Fundamental Internet Concepts (pdf)

1. What is Web?

Internet Protocol (IP) global network. When we talk about a network, our nervous system comes to mind. Similar to this, people's neural networks transport information/sense/information to the brain, where commands are then used to control muscles and joints. The **internet** is the network in question here.

In other words, networks connecting computers. The information system known as the Web is created by transferring data between interconnected hypertext that are published on this network.

By transferring hyper-text, or metadata, between networks, such as links, paragraphs, headings, etc., a huge information network is built. Hyper Text Markup Language (**HTML**) is the language that will make sure that it is transmitted throughout this transport in a certain markup format. Hyper Text Transfer Protocol (**HTTP**) is the name of this language's computer-to-computer transport protocol.

This is how the request response method is used by the client, which is your browser, to connect with the server. The technical specifics will be covered later.

It was necessary to establish a number of standards in order for the Web to emerge and grow. The **W3C** Consortium is in charge of developing these standards. Because it is exceedingly challenging to run the same web page on numerous different devices, operating systems, browsers, and clients, these standards are crucial to the evolution of the web. The standards guarantee that these pages will function in many browsers.

We can access and use HyperText files from anywhere in the world thanks to software like **Firefox, Opera, Edge, Safari, Chrome, Safari** that can read and use these standards. These programs are referred to as browser applications.

Of course, the standards listed above are not static; they are continually changing and evolving to meet the web demands of humans, machines, bots, and robots. The list developed for this development is shown below;

Web 1.0. (Publishing Web)

The goal in the early days of the Web was to develop flat static sites and communicate Text, Audio, Image, and Video content across them. The user could not interact with the web page under this system and could only view and listen to the data it displayed.

Web 2.0. (Social And Co-Created Web)

It was created to allow users to interact with the website more.

- **Rich Internet Application (RIA)** It was wanted to construct Web applications, and the visuals and interactions of these programs should be such that they did not seem like Desktop applications. HTML5, CSS3 standards were developed for this purpose.
- **WOA (Web Oriented Architecture)**, RSS, Web-Service, and Mashup architectures were developed to allow Web-based applications to communicate with one another and create richer and more remarkable applications.
- **Social Web:** Websites like Facebook, Twitter, game platforms, and others have begun to build infrastructures that will allow users to communicate more and more

intensely with one another in order to allow users to interact and exchange information with one another online.

Web 3.0. (Semantic & Intelligent Web)

Web 3.0, also known as Semantic Web. The number of devices linked to the Internet via the Internet of Things is estimated to reach approximately 50 billion. Because the existing web is too complicated for them to interact with one another, the web's vocabulary must be translated into data formats that unambiguously describe things, people, and events in a way that machines can comprehend. These must be more uniform and open in format.

For example, if a passenger in seat 12 of a plane forgets to buckle his seat belt, he may quickly chat to the seat-related screen and notify the person seated opposite him. This requires semantically carrying data such as seat, belt, and belt tightening time within the Web.

Of course, in the future, systems and software that process this data intelligently will be developed. As a result, the age of Artificial Intelligence will arrive. (Even now)

MetaData, Vocabulary and Syntax

Metadata is the data, information and docs. It can also be interpreted as metadata.

You're holding a book, this book;

- Type,
- Name,
- Who wrote it, when it was written, how many pages it has, and so on.
- How many editions of the book are there?

is a description of metadata. With this metadata, we can get the information we want more quickly by making book cards in the library.

This procedure may be used to papers, web pages, and media in an online context. In today's environment, when material is continually being created and consumed, it is critical to specify the metadata of the content.

This meta-data is utilized to interpret and comprehend the HTML page in programs such as search engines, other mails, helpers, and so on.

- more information
- in less time
- more accurately

By shifting services to software, higher quality services may be given.

The capacity of artificial intelligence to detect and analyze data like humans, particularly now, permits the construction of remarkable automation systems and their integration into business structures.

Vocabulary

It creates Meta-Data structures and vocabulary.

schema.org: Other search engines created by Google, Bing, Yandex, Yahoo etc. supported by many websites

opengraphprotocol.org: Facebook created vocabulary

Vocabulary such as Dublin Core Schema, FOAF, SIOC are available.

Syntax

It is possible to write these metadata with different syntaxes. The most popular ones are MicroData, JSON-LD, RFDa.

In the image below, you can see how the metadata of the Avatar film is made in different formats.

Microdata [edit]

The following is an example[19] of how to mark up information about a movie and its director using the schema.org schemas and microdata the scope of the itemtype. The kind of the current item can be defined by using the attribute `itemprop`.

```
<div itemscope itemtype="http://schema.org/Movie">
  <h1 itemprop="name">Avatar</h1>
  <div itemprop="director" itemscope itemtype="http://schema.org/Person">
  Director: <span itemprop="name">James Cameron</span>
(born <time itemprop="birthDate" datetime="1954-08-16">August 16, 1954</time>)
  </div>
  <span itemprop="genre">ali</span>
  <a href="../movies/avatar-theatrical-trailer.html" itemprop="trailer">Trailer</a>
</div>
```

RDFa 1.1 Lite [edit]

```
<div vocab="http://schema.org/" typeof="Movie">
  <h1 property="name">Avatar</h1>
  <div property="director" typeof="Person">
  Director: <span property="name">James Cameron</span>
(born <time property="birthDate" datetime="1954-08-16">August 16, 1954</time>)
  </div>
  <span property="genre">Science fiction</span>
  <a href="../movies/avatar-theatrical-trailer.html" property="trailer">Trailer</a>
</div>
```

JSON-LD [edit]

```
<script type="application/ld+json">
{
  "@context": "http://schema.org/",
  "@type": "Movie",
  "name": "Avatar",
  "director":
    {
       "@type": "Person",
       "name": "James Cameron",
       "birthDate": "1954-08-16"
    },
  "genre": "Science fiction",
  "trailer": "../movies/avatar-theatrical-trailer.html"
}
</script>
```

Microdata, RFDa, JSON-LD

What is Web 3 ?

> Web 3.0 (Semantic and Intelligent Web) The number of devices linked to the Internet via the Internet of Things is predicted to be in the billions (50 billion by 2020). Because the existing web is too

> complicated for them to interact with one another, the web's vocabulary must be translated into data formats that unambiguously describe things, people, and events in a way that machines can comprehend. These should be more consistent and open in format.

Web3.0 departs from its preconceived conceptual framework in that it grows and progresses in an organic manner. In other words, it is growing in response to market need and popularity.

Note: Cryptocurrency NFT, Cryptocurrencies were quite popular when I published this post in April 2022. **AI (Artificial Intelligence)** gained its place when we approached 2023. Although cryptocurrency is somewhat in the background, all of the themes covered are highly essential and will have a significant influence on the shape of the web.

- Cryptocurrency derivatives such as Bitcoin, Ethereum, and other digital money are now widely used, and numerous technologies have been built around them. At the same time, it has compelled governments to rethink the ideas of banking and money.
- Blockchain, the technology that underpins bitcoin, is now available in a variety of fields other than digital currencies. For instance, NFT (Non-fungible token)
- In particular, the use of personal data by social media, as well as the logic of creating adverts and content through it, is a call for personal data protection.
- As a result of virtualisation, MetaVerse, VR/AR glasses, and the expansion of the digital market and money, the need is expanding.

As a result, the Web will continue to evolve and develop in line with these needs. This new structure is also

> It will also facilitate new structures in which decentralized structures built on blockchain communicate with one another via token-based economies.

What therefore should we comprehend about Web 3.0? You might consider it a component of Web3 and Web3.0. Future technologies will allow the physical world to become increasingly digitized and virtualized in every way. Services that are more dispersed and accessible to users will replace centralized systems. Services built with code over static data will transform into services backed by artificial intelligence and taught with data.

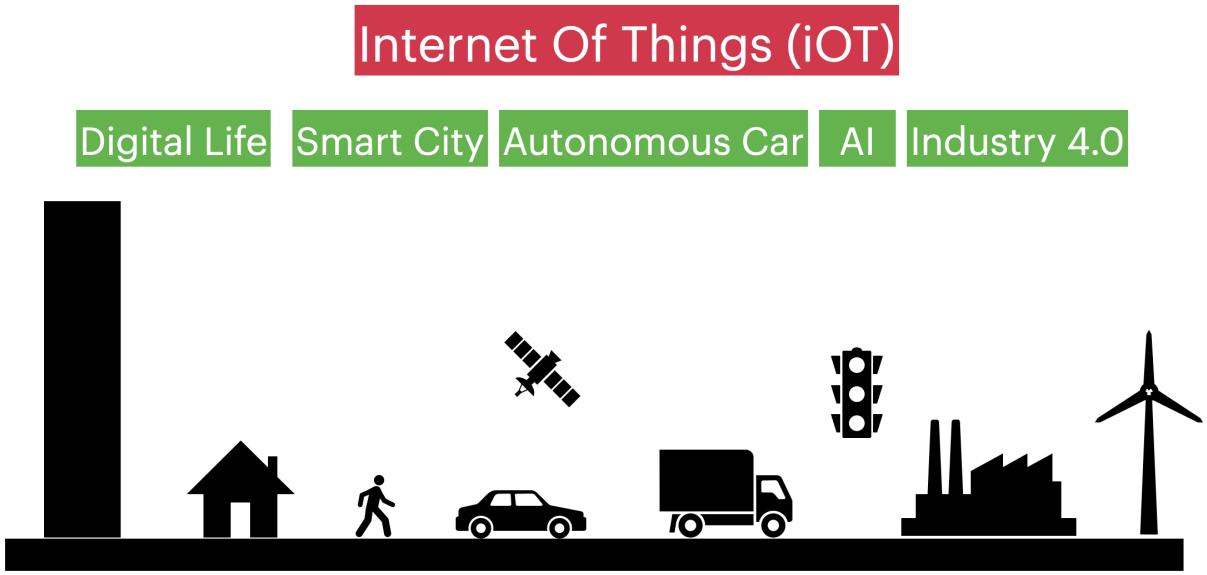
Web 3.0 Services

The areas I've included below contain a wide variety of sectors.

- Services Supported by Artificial Intelligence
- Cryptocurrency Services,
- NFT
- VR and AR (Virtual Reality and Augmented Reality),
- and Edge Computing Technologies

Web 4.0. (The Mobile, Machine and Object Web)

While the content between Web 1.0 and 3.0 was mostly data created by people, we are now entering a period in which information produced by devices, tools, and objects will be included in the common network thanks to wireless technologies, as well as the inclusion of unreal virtual world objects in this network, and the content created will be usable by the connections in this network.



For example, your Self-Driving/Driverless (Autonomous) vehicle will send data to the cloud in advance, where it will be analysed together with other vehicle data to establish the best path for you. You will be able to reach your goal faster and with less energy this way.

Web 4.0 is the phase generated by the proliferation of human-created content and the usage of information from smart devices and commodities in the real and virtual **** worlds by connecting this network.

This framework will serve as the foundation for Digital Life, Smart Cities, Industry 4.0, and Future Travel / Mobility. Of course, not only the HTTP protocol, but also MQTT and other protocols, will be used more often in this IOT framework, that is, for smaller but more frequent data transfers.

Web 5.0. (Sensory-Emotive Web)

The internet is an emotionally bland and stagnant platform. Although emojis are used to depict people's emotional states in blog posts, comments, YouTube, and Instagram, this is not real-time information. The increase in sensor data, for example, understanding your mood from your tone of voice while talking to Alexa and playing songs accordingly, smartwatches measuring your heartbeat or blood pressure and recommending sports accordingly, means that the web will make decisions and answers based on people's or human communities' emotional states. At this point, I believe we will begin a symbiotic life cycle of Web, Human, and Robots.

2.Internet Concepts

What’s RFC ?

We stated that there should be certain web development standards.

We will provide internet information such as DNS, HTTP, TCP, URL, and so on in the future. But, before the **RFC** (Request For Comments), **IETF** (Internet Engineering Task Force), who decides on these protocols? We must concentrate on this matter.

RFCs are internet standards and protocol technical papers. Following the establishment of these standards, new versions are papers that are issued in response to new requests that cannot be modified since technologies are based on it. Here are several RFC examples.

- **RFC2616:** (HTTP) Hypertext Transfer Protocol — HTTP/1.1
- **RFC1034:** (DNS) Domain Names
- **RFC791:** (IP) Internet Protocol
- **RFC821 :** (SMTP) Simple Mail Transfer Protocol
- **RFC6749:** The OAuth 2.0 Authorization Framework

ve tüm RFC indeksine bu adresten ulaşabilirsiniz.

The team in charge of this RFC procedure is known as the IETF (Internet Engineering Task Force).

What is HTML, HTTP and Hyper Text?

We discussed web development. Web 1.0, 2.0, and 3.0... This internet network is built on several fundamental notions, and the internet operates on these foundations. Hypertext, HTML, and HTTP are three of the most essential of these principles.

- that **Hyper Text** is used for the transport of data
- the markup language used for this, **HTML**
- The protocol for sending data is called **HTTP**.

This is how the vast bulk of the Internet works. So, in this section, I'll try to describe how this system works, what DNS is, and what it accomplishes.

What is IP?

Every computer with an Internet connection has at least one IP address. For example, when we input google.com.tr into any browser, the **Google** address that everyone knows reaches a number system in the background, such as our home address or postcode number. This numerical access address system that computers use to communicate with one another in the internet environment is referred to as IP in this context.

How do the web addresses we create become IP addresses?

Simply ping an address to gain access to its IP address. An example of this is shown below.

When we enter ping learnreactui.dev into the terminal, we can see that the address is really IP transformed in the background and attempts to reach through this IP. The address **76.76.21.61** that you see below is the actual address that computers use in the background to visit the LearnReactUI dev web page.

```
onurdayibasi@onurs-MacBook-Pro ~ % ping learnreactui.dev
PING learnreactui.dev (76.76.21.61): 56 data bytes
64 bytes from 76.76.21.61: icmp_seq=0 ttl=118 time=58.392 ms
64 bytes from 76.76.21.61: icmp_seq=1 ttl=118 time=60.948 ms
64 bytes from 76.76.21.61: icmp_seq=2 ttl=118 time=67.996 ms
64 bytes from 76.76.21.61: icmp_seq=3 ttl=118 time=62.463 ms
64 bytes from 76.76.21.61: icmp_seq=4 ttl=118 time=61.774 ms
64 bytes from 76.76.21.61: icmp_seq=5 ttl=118 time=61.060 ms
64 bytes from 76.76.21.61: icmp_seq=6 ttl=118 time=61.628 ms
64 bytes from 76.76.21.61: icmp_seq=7 ttl=118 time=68.666 ms
64 bytes from 76.76.21.61: icmp_seq=8 ttl=118 time=86.431 ms
64 bytes from 76.76.21.61: icmp_seq=9 ttl=118 time=101.205 ms
64 bytes from 76.76.21.61: icmp_seq=10 ttl=118 time=73.025 ms
64 bytes from 76.76.21.61: icmp_seq=11 ttl=118 time=79.441 ms
64 bytes from 76.76.21.61: icmp_seq=12 ttl=118 time=74.434 ms
```

ping to Learn React UI

IP addresses

- **iPV4** (93.184.216.119)
- **32-bit :** 128-bit addressing has evolved to enhance the number of addressing that is insufficient with the connecting of mobile devices and objects to the internet.

What is a Port?

If we need to discuss Application Layer Protocols on some of the previously listed TCP and UDP protocols, such as **HTTP, SMTP, FTP, DNS**.

You require the IP address of a computer and the port where this application server is executing to reach a machine on the other side; you connect with these protocols over these ports. The standard addresses for these ports are as follows.

Web: HTTP port: 80

Secure Web: HTTPS port: 443

Secure Shell: SSH port :22

Simple Mail Transfer Protocol: SMTP port : 25

File Transfer Protocol: FTP port :21

DB access: MySQL port : 3306

Cache access: Redis port : 6379

DNS Query: DNS UDP port : 53

Queue: ActiveMQ port: 61613

Port: A port is a communication endpoint used in computer networking to send and receive data between two or more devices through a network. The opposing program sends data to us along with its IP address and port number.

Assuming we are in front of a Web server, we can determine whether port 80 is open or not by attempting to connect to it using telnet.

For instance, you wish to access the **learnreactui.dev** website. When you enter http:// or https:// in the browser in this situation, you are essentially telling the browser that you want to view this page via the IP address in the background on port 80 or 443.

The Telnet protocol can be used to connect to a distant system without the requirement for a browser. As you can see below, we are unable to telnet into port 81, however **learnreactui.dev** is able to connect to ports 80 and 443.

```
onurdayibasi@onurs-MacBook-Pro ~ % telnet learnreactui.dev 80
Trying 76.76.21.142...
Connected to learnreactui.dev.
Escape character is '^]'.
^C^Cq
Connection closed by foreign host.
onurdayibasi@onurs-MacBook-Pro ~ % telnet learnreactui.dev 81
Trying 76.76.21.93...
^C
onurdayibasi@onurs-MacBook-Pro ~ %
onurdayibasi@onurs-MacBook-Pro ~ %
onurdayibasi@onurs-MacBook-Pro ~ % telnet learnreactui.dev 443
Trying 76.76.21.93...
Connected to learnreactui.dev.
Escape character is '^]'.
```

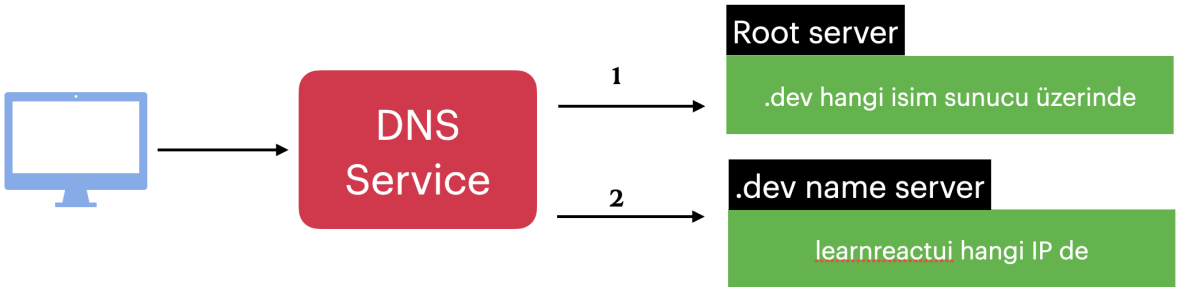
query telnet port

What is DNS ?.

Domain Name System is an acronym for Domain Name System. Individuals cannot be expected to recall computer IP addresses and understand what these numbers imply. Instead, it attempts to avoid IP confusion for the user by preserving a domain name in exchange for the IP, such as [https://learnreactui.dev/].

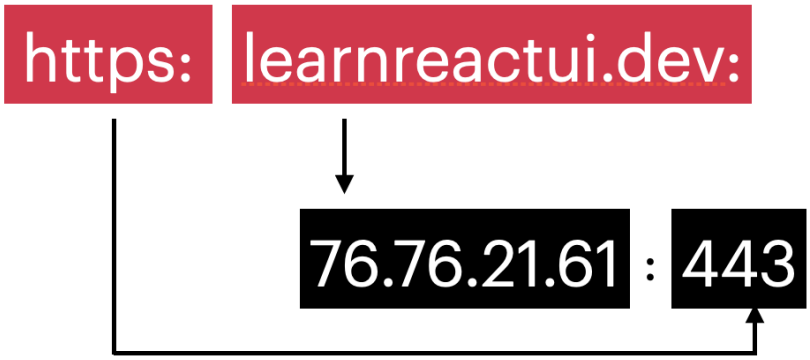
So, how does this domain name become an IP address? As seen below, the DNS Service first informs the name server of the location of the **.dev** table, after which the IP address corresponding to **learnreactui** is returned to the browser. This address/server is reached by the browser.

Name servers are available on the network for this purpose. These name logs and DNS entries are shared and synchronized by name servers.



DNS Name Query

As seen in the diagram above, we must employ a variety of services to obtain the IP address. This service is provided by the DNS Service. When you input a URL into your browser, it first determines which name server it is on based on **.dev .com .net** through DNS service, and then looks for the domain name you typed using this name server. For example, the IP address 76.76.21.61 is returned for **learnreactui.dev**. Now that the crawler knows the IP address, it may connect to the other machine in front of the https -> 443 port.



DNS Dönüşümü

These name servers are typically offered by Domain Hosting or Cloud Service providers, Universities, and so on.

Some of the Sample Name Servers;

- ns-4.awsdns-00.com **AWS** Sample Name Server
- ns1.BlueHost.com **Bluehost** Sample Name Server

It is possible to subject many sorts of documents to these name servers. These are the records that describe what the name server's responses to your requests are.

Sample DNS Records

So what records are made to these name servers?

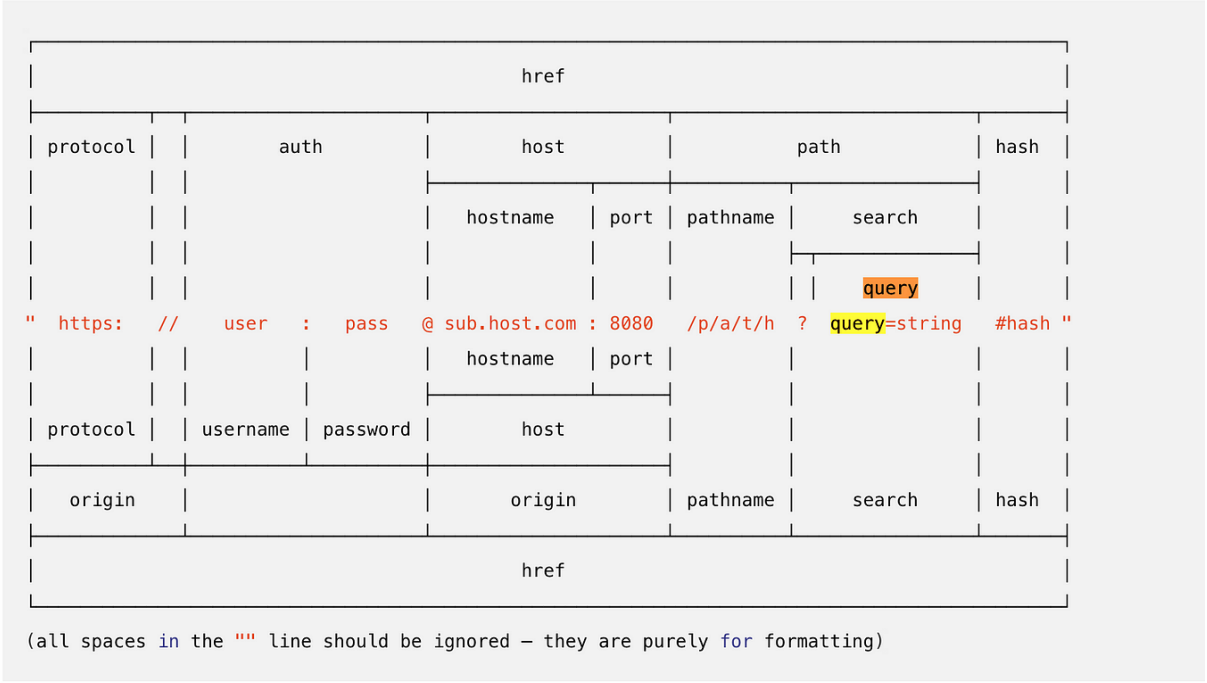
- **A Record:** is an iPv4 record. http://medium.com → 93.184.216.119
- **CName Record:** Cononical Name record. www, imap, etc. subdomain is given according to the domain name.
- **MX Record:** Mail server record. @ → mail.gmail.com
- **AAAA Record:** is an iPv6 record. http://medium.com → 2606:2800:220:6d:26bf:1447:1097:aa7
- **SOA Record :** Start Of Authority record.
- **TXT Record:** Text records that have nothing to do with traffic and that people can read.
- **NS Record:** Name server record. For example →ns-4.awsdns-00.com
- **SRV Record:** It is the record added for VoIp, IM, XMPP services in another location → _xmpp-server._tcp.gmail.com. IN SRV 5 0 5269 xmpp-server.l.google.com.
- **PTR Record:** Reverse DNS record. It is used to say that the name of this IP address is this Domain name.

What is URL

URL (Uniform Resource Locator) is an address identification system used on the internet to access resources (resources) such as text, graphics, code, CSS, and so on.

URL Model

WHATWG URL's `origin` property includes `protocol` and `host`, but not `username` or `password`.



Parsing the URL string using the WHATWG API:

https://nodejs.org/docs/latest/api/url.html

```
http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#SomewhereInTheDocument
```

Domain Name: We said that in the background servers access each other through IP records on NameServer. So **www.example.com** is a server offered at 93.184.216.34 in the background. When we type this Domain Name in the browser or URL with code from the background, the name servers convert it to IP while the network is progressing, allowing the data to progress.

Port: means the door on the server. In this way, more than one server can be served over the same IP. For example, **80 HTTP, 443 HTTPS, 25 SMTP, etc.**. different services can be provided over the same server.

Path: It is a static Web page address or API address on the service offered. (**path/to/myfile.html)**

Path/Query Parameters: ?key1=value1&key2=value2 this URL can be updated according to the content and dynamic parameters of the page to be created by passing a number of parameters. For example, if myfile.html is an html that prints the user information on the page, if key1 is the user id, the server can display the same page with different content using this information.

Anchor: #somewhere The hash after the hash is used to show a scroll or fragment to a certain part in the document. In the early stages of ClientSideApp, since this part did not change the server, they were providing SPA (Single Page App) by updating the browser by pulling user information with AJAX without going to the server again using **#username**. **Hash # concept** was used frequently in the past for this reason. Thanks to HistoryAPI pushState, React, Vue, Angular, Ember can now perform Single Page App without using hash

3. Network Concepts

In this part, I will first discuss the TCP/IP Model and the Internet, followed by Transport and Application Layer Protocols in the model's layers, and lastly several HTTP communication techniques.

- OSI Model Nedir
- TCP/IP vs OSI Model
- IPv4, IPv6
- TCP, UDP
- HTTP, SMTP, IMAP, POP3, FTP, MQTT, AMQP
- SOAP, RPC, REST, GraphQL, Webhook

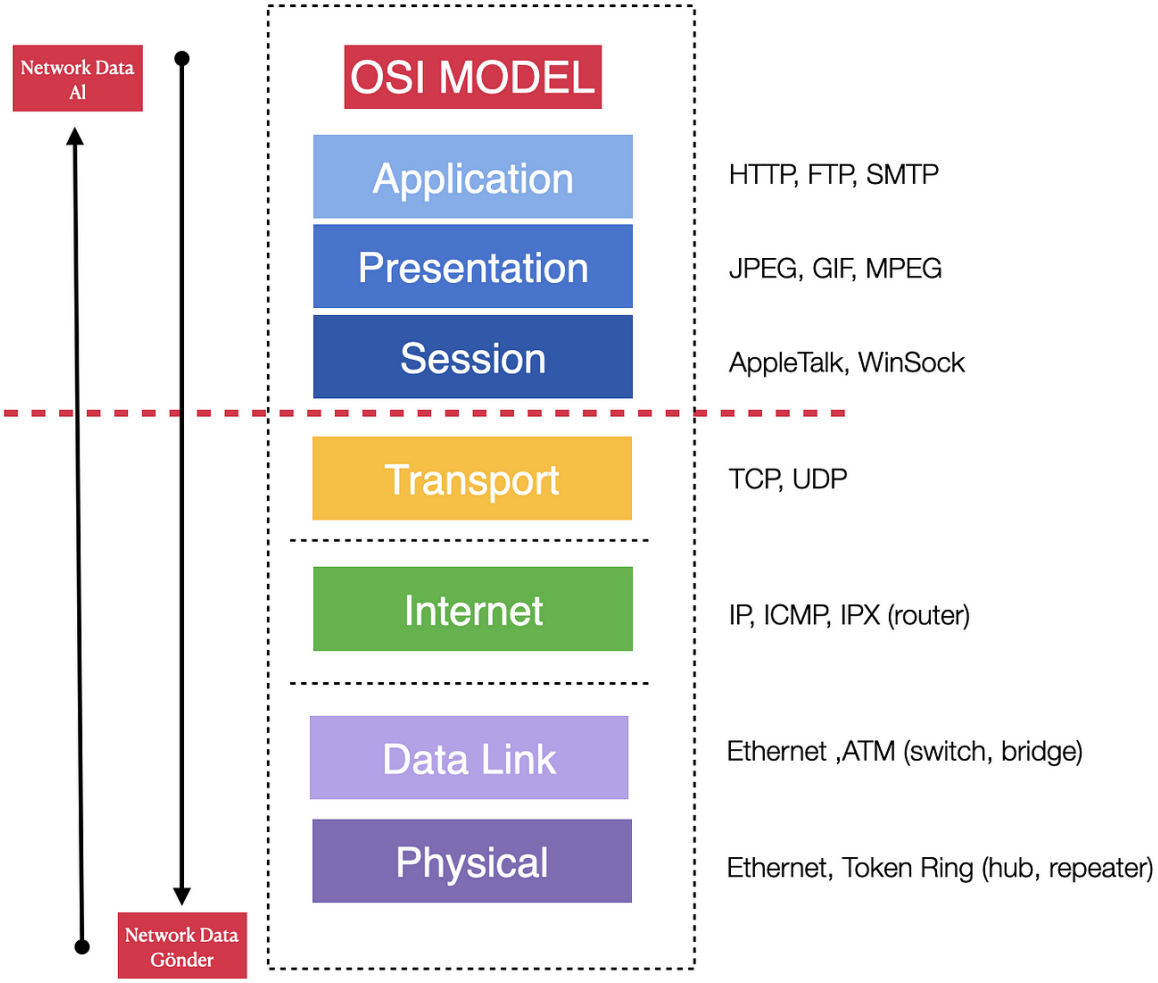
What is the OSI Model?

To better comprehend network principles, we employ OSI (Open Systems Interconnection) Layers. It enables us to more clearly see the notions that we will discuss below in our brains and to manage these concepts inside the layers in a more remembered manner.

Any Internet communication is routed through these protocols.

- user to network,
- to the other computer over the network
- and layers over it

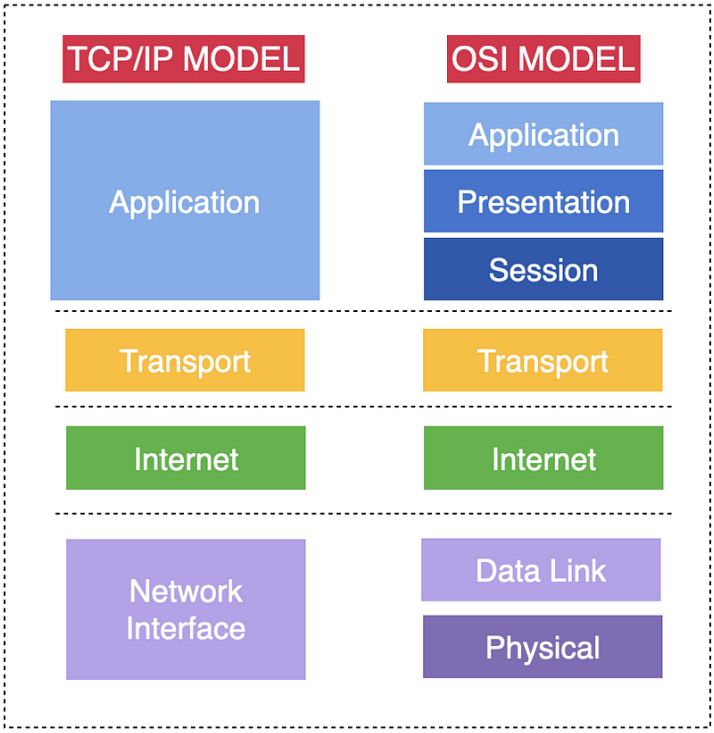
and delivered to the user.



OSI Layers

We can say that the current realisation of the OSI Model is TCP/IP. Since the structure is already very similar to each other, we can now make our conversations over TCP/IP Model.

TCP/IP vs OSI Model

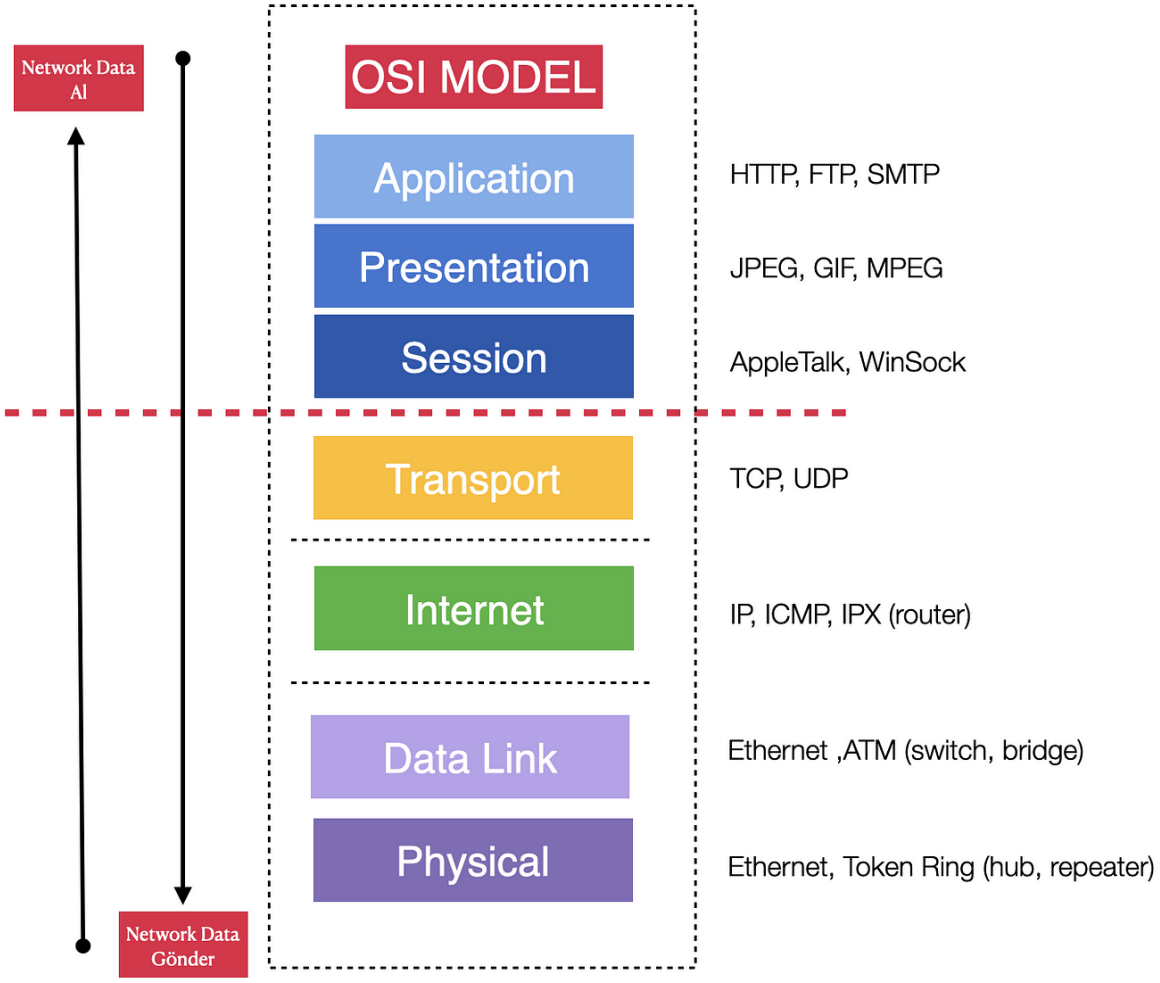


TCP/IP vs OSI

The concept of **TCP/IP** and **OSI** model are 2 networking methods accepted to provide communication. While the OSI model is more of a **conceptual model**, TCP/IP is the method used to provide **communication** in practice today.

When we look at the OSI Model from bottom to top

- At the network layer you see **IPv4 vs IPv6** addressing.
- Transport layer **TCP, UDP** protocols
- and finally in the Application layer you see **HTTP, FTP, SMTP, AMQP** protocols.



OSI Katmanları

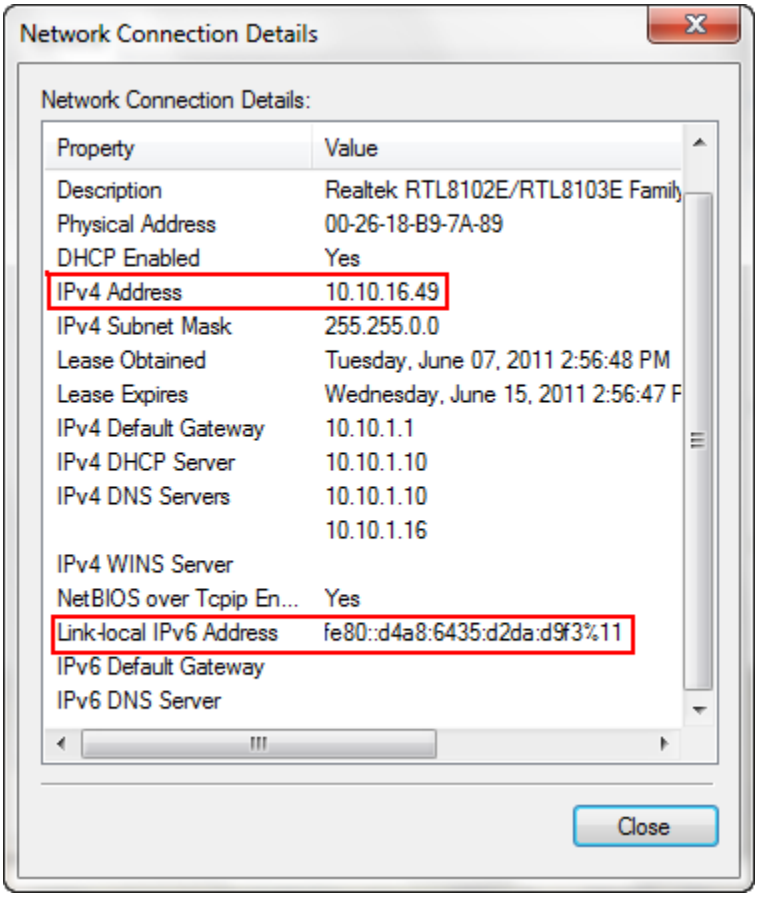
IPv4 , IPv6. (Network Layer)

IPv4 and IPv6 are the methods used for addressing computers at the network layer. While **iPv4** connects 4.3 billion devices with **32-bit addressing**, **iPv6** will have an infrastructure that supports a much larger number of devices with **128-bit addressing**.

This provides an infrastructure that can meet the increasing number of mobile iOT devices connected to the internet.

iPv6 provides not only **scalability** with addressing but also **security** with end-to-end encryption and **connectivity** without solutions such as NAT for the formation of the iOT

environment.



IPv4 vs IPv6

TCP , UDP (Transport Layer)

Transport layer

- **TCP (Transmission Control Protocol)** ,
- **UDP (User Datagram Protocol)**

When we examine them, you will see that these are protocols that allow the data to be sent to the relevant address after the data routes and destination addresses are defined.

TCP/UDP are two different approaches.

TCP puts an Acknowledge Segment in the data so that the data can reach the relevant address in a completely secure way. HTTP, HTTPS, POP3, SMTP, FTP, SFTP protocols

are realised by **Three-Way Handshake** over TCP. two computers or devices perform these operations by agreeing with each other, ensuring the correctness of this communication.

In **UDP**, it is not checked whether the packet has arrived or not. The aim is to deliver data using speed and less bandwidth. Protocols such as TFTP, SNMP, VoIP perform these operations over UDP.

You can see the differences of these two protocols in the picture below. TCP is a more reliable, sequential protocol that checks whether the data is going correctly, UDP is a faster and dynamic protocol.

Feature	TCP	UDP
Connection Status	Requires an established connection to transmit data (connection must be closed after transmission is complete)	Connectionless protocol that does not contain any requirements for opening, maintaining or terminating a connection
Data Sorting	Yes	No
Delivery Guarantee	Yes	No
Data Resend Resend for Lost Packages	Yes	No
Error Checking	There are extensive error checking and data validation mechanisms.	Checksums kullanan temel hata kontrol mekanizması var.
Transfer Methods	Data is read as byte stream; Messages are forwarded to segment boundaries	UDP packets with defined boundaries; sent individually and checked for integrity on arrival
Speed	Slower than UDP	Faster than TCP
Broadcasting	No	Yes
Usage Protocols	HTTPS, HTTP, SMTP, POP, FTP, vb	Video conferencing, streaming, DNS, VoIP, etc.

As you can see from the above comparison, TCP has a more controlled structure, while losing capabilities such as speed and brodcasting,

UDP is also used more on streaming Video Conferencing, Speech etc... while you need to manage control and data losses on the extra application side.

2 protocols have their own plus (+) and minus (-) aspects.

HTTP, SMTP, IMAP, POP3, FTP, MQTT, AMQP (App Layer)

In the Application Layer, there are different protocols such as **HTTP, SMTP, IMAP, POP3, SFTP, FTP/S, MQTT, AMQP**. From these protocols;

- **HTTP/S:** Web Browser, which is used to provide communication between Mobile and Web server, are protocols that generally use the (80/443) port.
- **SMTP, IMAP, POP3:** These are the protocols used for access by email clients.
- **FTP/S, SFTP (SSH):** are protocols that provide file transfer.
- Protocols such as **AMQP, MQTT, STOMP:** are protocols created for messaging purposes.
- **MQTT(Message Queue Telemetry Transport)** Machine-to-machine (M2M)/"Internet of Things" connectivity protocol is the protocol that provides fast data exchange for IOT.
- **AMQP (Advanced Message Queuing Protocol)** determines the communication methods of your servers/workers with middleware. It ensures that they can work together and securely. RabbitMQ, ActiveMQ etc..
- **STOMP** (Simple/Streaming Text Oriented Messaging Protocol)

HTTP Request and WebSocket

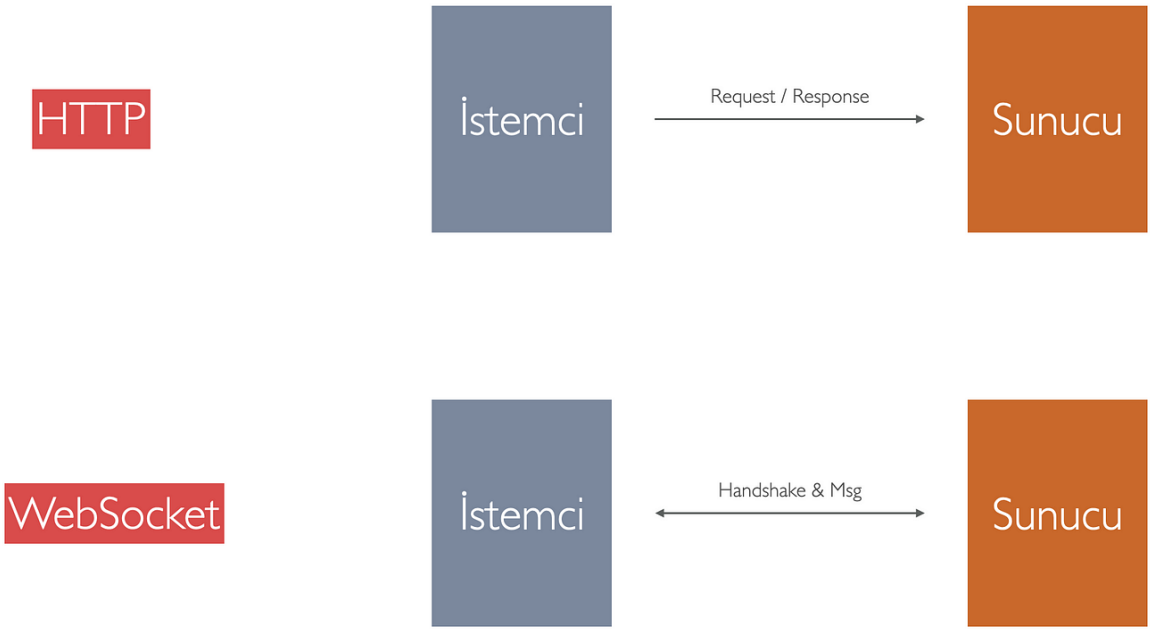
Browsers and mobile devices allow us to develop many different applications on them by communicating with servers.

For example, email tools, game, E-Commerce site, Netflix, Spotify, etc... In each application, we realise our applications by connecting to the servers from the devices we use.

I will be talking about 2 methods here.

- HTTP

- WebSocket



HTTP ve Websocket

As seen in the picture above, HTTPRequest is **one way**, that is, only the client sends some requests to the server and the server responds according to these requests,

WebSocket can send messages from the server to the client without waiting for the client to send a message, so there is a **bidirectional** triggering.

Areas of Use

HTTP Request / Response

It is the communication method used when we want to pull and show some data, lists from the server on the client, or create a record, place an order, press the OK button. Currently, in most of the Web / Mobile applications, the lists and data you see on your screen or when you place an order from here, etc... It is the client / server communication method operated on 90% pages.

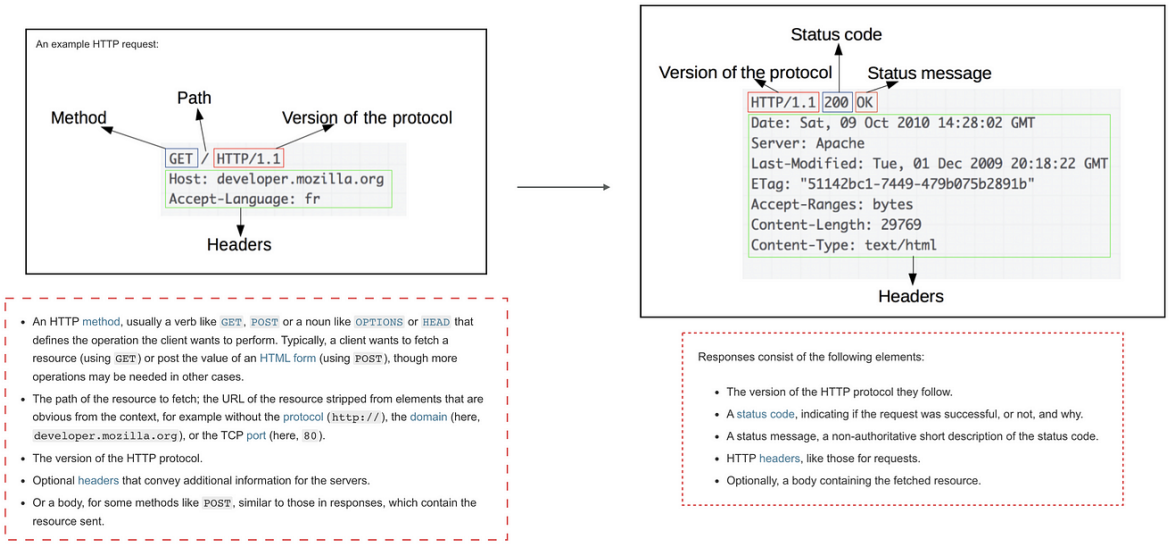
WebSocket Usage Areas

WebSocket makes sense to be used in situations with **real-time** updates and **continuous data flow** over the network

- **Real Time Application (Real Time Application)** : In cases where the screen needs to be updated with this data when continuous data is received on Web Sites such as Stock Exchange, Bitcoin, Live Weather, Election Status.
- **Game Application (Game Application)**: In the game, sometimes even seconds of user interactions can cause you to lose the game, so you and the other players' interactions need to flow to each other continuously.
- **Messaging Applications (Chat Application):** In applications such as Whatsapp, Slack, Messager, etc., we wait for the messages of the person we are instantly messaging to reach the other person.

HTTP Request / Response

It consists of two parts, Request (Request) and its Response (Response).



HTTP Request / Response

Request consists of ***Method, Path, Version Control and Header***. If we examine **Methods**;

HTTP defines a set of **request methods** that specify the desired action on a particular resource. They are also called **HTTP verbs**.

- The **CONNECT** method creates a tunnel to the server defined by the resource at the destination.
- The **OPTIONS** method is used to understand the communication options of the destination resource (in the case of CORS, the other domain asks if that verb exists).
- The **HEAD** method requests a response that is exactly the same as a GET request, only without the body.
- The **TRACE** method performs a message loop-back test along the path to the resource at the destination.
- The **GET** method requests a representation of the specified resource. Requests using GET should only retrieve data.
- The **POST** method is used to send an entity to the specified resource, which usually causes a state change or side effects on the server.
- The **PUT** method replaces all valid representations of the resource at the destination with the request payload ().

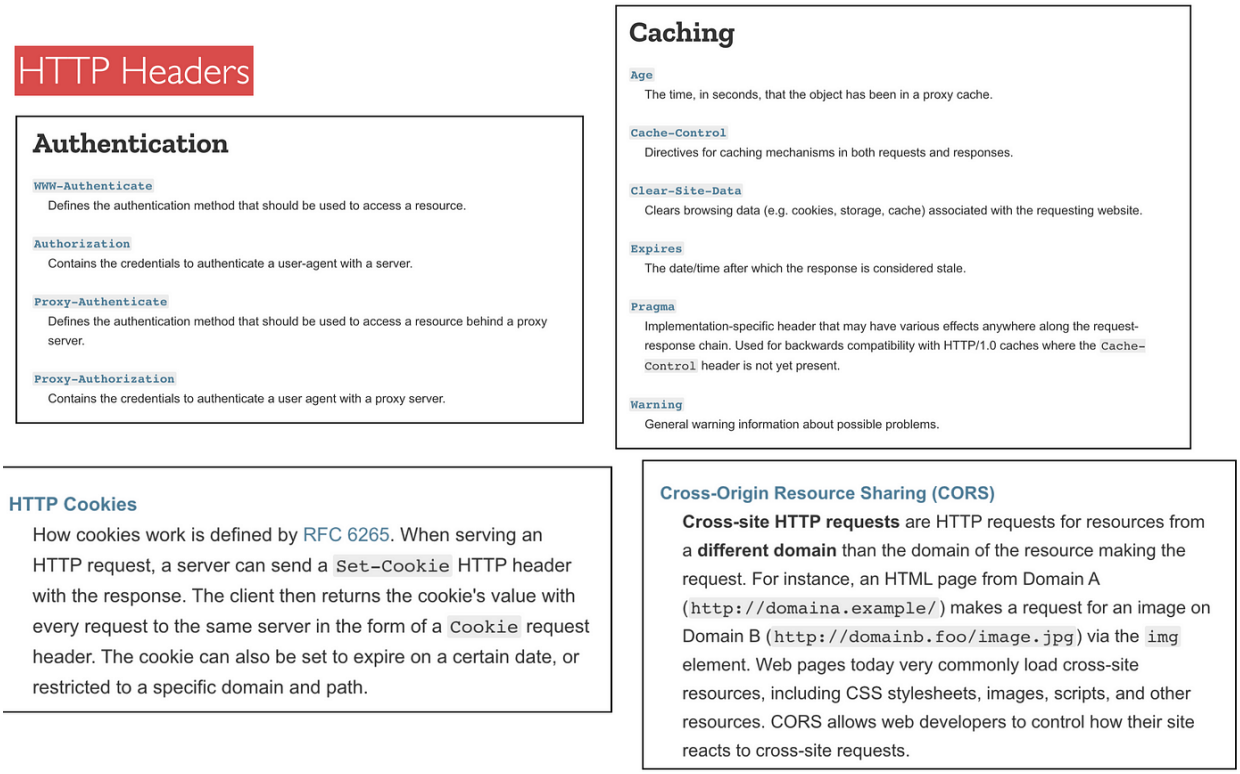
  request payload
- The **DELETE** method deletes the specified resource.
- The **PATCH** method is used to apply partial changes to a resource.

If we look at the Status Code in the Response;

- **Informational responses (100–199):**
- **Successful responses (200–299):** 200 OK. …
- **Redirects (300–399):** 301 — Permanent Redirect. 302 — Temporary Redirect, …,
- **Client errors (400–499):** 404 — Not Found, 410 — Gone, …
- **Server errors (500–599):**500 — Internal Server Error, 503 — Service Unavailable, …

HTTP Header Other Important Data Carried

- Authentication verisi
- Caching
- HTTP Cookies
- CORS (Cross-Origin Resource Sharing)
- HTTP Headers Group Context
- HTTP Headers Group Proxies



Headers..

Mime Type

It is used to specify the type of incoming data. Important MimeType types are ;

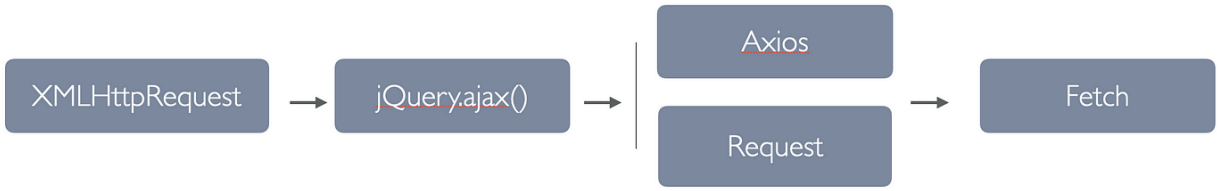
- **application/octet-stream**: unknown binary file
- **text/plain:** unknown textual file
- **text/css :**css textual file
- **text/html:** html textual file
- **text/js:** js textual file

Libraries Used for HTTP

The **XMLHttpRequest** library and similar derivatives of **JQuery.ajax** library and similar derivatives have emerged in order to make it easier to use and handle the state better over time.

Afterwards, **Request** was generally used on the npm and node side.

Increasing the use of promise and then structure, **Axios** and then the **Fetch** API in WebAPI with next generation capabilities was published



HTTP Request (Libraries)

When we compare these HTTPRequest libraries, you can see that the capabilities of the 3 libraries or APIs are very close to each other.

	fetch	axios	request
Intercept request and response	✅	✅	✅
Transform request and response data	✅	✅	✅
Cancel requests	✅	✅	❌
Automatic transforms for JSON data	✅	✅	✅
Client side support for protecting against XSRF	✅	✅	✅
Progress	✅	✅	✅
Streaming	✅	✅	✅

Fetch, Axios ve Request

Fetch API Kullanımı

Get Data

JavaScript Snippet

```
fetch('users.json').then(function(response) {
  return response.json();
}).then(function(json) {
  ChromeSamples.log('Parsed JSON from the response:', json);
}).catch(function(ex) {
  ChromeSamples.log('Parsing failed:', ex);
});
```

Get (https://googlechrome.github.io/samples/fetch-api/fetch-json.html)

Post Data

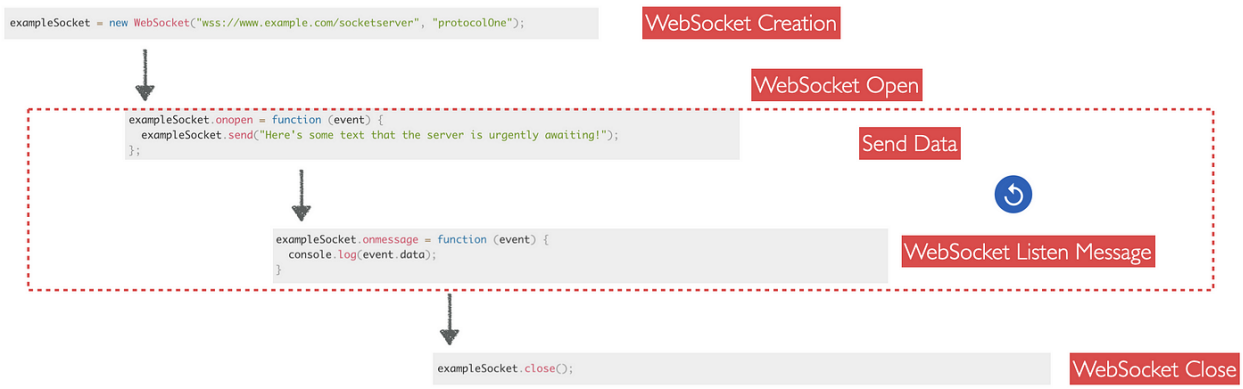
```
function createGist(opts) {
  ChromeSamples.log('Posting request to GitHub API...');
  fetch('https://api.github.com/gists', {
    method: 'post',
    body: JSON.stringify(opts)
  }).then(function(response) {
    return response.json();
  }).then(function(data) {
    ChromeSamples.log('Created Gist:', data.html_url);
  });
}
```

Post (https://googlechrome.github.io/samples/fetch-api/fetch-post.html)

WebSocket

Websocket Operations;

- **WebSocket Creation:** Websocket firstly initiates a connection between the client and the server.
- **WebSocket Open:** When this process is done, that is, when the client and server handshake with each other, we can now send messages to the server or listen to messages from the server
- **Send Data, Listen Msg:** Client and server communication
- **WebSocket Close:** Web socket connection is terminated when work is finished



WebSocket Çalışması

WebSocket Used Libraries

There are many ready-made libraries that we can use with WebSocket. I have listed some of them below;

- HumbleNet: A cross-platform networking library that works in the browser. It consists of a C wrapper around WebSockets and WebRTC that abstracts away cross-browser differences, facilitating the creation of multi-user networking functionality for games and other apps.
- µWebSockets: Highly scalable WebSocket server and client implementation for C++11 and Node.js.
- ClusterWS: Lightweight, fast and powerful framework for building scalable WebSocket applications in Node.js.
- CWS: Fast C++ WebSocket implementation for Node.js (uWebSockets v0.14 fork)
- Socket.IO: A long polling/WebSocket based third party transfer protocol for Node.js.
- SocketCluster: A pub/sub WebSocket framework for Node.js with a focus on scalability.
- WebSocket-Node: A WebSocket server API implementation for Node.js.
- Total.js: Web application framework for Node.js (Example: WebSocket chat)
- Faye: A WebSocket (two-ways connections) and EventSource (one-way connections) for Node.js Server and Client.
- SignalR: SignalR will use WebSockets under the covers when it's available, and gracefully fallback to other techniques and technologies when it isn't, while your application code stays the same.
- Caddy: A web server capable of proxying arbitrary commands (stdin/stdout) as a websocket.
- ws: a popular WebSocket client & server library for Node.js.
- jsonrpc-bidirectional: Asynchronous RPC which, on a single connection, may have functions exported on the server and, and the same time, on the client (client may call server, server may also call client).
- cowboy: Cowboy is a small, fast and modern HTTP server for Erlang/OTP with WebSocket support.

Websocket Kütüphaneler

What is REST?

REST (**Re**presentational **S**tate **T**ransfer), as the name suggests, is to use logical resources in the remote system by calling **GET, POST, PUT, PATCH, DELETE** methods with HTTP protocol.

It was developed by Roy Thomas Fielding as a PhD in 2000. He is also a member of the team that wrote the HTTP specification and HTTP Apache Server.

- Resources (What are resources)?

What the client needs in the Sucucu database;

- User
- Group,
- Invoice,
- Ticket,
- Account etc. objects

are the resources that we will allow the client to query, change and access through the client. In other words, we call **resources (resource)** the elements that clients can access and manage these objects kept in the database, file, etc. on the server

As in the example below, we want to manage access to the Ticket resource. How do we do this with REST?

- GET/tickets (get all tickets)
- GET/tickets/3 (Get ticket number 3)
- POST/tickets (create ticket)
- PUT/tickets/3 (update ticket 3)
- PATCH/tickets/3 (update part of ticket 3)
- DELETE/tickets/3 (delete ticket 3)

Can API calls be made on servers without REST API?

Servers can develop the HTTP protocol in a one-to-one REST structure without designing it as a resource. Mechanisms that do not use REST can be designed with a design to make function calls on the server side. For example **How Servlets work without REST

There are Servlet classes and their names in **web.xml**. You write **Servlets** by passing **POST/GET** from your client over this path with request parameters.

Since it would be costly to write a separate Servlet for each Controller, you make a **DispatcherServlet** yourself

- first parameter **cmd=operationType**
- your second parameter is data=JSON ... and set up a structure.

A single servlet meets the requests, switch control and call the relevant function. After finding the relevant function, you pass the data from the JSON parser and map it to a logical class and fill this class with this incoming data. (Jakson, Gson) are some of the libraries used to convert them into related Java Objects.

If the object you pass

- does not consist of very long parameters,
- session doesn't hold,
- If it can be cached, REST is a suitable method for you. In this way, you do not need to create many servlets or write functions such as **DispatcherServlet**.

Enterprise servers such as **Glassfish, Weblogic, WebSphere, JBoss** provide REST support as **JAX-RS** specification. Web Servers that include only some capabilities of J2EE, such as **Tomcat and Jetty**, do not support **JAX-RS**. You can use libraries such as **Jersey, CFX** from outside. Or the use of **Spring MVC** may be a preference for REST.

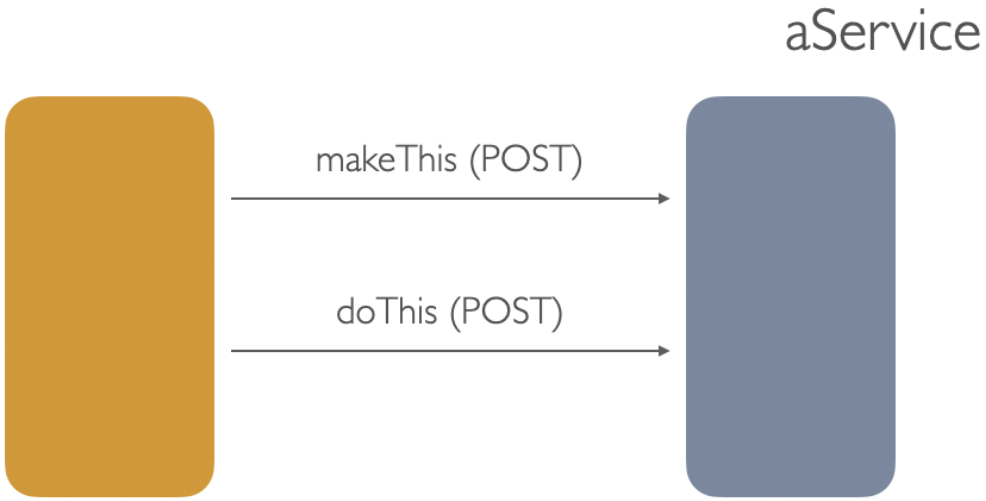
REST Usage Maturity Model (Richardson Maturity Model)

We have given brief information about REST until this section. Richardson has created a maturity model with the levels at which we use REST. I wonder at which level we use this REST.

This model consists of 4 levels. **Level-0 REST** shows that we use it at the lowest level. **Level-3 REST** means that we use it fully and benefit from its flexibility.

Level-0 Swamp of POX

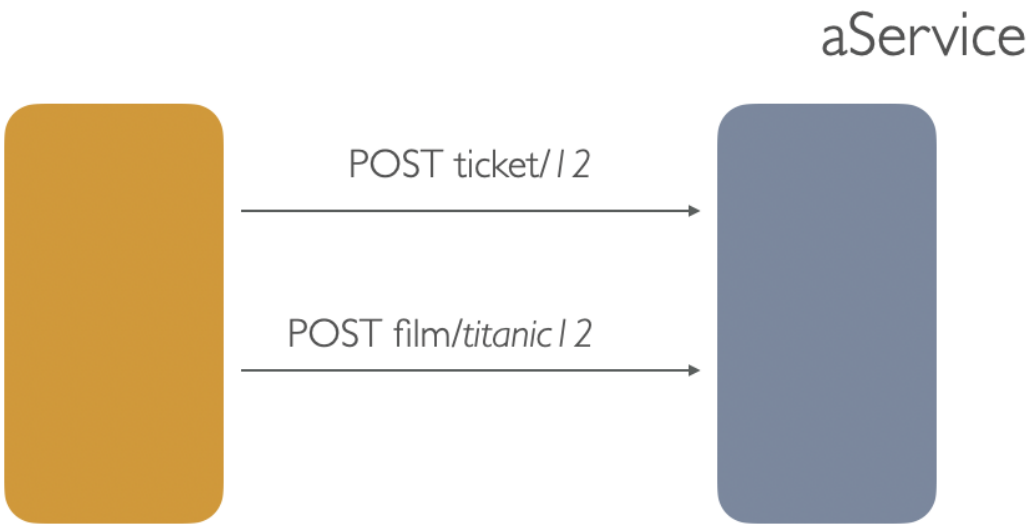
REST is to evaluate and use it as a transport protocol. It is no different from calling a method on a remote system by making a Remote Procedure Call. This usage is similar to SOAP, XML-RPC. In general, it performs calls with the POST method.



Level — 0

Level-1 Resources

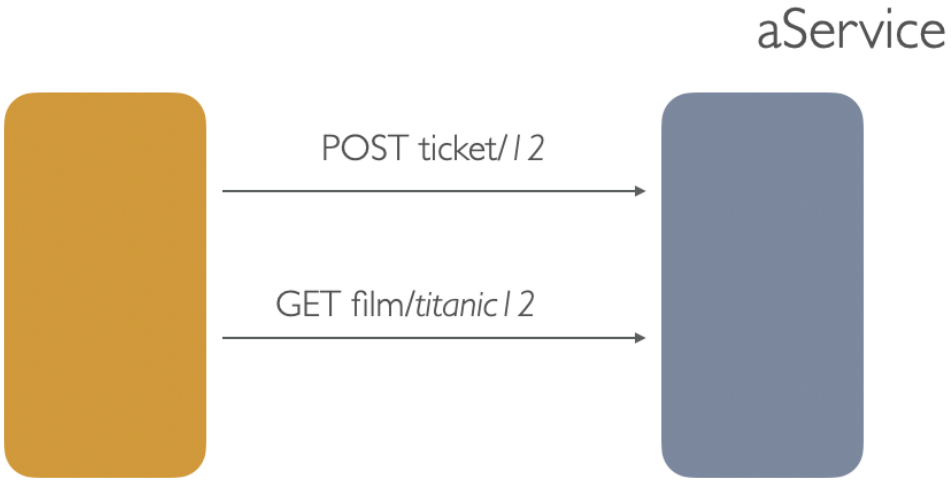
If we are making our REST calling API resource-based from URI, that is, if we assign it under a context, for example, when we say **http://example.org/ticket/12**, I want to access the object number 12 under the ticket resource directly via URI. Generally, calls are made with the POST method.



Level -1

Level-2 HTTP Verbs

At this level, not only POST and URIs are called. In addition, PUT, DELETE, POST and GET methods are called. There is no need to create separate URIs for each of them.

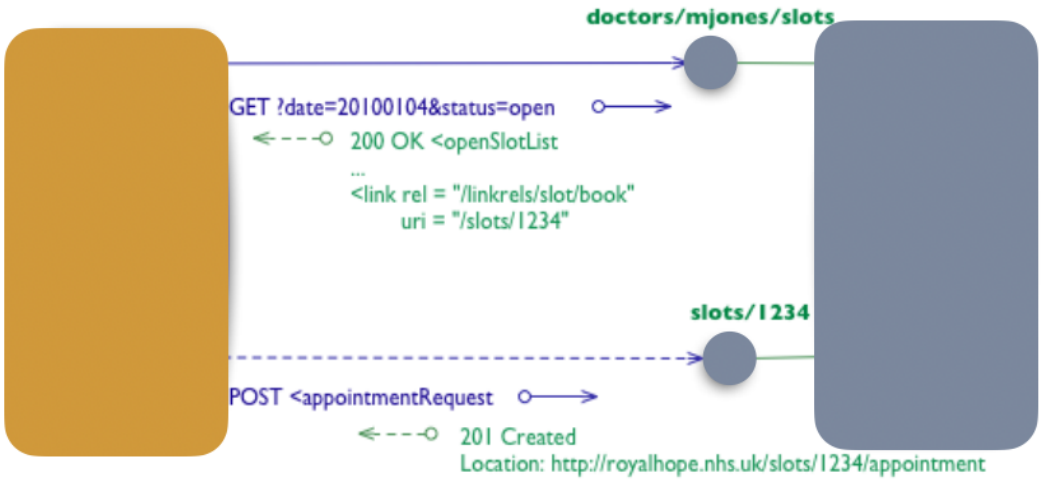


Level -2

Level-3 HyperMedia Controls

In this level of usage, when any URI is called, the operations that can be done with this resource are returned to the client as URI in the response. In this way, the client does not need to keep all the URI and the operations to be performed in the system. In the **HATEOAS** method, the client can continue its operations by making the service discovery through the returned response.

In the example below, Dr Mr Jones is querying the empty appointment status on 2010/01/04. It appears that there are 2 open appointment slots. At the same time, in these 2 appointment slots, reserve these slots link and uri are also returned. It will be quite easy to manage your flows with the server and its URI responses without writing extra code on the processor side.



Level -3

```
GET /doctors/mjones/slots?date=20100104&status=open HTTP/1.1
Host: royalhope.nhs.uk
```

But the response has a new element

```
HTTP/1.1 200 OK
[various headers]

<openSlotList>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450">
     <link rel = "/linkrels/slot/book"
           uri = "/slots/1234"/>
  </slot>
  <slot id = "5678" doctor = "mjones" start = "1600" end = "1650">
     <link rel = "/linkrels/slot/book"
           uri = "/slots/5678"/>
  </slot>
</openSlotList>
```

8. REST vs GraphQL

GraphQL Facebook was developed in 2012 to provide an API infrastructure that would provide a powerful Facebook Data for Facebook application developers, since hundreds of billions of API calls are made every day from mobile applications.

In 2012, multi-user applications such as Facebook and Linkedin were trying to build all their server parts based on HTML5, this logic collapsed at that time. Mark Zuckerberg recognised this mistake and returned to native applications. In the meantime, they realised that the interfaces served as HTML had to map Model-View objects continuously when they defined them as Restful API, and this would cause each client to parse the incoming objects and return its own object.

Developers have transformed to create a structure that returns an answer back in line with their own needs (in its structure). And GraphQL came out.

- Defines a data shape
- Hierarchical (Hierarchical)
- Strongly typed (Type)

- Protocol, not a storage
- Introspective
- Version free

REST => It provided access to resources via URL. (Blog) In addition to the book (blog), it also brings an author resource…

```
GET /books/1{
  "title": "Black Hole Blues",
  "author": {
    "firstName": "Janna",
    "lastName": "Levin"
  }
  // ... more fields here
}
```

GraphQL => We create a schema. In line with this schema, we create the queries we want and get appropriate answers to them.

```
GET /graphql?query={ book(id: "1") { title, author { firstName } } }{
  "title": "Black Hole Blues",
  "author": {
    "firstName": "Janna",
  }
}
```

GraphQL'de Book/Author Schema Samples

```
type Book {
  id: ID
  title: String
  published: Date
  price: String
  author: Author
}type Author {
  id: ID
  firstName: String
  lastName: String
  books: [Book]
}
```

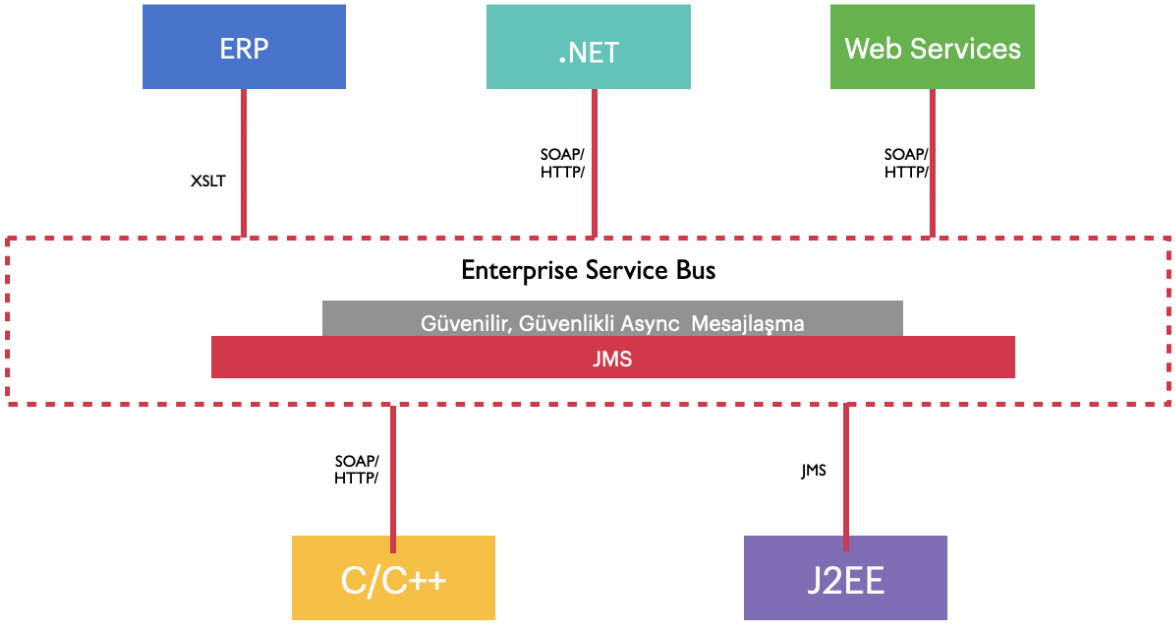
GitHub REST and GraphQL API definitions of a sample SaaS Service

- GitHub REST API https://developer.github.com/v3/
- GitHub GraphQL API https://developer.github.com/v4/

9. SOAP vs REST

Let's look at the concepts of SOAP and REST and their comparison.

When it comes to SOAP, you can think of it architecturally as an XML-based communication method for systems to talk to each other.



ESB üzerinden SOA Mimari

For SO, basically **WSDL** is created for communication. Using this WSDL in Java, many objects such as java client and transport objects, security, etc. are created using Apache AXIS or Apache CXF libraries.

During this communication

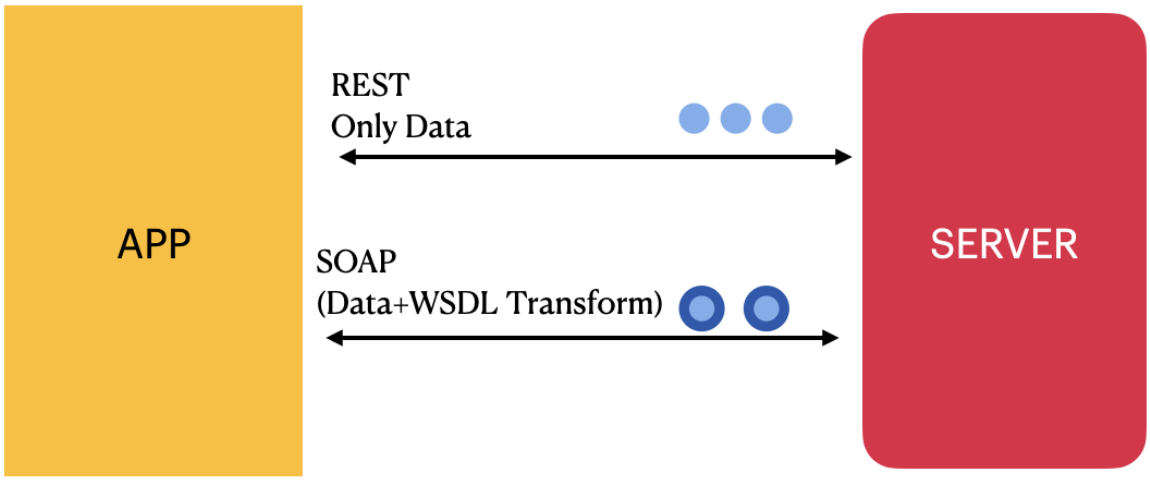
- Compliance with XML schemas,
- security

- It is **Simple Object Access Protocol** which includes many issues such as converting the object transported with XML into POJO objects.

REST (**re**presentational **s**tate **t**ransfer), which means Roy Fielding [thesis] (http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm), aims to manage the client-server relationship between the client and the server through resource or bulk resources with actions such as **GET, POST, DEL, PUT, PATCH**.

On the server side, these objects are sent to the client side as XML/JSON and received in the same formats.

The new method offered by Hypermedia, which REST provides at the last stage, is the ability to access and modify other resources and methods on the data by enabling the new method URL / Addresses to be run from the returned JSON.



REST vs SOAP

10. RPC vs REST

The first APIs created were XML-RPC or SOAP. XML-RPC (remote procedure call) was used. But due to the definition overhead of XML, the client-server relationship has gradually become JSON-based due to the fact that it is a SOAP-based structure.

- While RPC-based APIs create a nice API for actions (for procedures and commands). Slack Web API https://api.slack.com/web

- REST-based APIs are suitable for modelling the domain (resources and assets) and performing CRUD (create, read, update, delete) operations on it. GitHub Rest API https://developer.github.com/v3/**

11. Webhook (User-Defined HTTP Callbacks)

Up to this point we have been talking about normal APIs. **API (Application Programming Interface)** that is, a system uses APIs to share data or functions with you.

For example: You can find a hotel, find a restaurant, or ask this system to send a message, and so on. In summary, you make a request, it tells you that it is successful / unsuccessful in doing this job and does the job and returns the requested data if necessary. (Normal Request/Response)

You can think like Webhook Reverse API. You register an application, if you want this application to trigger you when an event occurs, you should use Webhook for this. In order for different applications to provide integration with each other, the events that occur in the application trigger other applications that subscribe to them with JSON Payloads over HTTP.

We can call this Observer Pattern for applications running on the web. Generally **HTTP Rest API, OAuth2 and JSON** technology is used.

Webhook is your server's effort to inform outsiders by publishing an Event, just like websocket. The difference from websocket,

- websocket **when communicating with the browser** ,
- webhook communicates with **another application server**.

Slack

The first place where Webhook appeared was Slack. In Slack

- Incoming Web Hook
- Outgoing Web Hook allows you to talk to Slack in RealTime.

Also Slack Slash / Commands uses this Webhook infrastructure to talk to external applications.

GitHub

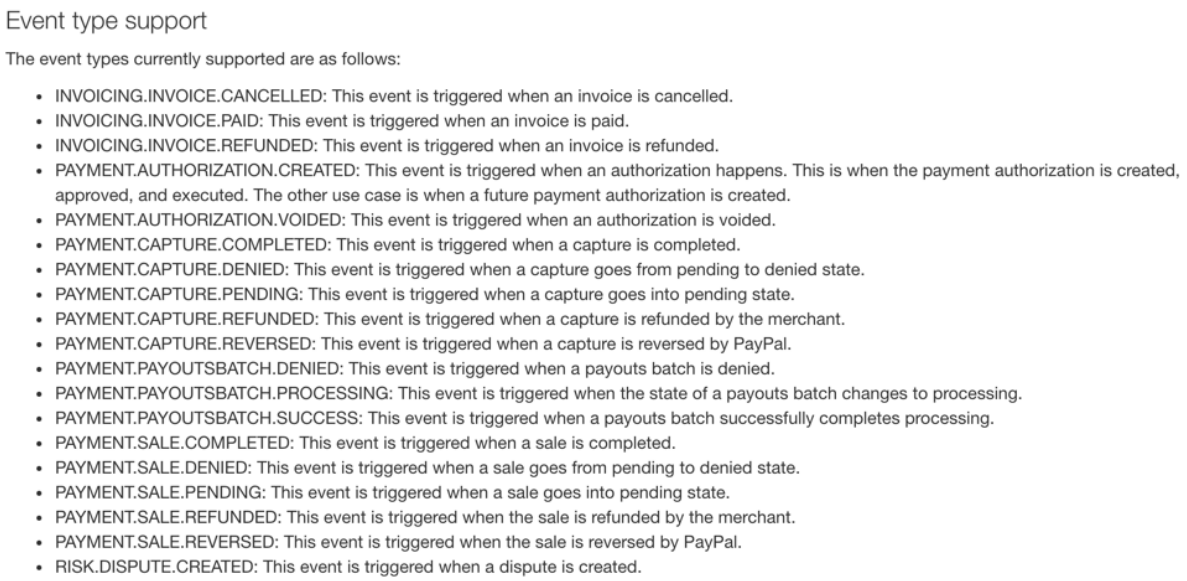
For example, you want to integrate your application on github. Check out the link here. If you **subscribe** to the following events, you can develop your application with appropriate behaviour.

Name	Description
*	Any time any event is triggered (Wildcard Event).
commit_comment	Any time a Commit is commented on.
create	Any time a Branch or Tag is created.
delete	Any time a Branch or Tag is deleted.
deployment	Any time a Repository has a new deployment created from the API.
deployment_status	Any time a deployment for a Repository has a status update from the API.
fork	Any time a Repository is forked.
gollum	Any time a Wiki page is updated.
issue_comment	Any time an Issue or Pull Request is commented on.
issues	Any time an Issue is assigned, unassigned, labeled, unlabeled, opened, closed, or reopened.
member	Any time a User is added as a collaborator to a non-Organization Repository.
membership	Any time a User is added or removed from a team. **Organization hooks only**.
page_build	Any time a Pages site is built or results in a failed build.
public	Any time a Repository changes from private to public.
pull_request_review_comment	Any time a comment is created on a portion of the unified diff of a pull request (the Files Changed tab).
pull_request	Any time a Pull Request is assigned, unassigned, labeled, unlabeled, opened, closed, reopened, or synchronized (updated due to a new push in the branch that the pull request is tracking).
push	Any Git push to a Repository, including editing tags or branches. Commits via API actions that update references are also counted. **This is the default event.**
repository	Any time a Repository is created. **Organization hooks only**.
release	Any time a Release is published in a Repository.
status	Any time a Repository has a status update from the API
team_add	Any time a team is added or modified on a Repository.
watch	Any time a User stars a Repository.

Github Webhook

Paypal, Stripe, Shopify vb..

There are webhooks available for you to develop systems integrated with payment systems Paypal webhook, Stripe webhook, Shopify webhook. The events of these are usually as follows;



Event type support

The event types currently supported are as follows:

- INVOICING.INVOICE.CANCELLED: This event is triggered when an invoice is cancelled.
- INVOICING.INVOICE.PAID: This event is triggered when an invoice is paid.
- INVOICING.INVOICE.REFUNDED: This event is triggered when an invoice is refunded.
- PAYMENT.AUTHORIZATION.CREATED: This event is triggered when an authorization happens. This is when the payment authorization is created, approved, and executed. The other use case is when a future payment authorization is created.
- PAYMENT.AUTHORIZATION.VOIDED: This event is triggered when an authorization is voided.
- PAYMENT.CAPTURE.COMPLETED: This event is triggered when a capture is completed.
- PAYMENT.CAPTURE.DENIED: This event is triggered when a capture goes from pending to denied state.
- PAYMENT.CAPTURE.PENDING: This event is triggered when a capture goes into pending state.
- PAYMENT.CAPTURE.REFUNDED: This event is triggered when a capture is refunded by the merchant.
- PAYMENT.CAPTURE.REVERSED: This event is triggered when a capture is reversed by PayPal.
- PAYMENT.PAYOUTSBATCH.DENIED: This event is triggered when a payouts batch is denied.
- PAYMENT.PAYOUTSBATCH.PROCESSING: This event is triggered when the state of a payouts batch changes to processing.
- PAYMENT.PAYOUTSBATCH.SUCCESS: This event is triggered when a payouts batch successfully completes processing.
- PAYMENT.SALE.COMPLETED: This event is triggered when a sale is completed.
- PAYMENT.SALE.DENIED: This event is triggered when a sale goes from pending to denied state.
- PAYMENT.SALE.PENDING: This event is triggered when a sale goes into pending state.
- PAYMENT.SALE.REFUNDED: This event is triggered when the sale is refunded by the merchant.
- PAYMENT.SALE.REVERSED: This event is triggered when the sale is reversed by PayPal.
- RISK.DISPUTE.CREATED: This event is triggered when a dispute is created.

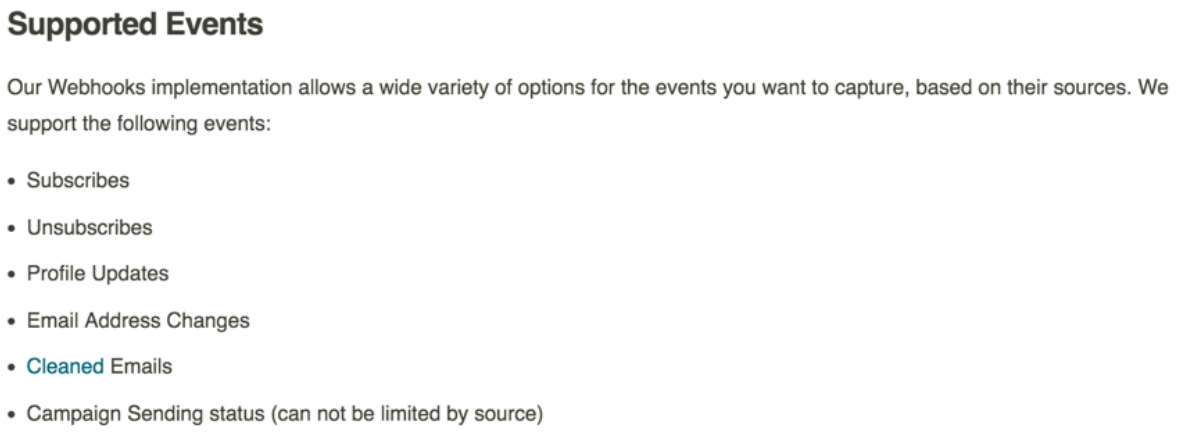
Payment Webhook

Web Pages, Blog vb Static Pages

We usually get updates on such pages via RSS updates. For example, you can see how you can catch events related to these feeds from the Superfeedr Webhook.

Email uygulamaları

You can see that email applications announce the events that take place in them to external applications with Webhook. SendGrid, MailChimp, Mailgun are some of these types of mail applications For example, if we look at MailChimp webhook, you can see the following events.



Email Webhook

12. SMTP ile ePosta İletişimi

Before the Internet data sharing, it first started with e-mail communication to each other, and then standards and terms were determined for e-mail. In this article, I will talk about these standards and protocols. I will explain what you should pay attention to in setting up a sample mail server.

Communication protocols for e-mail are **SMTP, POP3, IMAP, MSA, MTA, MUA, RFC 821, RFC 5821**.

SMTP: It is the protocol for sending and receiving mail to the mail server. It works on port 25. SSL uses port 587 for SMTP.

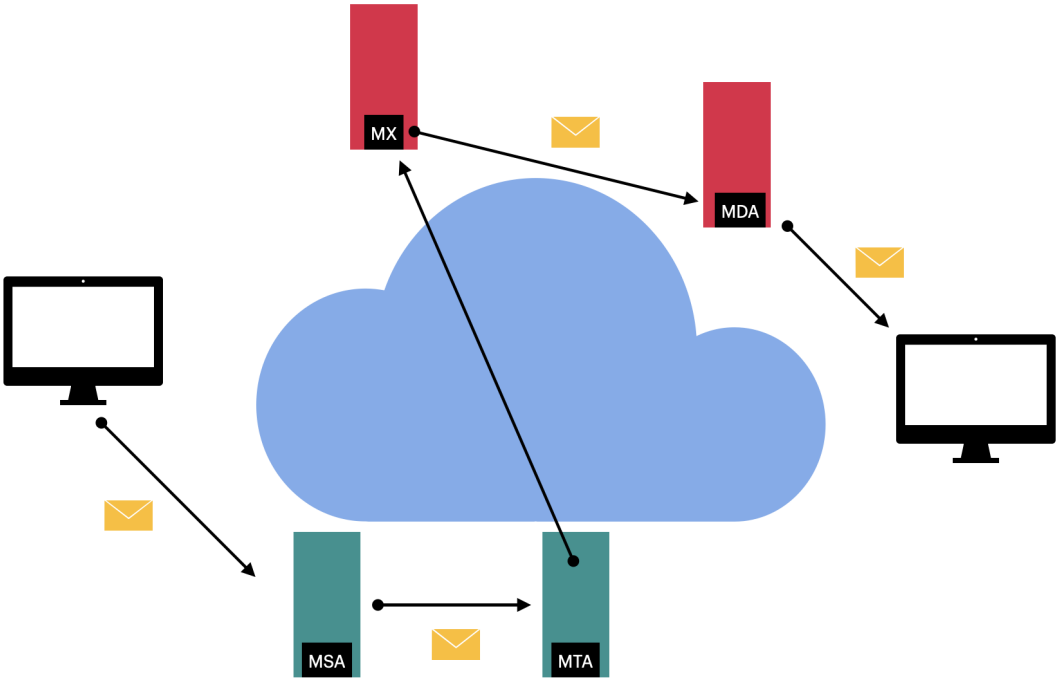
POP3, IMAP: It is a protocol that allows clients to receive mail from the mail server.

MUA, MSA, MTA, MX, MDA: Mail User Agent, Mail Server Agent, Mail Transfer Agent, Mail Exchanger, Mail Delivery Agent

RFC 821, RFC 5821 : It is the standard of mail content travelling between servers.

Mail Flow

MUA: Through the **Mail User Agent** application on the user's computer, you can think of it as looking at your Gmail account from Outlook or Browser. **MSA** When the user sends mail, it accesses the mail server, MTA (Mail Transfer Agent) then MX (**Mail Exchanger**) then MDA (**Mail Delivery Agent**) accesses the email to the other user.


MX
MDA
MSA
MTA