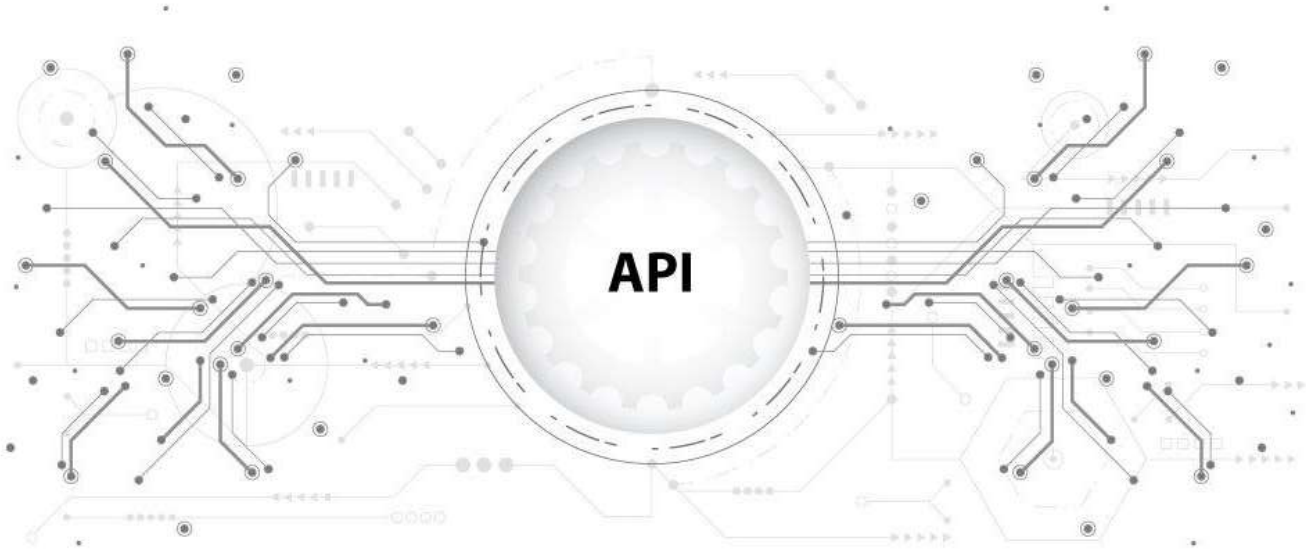
API

(Application Programming Interface)



API Nedir?

API, **uygulama programlama arayüzü** anlamına gelir. Bir API, bir uygulamanın diğer uygulamalar veya sistemlerle nasıl etkileşim kuracağını belirler.

Örneğin, bir web uygulaması, bir veritabanındaki verileri okumak veya değiştirmek için bir API kullanabilir.

Bu sayede, uygulama veritabanına doğrudan erişmek yerine, API aracılığıyla erişebilir ve veritabanının yapısı değiştirilirse bile uygulamanın çalışmasına etki etmez.

API’ler Nasıl Çalışır?

API mimarisi genellikle **istemci** ve **sunucu** bakımından açıklanır. İsteği gönderen uygulamaya istemci, yanıtı gönderen uygulamaya ise sunucu adı verilir.

Yani hava durumu örneğinde, hava durumu veritabanı bir sunucu iken, mobil uygulama ise bir istemcidir.

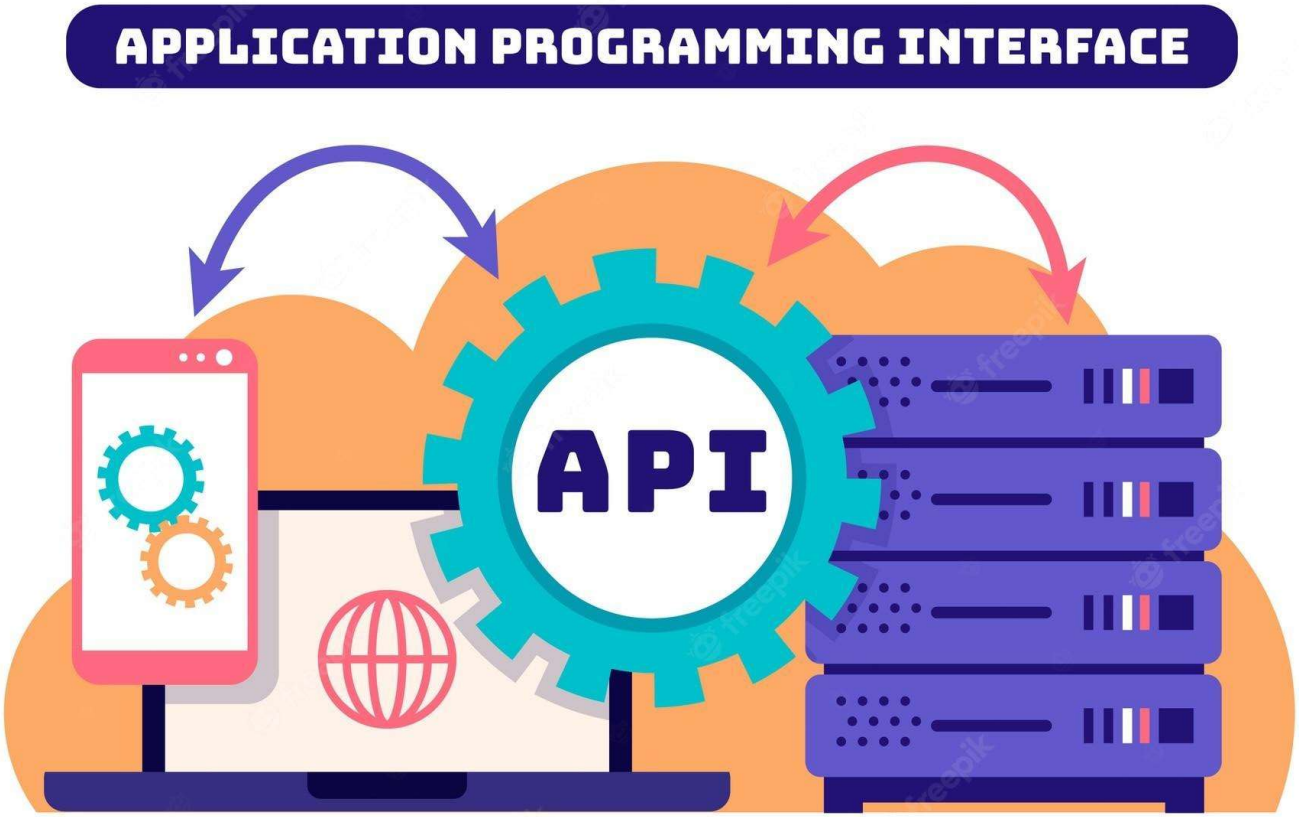
API'lerin çalışma şekli, uygulama ve amaçlara göre değişebilir. Ancak genel olarak, bir istemci tarafından bir istek gönderilir, API tarafından işlenir ve sonuç olarak bir yanıt döndürülür.

Çalışma aşamasını listeleyecek olursak:

1- Bir istemci (örneğin, bir web uygulaması) API'ye bir istek gönderir. Bu istek genellikle **HTTP protokolü** kullanılarak yapılır ve GET, POST, PUT veya DELETE gibi farklı metodlar kullanılabilir.

2- API, isteği **alır** ve **işler**. Örneğin, isteğe bağlı olarak veritabanından verileri alabilir veya başka bir sisteme bağlanabilir.

3- API, istemciye bir **yanıt** olarak işlem sonucunu döndürür. Bu yanıt genellikle JSON veya XML gibi bir veri biçimi kullanılarak döndürülür.


APPLICATION PROGRAMMING INTERFACE
API

API’ler de endpoint(uç nokta) kavramı

Endpoint, bir API üzerinde erişilebilecek bir hizmet veya **veri noktasını** tanımlar.

Endpoint, genellikle bir URL olarak tanımlanır ve API tarafından sunulan hizmetlerin veya verilerin hangi adreslere erişilebileceğini gösterir.

Örneğin, bir veritabanındaki kullanıcıların bilgilerini okumak için kullanılabilecek bir endpoint, "api.örnek.com/kullanıcılar" olabilir. Bir API'da birden **fazla** endpoint olabilir ve her endpoint farklı hizmetler veya veriler sunabilir.

Örneğin: Bir kütüphane için bir API, kullanıcıların kütüphanenin kataloğunda arama yapması, kitap bilgilerini okuması, kitap rezervasyonlarını yapması gibi işlemleri gerçekleştirmek için kullanabilecekleri **birçok endpoint** içerebilir.

api.library.com/search endpointi, kullanıcıların kütüphanenin katalogunda arama yapmasını sağlar.

api.library.com/books/{id} endpointi, bir kitabın bilgilerini okumak için kullanılabilir. {id} yerine kitabın katalog numarası girilir.

api.library.com/books/{id}/reserve endpointi, bir kitabın rezervasyonunu yapmak için kullanılabilir.

api-key (anahtar) kavramı

API anahtarı, bir API'ye erişmek için gerekli olan benzersiz bir tanımlayıcıdır.

API sağlayıcısı, API erişiminin yapılmasını istediği şekilde sınırlandırmak ve izlemek için API anahtarlarını kullanır.

API anahtarı, API isteklerinde HTTP başlıklarına(Headers) eklenir ve API sağlayıcısı tarafından doğrulanır. API anahtarı, bir API'ye yapılan istek sayısını sınırlamak, günlük kullanımı izlemek ve bir API'nin kullanımına izin verilen kullanıcıları ve uygulamaları tanımlamak için de kullanılabilir.

API Türleri Nelerdir?

Open API: Herkes tarafından erişilebilen ve anahtarlı veya anahtarsız kullanılabilen API'lerdir.

Internal API: Sadece belirli bir organizasyon tarafından kullanılan API'lerdir. Ve diğer kullanıcılara kapalıdır.

Partner API: Belirli bir iş ortaklığı için yapılmış API'lerdir.

Composite API: Birden fazla API'nin birleştirilmesi sonucu oluşan API'lerdir.

REST API ve SOAP API Nedir?

API, programların birbirleriyle nasıl iletişim kurduğunu belirleyen belirli bir dizi kural iken, REST ve SOAP API'ler, API'nin nasıl sunulacağını tanımlar.

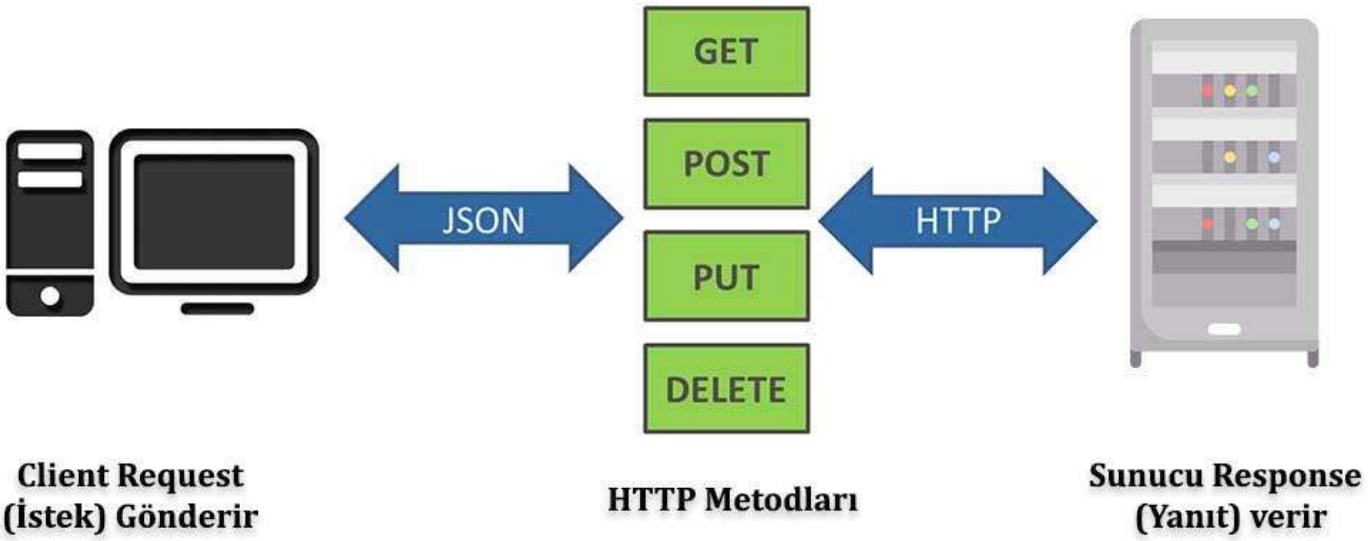
Her biri işlevsellik açısından benzerdir ancak birkaç temel farklılıklarla birbirlerinden ayrılırlar.

REST, client-server arasındaki haberleşmeyi sağlayan HTTP protokolü üzerinden çalışan bir mimaridir. İstemci ve sunucu arasında genelde JSON verilerini taşıyarak uygulamanın haberleşmesini sağlar. REST mimarisini kullanan servislere ise RESTful servis (RESTful API) denir. REST, esnek, büyük ölçekli ve geniş kapsamlı uygulamalar için uygun bir seçenektir.

SOAP, bir nesne erişim protokolüdür ve genellikle XML veri biçimini kullanır.

SOAP, daha önceki dönemlerde web hizmetleri için yaygın olarak kullanılmıştır ancak günümüzde REST daha yaygın olarak kullanılmaktadır.

SOAP, daha yüksek seviyede güvenlik ve hata işleme özelliklerine sahiptir.



Devam etmeden sıkça karşılaşacağınız kavramları inceleyelim.

HTTP Headers

HTTP istek veya yanıt mesajlarının metadatasını(verilerin hakkında bilgi) belirleyen ve gönderen taraf tarafından eklenen ekstra bilgileri içeren alanlardır. HTTP headers, bir HTTP isteğinin veya yanıtının nasıl yorumlanması gerektiği, sunucudan veya istemciden ne gibi ek bilgilerin talep edildiği gibi konuları belirler. Örneğin, Accept-Encoding header'ı, istemcinin hangi sıkıştırma algoritmasını kabul ettiğini belirtir ve Content-Type header'ı, yanıtta hangi medya tipinin bulunduğunu belirtir.

Bazı İstek Başlıkları(Headers)

Accept: İstemci tarafından kabul edilen içerik türlerini belirtir.

Accept-Encoding: İstemci tarafından kabul edilen veri sıkıştırma türlerini belirtir.

Authorization: Sunucuya kimlik doğrulama bilgilerini içerir.

Content-Type: İçerik verisinin türünü belirtir.

Host: İstemci tarafından istenen barındırma sunucusunun adını belirtir.

Bu başlıklar sadece birkaç örnekti. HTTP başlıklarının tam listesi, her bir istek veya yanıt verisi için belirli bir amaçla kullanılabilecek çok sayıda başlık içerebilir.

CRUD

Create, Read, Update, Delete (Oluşturma, Okuma, Güncelleme, Silme) anlamına gelen bir kavramdır ve veritabanı işlemleri için kullanılır.

API (Application Programming Interface) de bu veritabanı işlemlerinin web arayüzü olarak sunulmasını sağlar. Yani, CRUD işlemleri için API aracılığıyla istek yapılabilir ve cevap olarak gerekli veriler sunulabilir. Örneğin, veritabanındaki bir kayıt okuma işlemi için "GET" methodu, kayıt güncelleme işlemi için "PUT" methodu, kayıt oluşturma için "POST" methodu, kayıt silme için "DELETE" methodu kullanılabilir.

Request(İstek)

Bir uygulamanın verilerini veya kaynaklarını başka bir uygulama veya sistem tarafından kullanmasına izin vermek için kullanılan bir yapıdır. API'nin bir parçası olan request, belirli bir kaynağa erişmek için HTTP yöntemlerinden biri olarak (örneğin, GET, POST, PUT, DELETE) yapılan bir istektir.
API, bu istekleri işleyerek gerekli verileri veya kaynakları sağlar. Özet olarak, Request, bir API'ya bir istek yapmak için yapılan bir işlemdir.
İstek, API'ya belirli bir veri veya fonksiyon talep etmeyi amaçlar. Örneğin, bir kullanıcı profilini görüntülemek istediğinde, API'ya bir istek gönderir ve kullanıcı profil verilerini alır. Request, sunucudan veri veya bilgi almak için kullanılan bir araçtır.

HTTP Protocols Nedir?

HTTP metotları, bir web sunucusu ile bir istemci arasındaki veri transferi işlemlerini tanımlar. HTTP metotları, web sunucusuna istekte bulunurken kullanılan metodları tanımlar. Bunlar aşağıdaki metotları içerebilir:

GET: Web sunucusundan belirli bir veriyi almak için kullanılır. Bu metod genellikle web sayfası gibi statik verileri okumak için kullanılır.

POST: Web sunucusuna yeni veri göndermek için kullanılır. Örneğin, bir web formunun verilerini göndermek için kullanılır.

PUT: Web sunucusunda mevcut bir veriyi güncellemek için kullanılır.

DELETE: Web sunucusunda mevcut bir veriyi silmek için kullanılır.

Dikkat edilmesi gerekenler

HTTP metotlarının uygun şekilde kullanılması, veri güvenliği ve performans açısından önemlidir.

GET metodu ile **gönderilen** veriler güvensizdir ve URL'de görünebilir. Bu nedenle, gizli verileri GET metodu ile göndermek uygun değildir.

POST metodu ile gönderilen veriler güvenlidir ve URL'de görünmez, ancak bu metod performans açısından yavaştır.

PUT ve **DELETE** metotları ile yapılan işlemler geri alınamaz ve dikkatli kullanılması gerekir.

HTTP Status Nedir?

HTTP status kodları, sunucunun bir HTTP isteğine verdiği yanıtı tanımlayan kodlardır. Bu kodlar, sunucunun bir isteğe verdiği yanıtın durumunu belirtmek için kullanılır.

1xx (Bilgilendirici): Bu kodlar sunucunun isteği işleme devam ettiğini gösterir. Örnek olarak, 100 Continue kodu sunucunun isteğe devam etmesi gerektiğini belirtir.

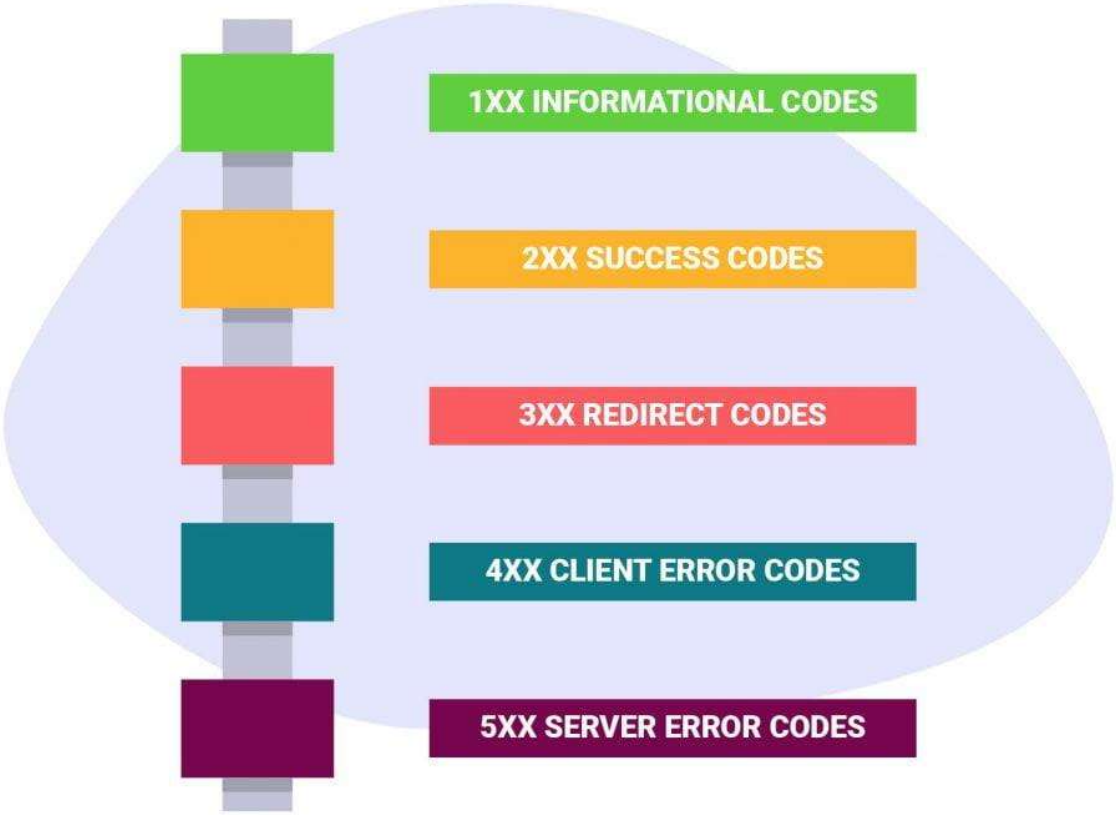
2xx (Başarılı): Bu kodlar sunucunun isteğe başarıyla yanıt verdiğini gösterir. Örnek olarak, 200 OK kodu sunucunun isteğe yanıt verirken herhangi bir sorun olmadığını gösterir.

3xx (Yönlendirme): Bu kodlar sunucunun isteğin işlenmesi için daha fazla adımın gerektiğini gösterir. Örnek olarak, 301 Moved Permanently kodu sunucunun bir kaynağın başka bir URL'ye taşındığını gösterir.

4xx (İstemci Hatası): Bu kodlar isteğin yanlış olduğunu veya eksik bilginin içerdiğini gösterir. Örnek olarak, 400 Bad Request kodu sunucunun isteğin geçersiz olduğunu belirtir.

5xx (Server Hatası): Bu kodlar sunucuda bir hata olduğunu gösterir. Örnek olarak, 500 Internal Server Error kodu sunucunun isteği işleyemeyebileceği bir hata olduğunu belirtir.

HTTP Status Codes



Response Nedir?

Response, HTTP isteği sonucu oluşan yanıttır. HTTP yanıtı, sunucunun isteğe verdiği veriyi veya durum kodunu içerir.

HTTP yanıtı sunucudan isteğe yanıt vermesi anlamına gelir. Yanıt, HTTP durum kodunun yanı sıra veri ve metadatayı da içerebilir.

Status, isteğe yanıt verildiğinde sunucunun isteğe uygun bir şekilde yanıt verip vermediğini gösterir.

Yanıt verileri, JSON, XML veya HTML gibi farklı veri formatlarında sunulabilir ve isteğe göre işlenir.

JSON Nedir?

Json açılımı JavaScript Object Notation'dır ve json nedir dediğimizde verileri yapılandırmak için kullanılan minimal, okunabilir bir formattır. JavaScript; nesne gösterimi, anahtar/değer çiftlerine ve sıralı listelere dayanan yapılandırılmış verilerin şemasız, metin tabanlı bir temsilidir. Günümüzde JSON, web ve mobil istemciler ile sunucu arasında veri alışverişi yapmak için bir standarttır.

JSON, web istemcileri ve web sunucuları arasında bilgi alışverişinde bulunmak için yaygın olarak kullanılır.

Son 15 yılda JSON, web'de her yerde bulunur hale gelmiştir.

JSON Yapısı

Anahtar-değer çiftleri: JSON verileri anahtar-değer çiftlerinden oluşur. Anahtar, veri için bir etikettir ve değer, anahtar ile ilişkili veridir.

Nesneler: JSON nesneleri, anahtar-değer çiftlerinden oluşan kümelerdir. Nesneler, { parantez } işaretleri arasında tanımlanır.

Diziler: JSON dizileri, değerlerden oluşan sıralı listelerdir. Diziler, [parantez] işaretleri arasında tanımlanır.

Tam sayılar: Tamsayı değerleri içerir.

Ondalıklı sayılar: Ondalıklı sayı değerleri içerir.

Boolean: Boolean değerler, true veya false değerlerini içerir.

Null: Null, belirsiz veya tanımsız bir değeri ifade eder.

Aşağıdaki örnekte, "ürünler" isimli bir dizide ürün bilgilerinin JSON yapısı gösterilmektedir:

```
{ "ürünler": [
   {
    "ürün_adı": "Telefon",
    "fiyat": 2500,
    "ürün_türü": "elektronik",
    "stok_durumu": true
   },  // objeler virgül ile birbirinde ayrılır
   {
    "ürün_adı": "Bilgisayar",
    "fiyat": 6000,
    "ürün_türü": "elektronik",
    "stok_durumu": true  }
] }
```

JSON formatındaki ürün bilgilerine javascript'de erişmek dizilerde uyguladığımız yöntemden farksızdır

```
var ürünler = jsonVerisi.ürünler;

for (var i = 0; i < ürünler.length; i++) {

  console.log("Ürün Adı: " + ürünler[i].ürün_adı);
  console.log("Fiyat: " + ürünler[i].fiyat);
  console.log("Ürün Türü: " + ürünler[i].ürün_türü);
  console.log("Stok Durumu: " + ürünler[i].stok_durumu);
}
```

XML Nedir?

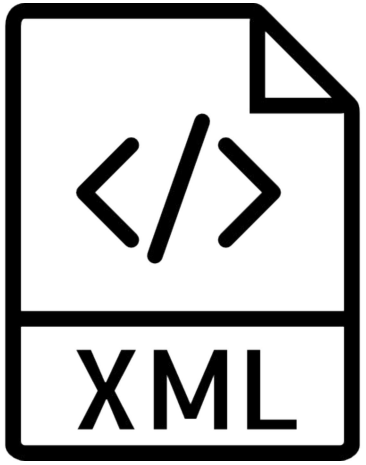
XML (Extensible Markup Language), verilerin saklanması ve taşınması için kullanılan bir metin tabanlı markup dilidir.

XML, HTML'ye benzer şekilde etiketler kullanılarak verilerin işlenmesine olanak tanır, ancak HTML sadece belirli bir veri türünü işleyebilirken, XML verilerin neredeyse her türünü işleyebilir.

Örnek olarak, aşağıdaki XML dosyası, bir kitap katalogu verisi içermektedir:

Bu örnekte, etiketler (örneğin <catalog>, <book>, <title>, vb.) verileri tanımlar ve veriler (örneğin The Great Gatsby, F. Scott Fitzgerald, vb.) etiketler arasında yer alır. Bu şekilde, verilerin ne tür veriler olduğu ve nasıl işleneceği belirtilmiş olur.

```
<catalog>
  <book>
    <title>The Great Gatsby</title>
    <author>F. Scott Fitzgerald</author>
    <publisher>Charles Scribner's Sons</publisher>
    <isbn>9780743273565</isbn>
  </book>
  <book>
    <title>To Kill a Mockingbird</title>
    <author>Harper Lee</author>
    <publisher>J. B. Lippincott & Co.</publisher>
    <isbn>9780061120084</isbn>
  </book>
</catalog>
```



Javascript ‘de API'ye istek atma yolları

XMLHttpRequest (XHR): JavaScript ile geleneksel olarak kullanılan bir yoldur ve klasik bir HTTP isteği göndermek için kullanılır.

Fetch API: Modern bir JavaScript yöntemidir ve XHR'nin bir alternatifi olarak sunulmuştur. Fetch API, asenkron olarak veri çağırmak ve HTTP isteklerini göndermek için kullanılır.

Axios: Axios, JavaScript dünyasında popüler olan bir açık kaynak kütüphanesidir ve XHR veya Fetch API'nin bir alternatifi olarak kullanılabilir. Axios, çoklu platformlarda ve tarayıcıda kolayca çalışmasını sağlar ve REST API'lere istek göndermek için kullanılabilir.

XMLHttpRequest (AJAX): Geleneksel yol

```
let istek = new XMLHttpRequest();
istek.open('GET', 'https://jsonplaceholder.typicode.com/users');
istek.send();    //İsteği gönderir


istek.onload = () => {       //istek yüklenirken koşul oluşturuluyor
  if (istek.status == 200) {
    console.log(JSON.parse(istek.response)); //Geri dönüş json a çevriliyor
  } else {
    console.log(`hata: ${istek.status} ${istek.statusText} `);
  }
};
```

Fetch(): TEMEL Kullanım

FETCH API' yi kullanmak için fetch metoduna istek yapacağımız url' i parametre olarak vermek gerekiyor.

```
fetch('https://jsonplaceholder.typicode.com/users')
```

fetch() metodundan sonra,veri geldikten sonra yapılacak işlemleri belirtmek için metodun sonuna then() promise metodunu ekleriz

```
fetch(url)
  .then((response) => console.log(response))
```

Gelen veriyi(response) console'a yazdık.

Fetch ile Get Methodu Örneği

jsonplaceholder.typicode.com sitesinden kullanıcı verisini alalim.

```
fetch("https://jsonplaceholder.typicode.com/users")
.then(response => response.json())  // gelen veriyi json'a çeviriyoruz
.then((kullanicilar) => {
    kullanicilar.forEach(kullanici => {
    console.log(kullanici.name);  // İsimleri console' a yazdırma
  });
})
```

```
Mrs. Dennis Schulist
Kurtis Weissnat
Nicholas Runolfsdottir V
Glenna Reichert
Clementina DuBuque
>
```

Fetch ile Post Methodu Nasıl Kullanılır?

jsonplaceholder.typicode.com sitesine yeni post ekleyelim.

İlk olarak göndereceğimiz veriyi API tarafından belirtilen doğru veri formatına göre hazırlamamız gerekiyor.

```
const data = {
    userId: 25,
    id: 5,
    title: 'Selam Dostum',
    body: 'Sana Dünyadan selam getirdim!'
};
```

API'nin yönergelerine göre gerekli headers'ı ve methodu belirtip önceden tanımladığımız data'yı gönderdik:

```
fetch('https://jsonplaceholder.typicode.com/posts', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json; charset=UTF-8',
  },
  body: JSON.stringify(data),
})
```

Get methodunda olduğu gibi işlemin durum bilgisini görmek için .then ile gelen response'u ve error'u ele alabilirsiniz.

Fetch ile Put Methodu Nasıl Kullanılır?

jsonplaceholder.typicode.com sitesinde mevcut olan bir postu güncelleyelim

```
fetch('https://jsonplaceholder.typicode.com/posts/1', {
  method: 'PUT',
  body: JSON.stringify({ id: 1,title: 'foo',body: 'bar',userId: 1, }),
  headers: {
    'Content-type': 'application/json; charset=UTF-8',
  },
})
```

Fetch ile Delete Methodu Nasıl Kullanılır?

API' de mevcut olan postu silme işlemi aşşağıdaki gibidir.

```
fetch('https://jsonplaceholder.typicode.com/posts/1', {
  method: 'DELETE',
})
  .then((res) => console.log(res));
```

Dikkat edilmesi gerekenler:

Doğru URI: Silinmesi istenen kaynağın URI'si doğru şekilde belirtilmelidir

Geri Alınamaz Olma: DELETE methodu çalıştıktan sonra kaynak geri alınamaz, dikkatli kullanılmalıdır

Axios(): TEMEL Kullanım

Axiosu kullanmak için önce axios kütüphanesi projeye dahil etmemiz gerekiyor.

Node.js için terminale aşşağıdaki komutu girerek projeye dahil edebilirsiniz

```
npm install  axios
```

Bir react projesinde kullanımı için izlemeniz gereken diğer adımlar:

1- İstek yapmak istediğiniz component içinde Axios'u import etmek

```
import axios from 'axios';
```

2- Axios'u kullanarak HTTP isteği yapmak.

axios'da HTTP çağrısı yapabilmek için HTTP metod tiplerine özel yazılmış fonksiyonları kullanmaktır.

```
axios.get(url) & axios.post(url, data) &  axios.delete(url) & axios.put(url, data)
```

Örneğin aşşağıdaki şekilde kullanılabilir

```
axios.get("/users");
 veya
axios.post("/login", {
    username: "kullanici adi",
    password: "parola"
});
```

3- İstek yapıldıktan sonra gelen cevabın (response) işlenmesi:

Geri dönen cevabı fetch methodundaki gibi .then promise methodu ile ele almak gerekiyor. Ancak axios veriyi otomatik olarak JavaScript objesine dönüştürür. Yani parse işlemi için, ikinci .then methoduna ihtiyacımız yok.

```
axios
  .get('https://jsonplaceholder.typicode.com/users')
  .then((response) =>console.log(response) )
  .catch((err) => console.log(err));
```

4- Axios ile istek yaparken API'nin bizden istediği header verilerini göndermek isteyebilirsiniz.

O zaman tanımladığımız header'ları axiosun çağırdığımız methoduna parametre olarak giriyoruz.

```
const config = { // config objesinde header tanımlanıyor
  headers: {
    'Content-Type': 'application/json',
  },
};
axios.post(url, data, config);  // config(headers) parametre olarak verildi
```

5- Axios, istekleri yaparken çeşitli global ayarlar içerebilir.

Bunlar, her bir isteğe uygulanmasını istediğiniz farklı seçeneklerdir.

Örneğin, her bir isteğe gönderilmesi gereken ortak bir "Authorization" başlığının olması gerektiği durumlarda, bu başlığı tek bir yerde tanımlayabilirsiniz ve her bir istekte otomatik olarak uygulanmasını sağlayabilirsiniz.

Bunun yanı sıra, Axios, "base URL" olarak adlandırılan ve her bir isteğin başına eklenmesi gereken bir URL tanımlayabilirsiniz.

```
axios.defaults.headers.post['Content-Type'] = 'application/json';
```

```
baseURL: 'https://api.example.com/',
```

Örnek - Aşağıdaki örnekte, Axios'un global olarak base URL ve headers (token gibi) ayarlarını değiştirmek için axios.create() fonksiyonu kullanılır:

```
import axios from 'axios';


const instance = axios.create({
  baseURL: 'https://jsonplaceholder.typicode.com/',
  headers: {
    'Authorization': 'Bearer ' + localStorage.getItem('token')
  }
});
export default instance;
```

Örnek - Üstte global ayarlarını tanımladığımız axios hookunu kullanalım.

```
import api from './instance';  //yukarıda tanımladığımız methodu api ismiyle çağırdık


function App() {


  useEffect(() => { // sayfa yüklendiğinde useEffect içindeki kodlar çalışır
    api
      .get('users')  // base url belirttiğimiz için urlnin başındaki kısmı yazmadık
      .then((response) => console.log(response))
      .catch((err) => console.log(err));
  }, []);


}
```

Çektiğimiz verileri listelediğimiz bir **örnek** yapalım:
1- İlk olarak, "axios" kütüphanesi projede kullanılmak üzere "import" edilir.

2- Sonra, React'in "useEffect" ve "useState" hook'ları kullanılır. "useState" hook'u, verilerin tutulması için boş bir dizi olarak ilk değer verilir ve "setKullanicilar" fonksiyonu ile boş dizi değiştirilir.

3- "useEffect" hook'u, component yüklendiğinde verilerin çekilmesini sağlar.

4- "axios.get" ile veriler "https://jsonplaceholder.typicode.com/users" adresinden çekilir. "then" içinde, çekilen veriler "setData" ile "data" dizisine atanır. Eğer hata oluşursa, "catch" bloğu ile loglanır.

5- Son olarak, veriler "map" fonksiyonu ile döngü halinde ekrana yazdırılır. Her bir kullanıcı için ad, telefon ve e-posta bilgisi yazdırılır.

```jsx
import axios from 'axios';
import { useEffect, useState } from 'react';

function App() {

 const [data, setData] = useState([]);

 useEffect(() => {
   axios
     .get('https://jsonplaceholder.typicode.com/users')
     .then((response) => setData(response.data))
     .catch((err) => console.log(err));
 }, []);

// return kısmı yan tarafta belirtilmiştir
```

```jsx
return (
    <>
      {data.map((user) => (
        <div>
          <p>{user.name}</p>
          <p>{user.phone}</p>
          <p>{user.email}</p>
        </div>
      ))}
    </>
  );
```

Örnek - Async await ile kullanım:

Async/await, JavaScript içinde asenkron fonksiyonların yönetimini kolaylaştırmayı amaçlar. Asenkron fonksiyonlar, belirli bir noktadan sonra devam etmelerini bekleyen diğer işlemlerin etkisini minimumda tutar.
Ayrıca, asenkron fonksiyonların çalışması sonucunda dönen değerleri beklemek için Promise objeleri kullanmak zorunda kalmayız.
Bunun yerine, async/await sözdizimi ile fonksiyonlarımızın tamamlanmasını beklemek ve dönen değerleri kullanmak kolay hale gelir.

```
async function getUsers() {
  try {
    const response = await axios.get(
      'https://jsonplaceholder.typicode.com/users'
    );
    const users = response.data;
    console.log(users);
  } catch (error) {
    console.error(error);
  }
}
```

Axios'un **Avantajları**:

1- Sunucuya post, put, patch ve delete istekleri göndermek kolaydır.

2- İstek başlıkları(headers) ve sunucu tarafından dönen durum kodlarını ele almak kolaydır.

3- Çoklu İstekleri Destekleme: Axios, aynı anda birden fazla HTTP isteği yapmanıza olanak tanır. Bu, verileri birden fazla sunucudan çekmeniz gerektiğinde yararlıdır.

4- Sunucudan Gelen Verileri Otomatik Olarak Dönüştürme: Axios, sunucudan gelen verileri otomatik olarak JavaScript nesnelerine dönüştürür, bu nedenle verileri kolayca kullanabilirsiniz.

Axios ve Fetch Arasındaki Temel **Farklar**:

1- Axios, istemci tarafında HTTP istekleri yapmak için kullanılan bir JavaScript kütüphanesidir, ancak fetch, JavaScript'te yerleşik bir API'dir.

2- Axios, daha fazla özelleştirme seçeneği sunar ve istek ve cevapların yapılandırılmasına izin verir, ancak fetch daha basit ve temel bir API'dir.

3- Axios, sunucuya yanıt vermeyen veya hata oluşan istekleri ele almak için daha iyi bir destek sunar, ancak fetch bu durumlarda manuel olarak ele alınmalıdır.

4- Axios, sıkıştırılmış verileri otomatik olarak açar ve JSON verilerini dönüştürür, ancak fetch verileri manuel olarak açılması gerekir ve JSON verileri dönüştürülmelidir.

Bir **frontend** developer API hakkında bilmesi gereken şeyler şunlardır:

1- API Endpoint'leri: API endpoint'leri, verilerin alınabileceği veya gönderilebileceği URL'lerdir. Frontend developer API endpoint'lerinin ne zaman ve nasıl kullanılacağını anlamalıdır.

2- Veri Formatları: API verilerini genellikle JSON veya XML formatında sunar. Frontend developer bu veri formatlarını anlamalı ve kullanabilmelidir.

3- İstek Tipleri: API'lere GET, POST, PUT, DELETE gibi farklı istek tipleri ile erişilebilir. Frontend developer hangi istek tipinin hangi durumlarda kullanılması gerektiğini anlamalıdır.

4- HTTP Durum Kodları: API tarafından gönderilen her cevap bir HTTP durum kodu içerecektir. Bu durum kodları, API tarafından verilen yanıtın başarılı veya başarısız olduğunu gösterir. Frontend developer bu durum kodlarını anlamalı ve kullanmalıdır.

5- Güvenlik ve Yetkilendirme: Bazı API'ler kimlik doğrulama ve yetkilendirme gerektirir. Örneğin: Kullanıcı login olunca Kimlik doğrulama işleminin doğru sonuç vermesi durumunda, yetkilendirme gerekli olabilir.

6- Hata Yönetimi: API tarafından verilen cevaplarda hata oluşabilir. Frontend developer hata yönetimi için nasıl düzgün bir yol izleneceğini ve hata mesajlarını nasıl yorumlayacağını anlamalıdır.