle sunucuya bağlıdır.
  - Sunucu GET çalıştırıldığında bir kaynağı silebilir.

- `"/x.html"` ele alındığında:
- **Yanlış:** GET `"www.abc.com/x.html/delete"` ile `"x.html"` silinmesi.
- Ayrıca sorgu içerisinde kullanılması da yanlıştır.
  - `"www.abc.com/x.html?method=delete"`
- Yollar gerçekleştirilecek eylemleri değil, kaynağı tanımlamalıdır.



- RFC 7231: Bir metodun **idempotent** olarak tanımlanabilmesi için bir veya daha fazla çağırılması durumunda aynı etkiyi gösteriyor olması gerekmektedir.
- Bir istemci bir isteği gönderdikten sonra bağlantı sonlanırsa ve ardından yeni bir bağlantı kurulursa, bu isteği tekrar gerçekleştirebilir.

# Hangi metotlar idempotent olarak tanımlanır?

- GET ✓
- POST
- DELETE ✓
- PUT ✓
- PATCH
- HEAD ✓

- Metot/Kaynak bilgileri dışında extra bilgiler de içermektedir.
- `<key>:<value>`
- Örnek:
  - Kaynak hangi formatta olmalı? HTML? JSON?
  - Hangi dilde olmalı? Türkçe? İngilizce?
  - Ben kimim (kimliklendirme için önemlidir)
  - TCP bağlantısı devam etmeli mi yoksa bu istekten sonra kapatılmalı mı?
  - vs.

▼ Hypertext Transfer Protocol

> GET / HTTP/1.1\r\n

Host: google.com\r\n

Connection: keep-alive\r\n

Upgrade-Insecure-Requests: 1\r\n

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.90 Safari/537.36

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.9

Accept-Encoding: gzip, deflate\r\n

- www.google.com adresine istek atıldığında oluşan request header.
- Chrome Dev Tools veya Wireshark gibi programlarla takip edilebilir.

- Son header bilgisinden sonra, bir boş satır bulunmalıdır.
- Request: POST, PUT, PATCH için gereklidir.
- Response: GET için gereklidir.

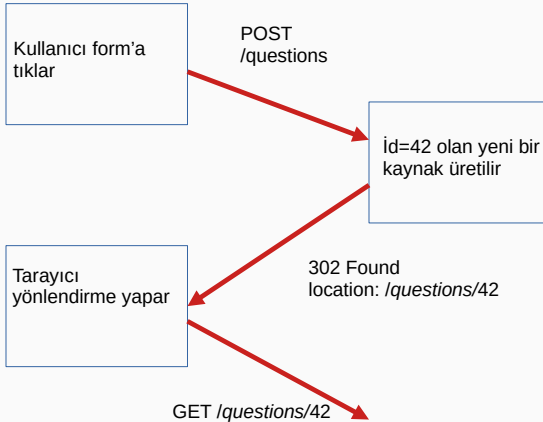
- HTTP request ile aynı türde header ve body'e sahiptir.
- Sadece ilk satırda farklılaşmaktadır.
- <PROTOCOL><STATUS><DESCRIPTION>
  - Örnek: "HTTP/1.1 200 OK"
- Bir istek yapıldığında sunucuda pek çok değişiklik gerçekleşebilir ve bu değişiklikler durum kodları (status) ile bildirilir.

- 3 basamaklı sayılardan oluşur.
- **1xx**: Bilgi ve geçici yanıtlar
- **2xx**: Başarılı
- **3xx**: Yönlendirme
- **4xx**: Kullanıcı hataları
- **5xx**: Sunucu hataları

- **200:** OK
- **201:** Kaynak oluşturuldu
- **202:** Kabul edildi ancak tamamlanmadı (ör: arkaplan işlemleri)
- **204:** no content (ör: PUT veya DELETE sonucu)

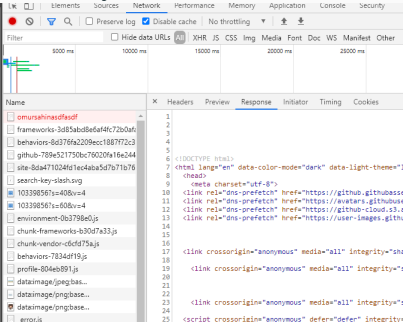
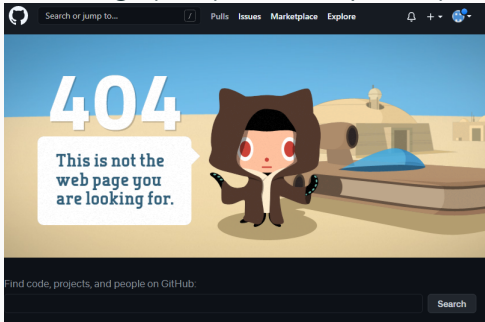


- **301:** Daimi yönlendirme
  - X, Y adresine yönlendirildiğinde istemci bir daha X üzerinden işlem yapmaz, işleme Y üzerinden devam eder.
- **302:** Geçici yönlendirme
  - Method değişimi olabilir (POST'tan GET'e gibi)
- **307:** Aynı method ile geçici yönlendirme
  - ör: POST, POST olmaya devam eder.
- "Location" başlığı: Nereye yönleneceğini belirten URI



- **400**: bad request (genel hata kodu)
- **401**: unauthorized (yetkisiz)
- **403**: forbidden (Mevcut kullanıcı ile erişim izni olmaması durumu)
- **404**: Not Found (İçlerinde en ünlüsü :) )
- **405**: Method not allowed (ör: yalnızca okunabilir bir kaynak üzerinde DELETE uygulanırsa)
- **415**: unsupported media type (ör: yalnızca JSON destekli sunucuya XML gönderilirse)

Bir hata gerçekleşse bile, response içerisinde body bulunabilir.



- **500: Internal Server Error**
  - Genellikle NullPointerException gibi bug'larda ortaya çıkan hatadır.
    - Not: Eğer uygulama çökerse hiçbir cevap alınamaz.
  - Veritabanı bağlantısı gibi harici servislerde de hata olması durumunda bu hata ortaya çıkar.
- **503: Service Unavailable**
  - Çok fazla istek atılması durumunda veya planlı bakım durumlarında bu hata ortaya çıkar.

# HTTPS (HTTP Secure)

- HTTP'nin Transport Layer Security (TLS) kullanılarak şifrelenmiş halidir.
- Genellikle 80 portu yerine 443 portu üzerinde çalışır.
- URI tanımı HTTP ile hemen hemen aynıdır (sadece scheme değişir.).
- Bütün HTTP mesaj şifrelidir ancak TCP kullanılmaya devam edilir.
  - Bunu anlamı, uzak sunucuya ait IP adres ve PORT bilgilerine hala erişilebilir durumdadır.
  - Proxy ağlar kullanılarak (TOR gibi) veya VPN sağlayıcılar kullanılarak bu engellenebilir.

## 2-Web Servisler

---

- Network üzerinde API sağlama
- TCP bağlantıları
- HTTP en yaygın protokoldür
- Web servisler, TCP portu üzerinden bağlantı oluşturup gelen talepleri işleme alan süreçlerdir.



- REST
  - En fazla tercih edilen
  - HTTP protokolüne sıkı sıkıya bağlıdır
  - Bir protokol değil, mimari kılavuzdur.
  - Genellikle JSON kullanılır.
- SOAP
  - Geçmişte çok yaygındı ancak artık oldukça azalmış durumda.
  - HTTP üzerinde kullanılan bir protokoldür.
  - XML kullanılır.
- GraphQL
  - Yeni eleman :)

- Uygulamaya ağ üzerindeki çeşitli işlevsellikleri eklemek için.
  - Ör: listeyi görmek için bakabilirsiniz:  
<https://www.programmableweb.com/>
- Frontend ve backend'i birbirinden ayırmak isterseniz
  - JavaScript HTML render etmek için client tarafında işlemleri gerçekleştirirken, backend yalnızca web servislerdir.
- Mikroservis mimarisi
  - Büyük sistemler yönetilebilir küçük servislere bölünebilir.
  - Yeni büyük sistemlerde oldukça önemlidir.

Resource summary

- About
- Changes
- Channels
- Comments
- Files
- Permissions
- Replies
- Revisions
- Drives
- Standard features

MISC. REFERENCE

Single-parenting behavior changes  
 Search query terms  
 Google Workspace documents and supporting MIME types  
 Shared drive versus My Drive differences

list	GET	/files/ <i>fileId</i> /comments	Lists a file's comments. Required query parameters: fields
update	PATCH	/files/ <i>fileId</i> /comments/ <i>commentId</i>	Updates a comment with patch semantics. Required query parameters: fields

## Files

For Files Resource details, see the [resource representation](#) page.

Method	HTTP request	Description
URLs relative to <a href="https://www.googleapis.com/drive/v3">https://www.googleapis.com/drive/v3</a> , unless otherwise noted		
copy	POST /files/ <i>fileId</i> /copy	Creates a copy of a file and applies any requested updates with patch semantics. Folders cannot be copied.
create	POST <a href="https://www.googleapis.com/upload/drive/v3/files">https://www.googleapis.com/upload/drive/v3/files</a> and POST /files	Creates a new file.
delete	DELETE /files/ <i>fileId</i>	Permanently deletes a file owned by the user without moving it to the trash. If the file belongs to a shared drive the user must be an organizer on the parent. If the target is a folder, all descendants owned by the user are also deleted.
emptyTrash	DELETE /files/trash	Permanently deletes all of the user's trashed files.
export	GET /files/ <i>fileId</i> /	Exports a Google Doc to the requested MIME type and returns the exported content.

#### CHAMPION MASTERY-V4

Development API key

#### CHAMPION-V3

Development API key

#### CLASH-V1

Development API key

#### LEAGUE-EXP-V4

Development API key

#### LEAGUE-V4

Development API key

#### LOL-STATUS-V3

Development API key

#### LOL-STATUS-V4

Development API key

#### LOR-MATCH-V1

Development API key

#### LOR-RANKED-V1

Development API key

#### LOR-STATUS-V1

Development API key

#### MATCH-V4

Development API key

#### SPECTATOR-V4

Development API key

#### SUMMONER-V4

Development API key

#### TFT-LEAGUE-V1

Development API key

#### TFT-MATCH-V1

Development API key

## LOR-RANKED-V1

[api](#) /lor/ranked/v1/leaderboards

The leaderboard is updated once an hour.

[Jump to inputs](#)

### RESPONSE CLASSES

Return value: LeaderboardDto

#### LeaderboardDto

NAME	DATA TYPE	DESCRIPTION
players	List< <a href="#">PlayerDto</a> >	A list of players in Master tier.

#### PlayerDto

NAME	DATA TYPE	DESCRIPTION
name	string	
rank	int	
lp	int	League points.

### RESPONSE ERRORS

HTTP STATUS CODE	REASON
400	Bad request
401	Unauthorized
403	Forbidden
404	Data not found

Documentation

Search the docs

Twitter API

Getting started

Tutorials

Tools and libraries

Migrate

API reference index

The new Twitter API

Fundamentals

Tweets

Users

# API reference index

This resource is the Twitter API-specific API reference index. If you are looking for a list of endpoints from the entire platform, including Twitter Ads API and Labs, please visit the [platform API Reference Index](#).

## Twitter API v2

### Tweets

#### Filtered stream

- GET /2/tweets/search/stream
- GET /2/tweets/search/stream/rules
- POST /2/tweets/search/stream/rules

#### Hide replies

- PUT /2/tweets/:id/hidden

#### Sampled stream

- GET /2/tweets/sample/stream



API methods	
by section	by oauth scope
account	
/api/v1/me	oauth
/api/v1/me/blocked	oauth
/api/v1/me/friends	oauth
/api/v1/me/karma	oauth
/api/v1/me/prefs	oauth
/api/v1/me/triphasis	oauth
/prefs/blocked	oauth
/prefs/friends	oauth
/prefs/messaging	oauth
/prefs/trusted	oauth
/prefs/where	oauth
captcha	
/api/needs_captcha	oauth
collections	
/api/v1/collections/add_post_to_collection	oauth
/api/v1/collections/collection	oauth
/api/v1/collections/create_collection	oauth
/api/v1/collections/delete_collection	oauth
/api/v1/collections/follow_collection	oauth
/api/v1/collections/remove_post_from_collection	oauth
/api/v1/collections/reorder_collection	oauth
/api/v1/collections/subreddit_collections	oauth
/api/v1/collections/update_collection_display_layout	oauth
/api/v1/collections/update_collection_title	oauth
emoji	
/api/v1/subreddit/emoji.json	oauth
/api/v1/subreddit/emoji/emoji_name	oauth
/api/v1/subreddit/emoji_asset_url/ad	oauth
/api/v1/subreddit/emoji_custom_emoji	oauth

This is automatically-generated documentation for the reddit API.  
Please take care to respect our [API access rules](#).

## overview

### listings

Many endpoints on reddit use the same protocol for controlling pagination and filtering. These endpoints are called Listings and share five common parameters: `after`, `before`, `limit`, `count`, and `show`.

Listings do not use page numbers because their content changes so frequently. Instead, they allow you to view slices of the underlying data. Listing JSON responses contain `after` and `before` fields which are equivalent to the "next" and "prev" buttons on the site and in combination with `count` can be used to page through the listing.

The common parameters are as follows:

- `after` / `before` - only one should be specified. these indicate the **fullname** of an item in the listing to use as the anchor point of the slice.
- `limit` - the maximum number of items to return in this slice of the listing.
- `count` - the number of items already seen in this listing. on the html site, the builder uses this to determine when to give values for `before` and `after` in the response.
- `show` - optional parameter; if `all` is passed, filters such as "hide links that I have voted on" will be disabled.

To page through a listing, start by fetching the first page without specifying values for `after` and `count`. The response will contain an `after` value which you can pass in the next request. It is a good idea, but not required, to send an updated value for `count` which should be the number of items already fetched.

### modhashes

A modhash is a token that the reddit API requires to help prevent CSRF. Modhashes can be obtained via the `/api/me.json` call or in response data of listing endpoints.

The preferred way to send a modhash is to include an `X-Modhash` custom HTTP header with your requests.

Modhashes are not required when authenticated with OAuth.

master2 branches0 tags

Go to fileAdd fileCode

valueof Updated API languageekicob on 1 Oct 202055 commits

README.mdUpdated API language6 months ago

README.md

# Medium's API documentation

This repository contains the documentation for [Medium's API](#).

## Contents

- Overview
- Authentication
  - Browser-based authentication
  - Self-issued access tokens
- Resources
  - Users
  - Publications
  - Posts
  - Images
- Testing

## 1. Overview

Medium's API is a JSON-based OAuth2 API. All requests are made to endpoints beginning: <https://api.medium.com/v1>

All requests must be secure, i.e. [https](#), not [http](#).

Developer agreement

About

Documentation for Medium's OAuth2 API

Readme


Releases

No releases published

Packages

No packages published

Contributors 17



+ 6 contributors

## 3-RESTful API

---



- Representational State Transfer (REST)
- Web servislerin en yaygın kullanılan türüdür.
- Kaynağa HTTP üzerinden erişim sağlanır.
- REST bir protokol değildir, HTTP endpointlerin nasıl tanımlanacağı ile ilgili mimari bir kılavuzdur.
  - Örnek: GET metodu ile kaynakları silmemelisiniz ancak isterseniz silmenizi engelleyen bir durum da bulunmamaktadır.
- 2000 yılında bir doktora tezinde ortaya atılmıştır.

- Tek bir arayüz
- Durumsuz (stateless)
- Önbelleklenebilir olmalı
- İstemci-Sunucu
- Katmanlı mimari
- Code on demand (opsiyonel)

- URI tarafından tanımlı, kaynak tabanlı
- Kaynak her şey olabilir
  - Örnek: SQL veritabanı satırı, bir diskteki resim dosyası vs.
- İstemci kaynağın gösterimini görebilir ve aynı kaynağa farklı formatlarda erişim sağlayabilir.
  - Ör: XML, JSON ve text gibi
- Hypermedia as the Engine of Application State (HATEOAS)
  - Kaynaklarla ilgili eylemler, cevap olarak döndürülür. Ancak çok az uygulamada kullanılıyor.

- Kaynak veritabanı veya dosyalarda tutuluyor olabilir.
- Ancak web servisin kendisi durumsuz olmalıdır.
- Yani işlemle ilgili bütün bilgiler istekle birlikte gelmelidir.
  - Ör: HTTP Header ile
- Örnek:
  - Gerçekleştirilmek istenen eylem her zaman tekrar başlatılabilir olmalıdır.
  - Yatay ölçekleme: Aynı servisin birden fazla örneği varsa hangisinin hangi sıra ile cevap verdiğinin önemi olmamalıdır.

- Önbellekleme: Bir önceki elde edilen veri henüz değişmemiş ve geçerliliğini koruyor ise yeni bir istek yapmaktan sakınmak gerekmektedir.
- Ölçeklenebilirlik için oldukça önemlidir.
- Kaynaklar ölçeklenebilir olup olmamasına göre tanımlanmalıdır.

- İstemci ve sunucu arasında oldukça belirgin bir ayrım bulunmalıdır.
- Sunucu yalnızca URI'i ve gösterimi (JSON gibi) bilmelidir. İç işleyişle ilgili bilgiye sahip olmamalıdır.
  - Ör: verinin veritabanında mı yoksa dosyada mı saklandığını dahi bilmemelidir.
- Sunucu, istemcinin veriyi nasıl kullandığını bilmemelidir.
- Sonuç olarak, sunucu ve istemci birbirinden bağımsız olarak geliştirilmeli ve güncellenmelidir. Yalnızca URI ve gösterim aynı kalmalıdır.

- İstemci için sunucuya giden yolun bir önemi olmamalıdır.
- Tipi örnek:
  - Load balancer için veya erişim politikalarına erişim için kullanılır.

- Sunucular client tarafındaki işlevselliği artırmak için bazı çalıştırılabilir kod parçaları gönderebilirler.
- REST sınırlamaları içerisinde tek opsiyonel olandır.



- Pek çok API'ya geliştiricileri tarafından REST adı verilmektedir.
  - Ancak teknik olarak REST değildir.
- Örneğin hemen hemen hiçbir API HATEOAS kullanmamaktadır.
- Bu yüzden günümüzde REST genel olarak "Bir URI ile hiyerarşik olarak kaynakların tanımlandığı ve HTTP fiil/method'ları kullanılarak bunlar üzerinde işlemlerin gerçekleştirildiği yapılar" olarak tanımlanabilir.

- Örneğin tam URL: **www.foo.com/products** olsun.
- GET **/products**
  - Mevcut bütün ürünleri döndürür.
- GET **/products?k=v**
  - Bazı parametrelerle filtrelenmiş bütün ürünleri döndürür.
- POST **/products**
  - Yeni ürün oluşturur.
- GET **/products/{id}**
  - Verilen id değerine sahip ürünün bilgilerini döndürür.
- GET **/products/{id}/price**
  - id değerine sahip ürünün fiyat bilgisini döndürür.
- DELETE **/products/{id}**
  - id değerine sahip ürünü siler.

- **/users/3/items/42/description**: kaynağını ele alalım.
- **/users**: bir kullanıcı kümesini belirtmektedir.
- **/3**: /users seti içerisinde 3 id'li kullanıcıyı belirtmektedir.
- **/items**: 3 id'li kullanıcıya ait eşya setini belirtmektedir.
- **/42**: 3 id'li kullanıcının 42 id'li eşyasını belirtmektedir.
- **/description**: 42 id'li eşyanın description alanını belirtmektedir.

- GET **/users/3/items/42/description**
- Anlamı, 3 id'li kullanıcıya ait 42 id'li eşyanın description alanını ver.
- Peki GET **/items/42/description** isteğinde bulunulsa ne olurdu?
- Teknik olarak 2 farklı URI olduğu için 2 farklı kaynak olması beklenirdi.
- Ancak pratikte aynı kaynağı işaret etmektedir.

- `/users/3/items/42/description`
- SQL veritabanında 2 farklı tablo olabilir (Users ve Items gibi)
- Veya yalnızca diskte tek bir JSON dosyasında tutuluyor olabilir.
- Veya iki farklı web servisten gelen veriler olabilir.
- Veya canınız nasıl isterse :)
- Buradaki önemli nokta, istemci bu kısımda nelerin gerçekleştiği ile ilgilenmemektedir.

- 1) GET **/users/3/itemIds**
- 2) GET **/items/42/description**
- 1.si 3 id'li kullanıcıya ait bütün eşyaların id'lerini döndürür. Ardından spesifik bir eşyanın description alanını almak için 2. istek atılır.
- Ancak 2. GET yerine aşağıdaki istek atılsa ne olurdu?
  - **/users/3/items/42/description**

- 1.) GET **/users/3/itemIds**
- 2.) GET **/items/42/description**
- 3.) GET **/users/3/items/description**
- 2. veya 3. endpointe ihtiyaç olup olmadığı tamamiyle istemcinin API ile olan etkileşimine bağlıdır.
  - Kullanıcıdan bağımsız olarak eşyalara erişim gerekiyor mu?

# Yol elemanları (Path Elements)

- `/users/3/items/42/description`
- İstemci `/users` veya `/items`'in bir küme olduğunu ancak `/description`'in olmadığını nasıl anlayabilir?
- "Teknik olarak" her biri de birer yol elemanıdır.
- İstemci API'ya ait dokümantasyonu okumak zorundadır.
- Ancak bunu daha anlaşılabilir kılmak için kümeye ait isimleri çoğul hale getirmek bir **kuraldır**.



- Türkiye'de bulunan bütün kullanıcıları çekmek istediğimizi varsayalım.
- 1.) GET **/users/inTurkey**
- Problem, tek bir id'ye sahip kullanıcı çekmek istesek ne olacaktı?
- 2.) GET **/users/{id}**
  - {} içerisinde bulunanlar değişken parametrelerdir.
  - Belirsizlik: **/users/inTurkey** her iki endpoint ile de eşleşmektedir.
    - inTurkey burada kullanıcı id'si olarak işlem yapar

- 1.) GET **/users/inTurkey**
- 2.) GET **/users/byId/{id}**
- Burada belirsizlik kalkmaktadır ancak,
- Ara kaynakların semantiği kaybolmaktadır (**/users/byId ???**)
- Ayrıca URI yalnızca kaynağı temsil etmelidir, o kaynağa yapılacak eylemleri değil.

- 1.) GET `/users?country=turkey`
- 2.) GET `/users/{id}`
- Bir kümeye filtre uygulanmak istendiğinde, sorgu (query) parametreleri kullanılmalıdır.
- Bu sayede endpoint değişmeden ekstra filtre seçenekleri de (örnek: `ageMin=42` gibi) eklenebilir.

- **POST /users**
  - POST ile gerçekleştirilir.
  - Yeni kaynak oluşturmak için gönderilen Payload kullanılır.
  - Gönderilen cevapta location HTTP header'ı oluşturulan yeni kaynağın lokasyonunu belirtir (ör: Location: /users/42)
- **PUT /users/42**
  - URI'de belirtilen yeni kaynağa doğrudan PUT işlemi uygulanır.
  - Spesifik bir id'ye ihtiyaç bulunmaktadır.

- POST veya PUT, hangisi kullanılmalı?
- Eğer Id alanı sunucu tarafından seçilecekse POST
- Eğer istemci tarafından id seçilecekse PUT ancak bu değer tekil olmalıdır.
  - Eğer aynı id'ye sahip kaynak varsa üzerine yazacaktır.

- 1.) GET **/users/42** → Response 404
- 2.) PUT **/users/42**
- Bunun bir anlamı bulunmamaktadır çünkü:
  - Bir 404 hatası bulana kadar çok defa çalıştırmak gerekebilir.
  - 404 bulduktan sonra PUT çalıştırılana kadar bir başka istekte kaynak oluşturulmuş olabilir bu yüzden üzerine yazılır.

- 1.) POST **/users** → Location: **/users/42**
- 2.) PUT **/users/42/address**
- POST ile adres bilgisi dolmamış yeni bir kullanıcı oluşturduğumuzu varsayalım.
- PUT kullanarak adres bilgisini doldurabilirsiniz.
- Ancak çoğu zaman PATCH ile bu işlem gerçekleşmektedir çünkü PUT ile bütün verileri göndermek gerekir.
  - Örnek: PATCH **/users/42**

- 1.) GET /users/42
- 2.) GET /users/42.json
- 3.) GET /users/42.xml
- Bilindiği üzere, REST servisler verilerini SQL veritabanında veya bir csv dosyasında depolayabilir.
- Gelen kaynak bilgisi istemcinin ihtiyacı olan veri türüdür.
- Ancak burada problem yaratan nedir?



- 1.) GET /users/42
- 2.) GET /users/42.json
- 3.) GET /users/42.xml
- Çünkü URI farklıdır ve teknik olarak 3 farklı kaynak bulunmaktadır.
- URI içerisine ".json" eklemek anlamı (semantik) değiştirmez.

- GET `/users/42`
- Kaynak içerisine bu türde uzantı yazmaktan kaçınmalısınız.
  - Pek çok API'ın böyle yaptığını görebilirsiniz.
- Farklı tipte kaynak istersen HTTP header içinde (accept gibi) belirtmelisiniz.
  - Örnek: "Accept: application/json"
- İstemci özel bir tip istediğinde (XML gibi) sunucunun bunu destekliyor olduğu anlamına gelmemektedir.
- Eğer Accept header'ı gönderilmez veya genel (`/*/*`) gönderilir ise varsayılan gösterim ile (Örnek: JSON) cevap döner.

## 4-Statik ve Dinamik Kaynaklar

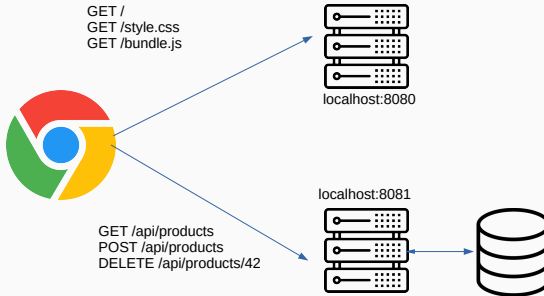
---

- HTML dosyaları
  - SPA'larda genellikle index.html
- CSS
- JavaScript kaynak kodları
  - Örnek: bundle.js
- Resim, doküman veya indirilebilir diğer tipte dosyalar
  - Örnek: PDF
- vs.

- Zaman içerisinde genellikle değişen verilerdir.
- Kullanıcı etkileşimine bağlıdır.
  - Hesap oluşturma, alışveriş sepeti vs.
- Uzun zamanlı depolama alanlarında (SQL veya NoSQL veritabanları gibi) saklanırlar.
- REST'te genellikle bu kaynaklar JSON formatında gösterilir.
- Ele alınması ve oluşturulması genellikle backend adı verilen iş katmanında gerçekleştirilir.

- Genellikle statik kaynaklar web uygulamanın frontend tarafında tanımlanır.
  - HTML/CSS/JS/görüntüler/vs.
- Backend JSON formatında veritabanındaki verileri sağlayan bir sunucudur.
- Backend çeşitli programlama dilleri ile yazılmış iş katmanındaki bir süreçtir.
- Hem statik hem dinamik kaynaklar için HTTP kullanılır.

- Frontend (React uygulaması vs.) statik kaynaklar HTTP sunucu tarafından sağlanır.
- Backend tarafındaki iş katmanı ve veritabanına erişim de HTTP sunucusu ile sağlanır.



- Frontend ve backend aynı HTTP sunucuda barınabilir.
  - Bu CORS problemini (bir sonraki ders ele alacağız) de engelleyecektir.

