

# DOM Kavramı

DOM, HTML elemanlarının ve özelliklerinin bir **ağaç yapısı** olarak temsil edildiği **hiyerarşik** bir modeldir.

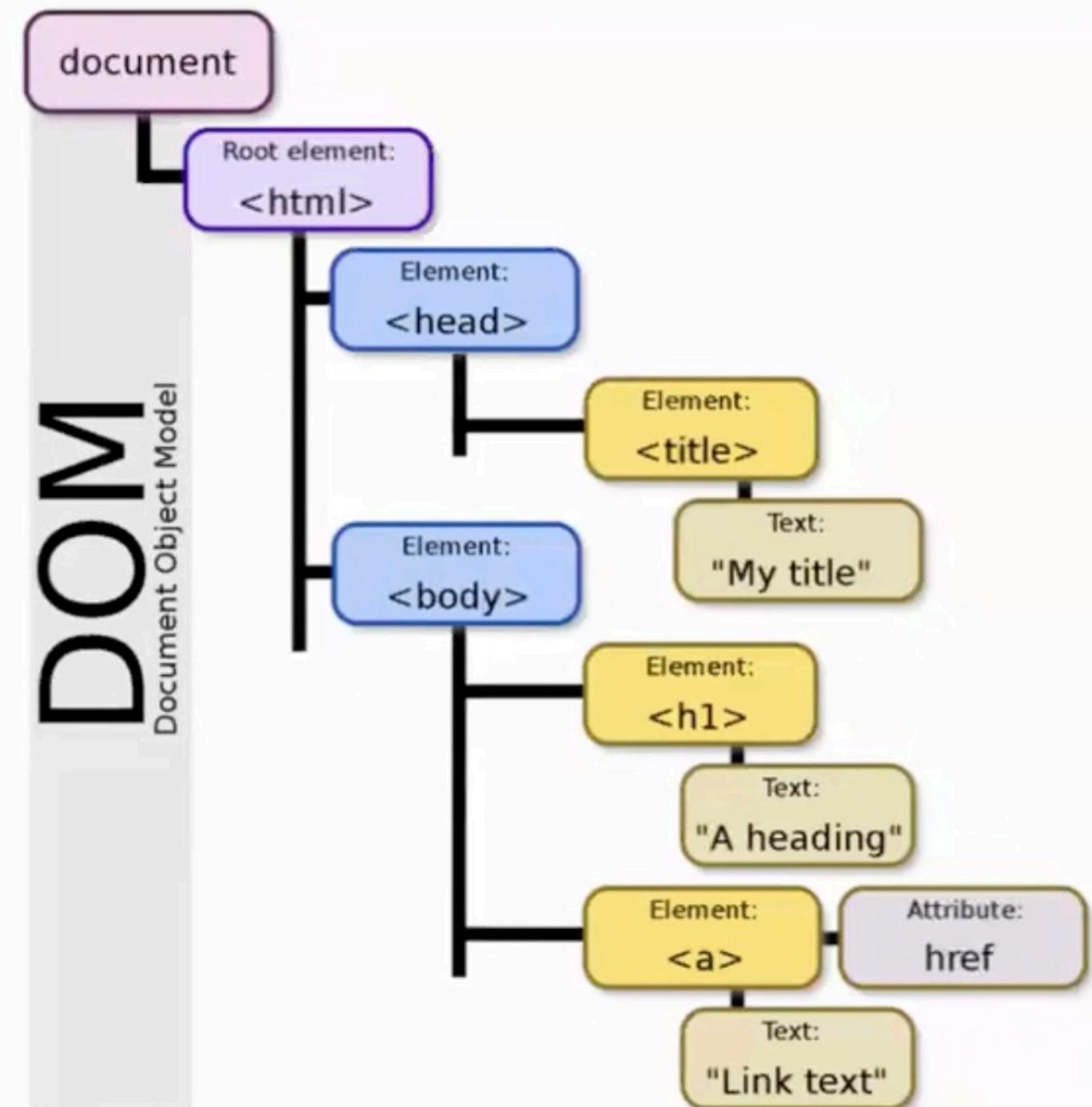
- HTML: **Statiktir**, veri göstermek için bir sunum dilidir.
- **Document Object Model**: Browser'ın HTML dosyasını okuyarak oluşturduğu bir **JavaScript objesidir**.
- **JavaScript** ile bu obje üzerinde değişiklikler yaparak sayfayı **dinamik** yapabiliriz.

**HTML: Ev Planı**

**DOM: Ev**

# DOM Kavramı

- DOM ağacının kökünde **document** objesi bulunur.
- Her bir element objedir. Bu objelerin DOM ağacındaki yerlerine **node**, yani düğüm denir.
- Tagler, HTML dosyasında bulundukları konuma göre, yani sahip oldukları parent-child ilişkilerine göre DOM ağacına yerleştirilir.



# DOM elemanlarını seçmek

GetElement Metodları	Örnek Kullanım
getElementById id attribute'una göre göre seçmek için kullanılır	document.getElementById("darkMode")  1 tane
getElementsByName name attribute'una göre elementleri seçmek için kullanılır	document.getElementsByName("formItem") n tane
getElementsByTagName tag ismine göre seçmek için kullanılır	document.getElementsByTagName("body") n tane
getElementsByClassName class ismine göre seçmek için kullanılır	document.getElementsByClassName("btn") n tane

QuerySelector Metodları	Örnek Kullanım
querySelector parantez içindeki selector ile eşleşen ilk elemanı seçer	document.querySelector("section.features") 1 tane
querySelectorAll parantez içindeki selector ile eşleşen her elemanı seçer	document.querySelectorAll(".photos img") n tane

# DOM elemanlarını seçip değişkende saklayabiliriz

```
> document.getElementById("acKapa");  
<button id="acKapa">Karanlık temayı aç</button>
```

```
const button = document.getElementById("acKapa");
```

# DOM Manipülasyonları: **textContent**

Bir element'in içindeki yazıyı **textContent** ile değiştirebiliriz. (örn: button metni, sayfanın title'ı)

```
const button = document.getElementById("acKapa");
button.textContent = "Karanlık temayı aç";

// ya da tek satırda
document.getElementById("acKapa").textContent = "Karanlık temayı aç";
```

# DOM Manipülasyonları: **classList**

Bir element'e **classList** metodları ile;

- class ekleme, çıkarma yapabiliriz
- veya o class'ı içerir mi kontrolü yapabiliriz.

```
eleman.classList;           // Elemanın sahip olduğu classları listeler.  
eleman.classList.add("dark"); // Elemana dark classını ekler. Varsa tekrar eklemez.  
eleman.classList.remove("dark"); // Elemandan dark classını varsa çıkarır. Yoksa değişiklik yapmaz.  
eleman.classList.toggle("dark"); // Elemanda dark classı yoksa ekler, varsa çıkarır.  
eleman.classList.contains("dark"); // Elemanda dark classı varsa true, yoksa false döner.
```

# DOM Manipülasyonları: **style** property

**style** property ile bir element'in rengini, arkaplan rengini, font büyüklüğünü, kenar çizgi kalınlığını v.b. bir çok stil özelliğini değiştirebiliriz.

```
document.querySelector(".name").style.color = "red";
```

- .backgroundColor = “black”
- .fontSize = “32px”
- .color = “#ab45tr”
- .margin = “0 auto”
- .padding = “1rem 2rem”
- .borderRadius = “20px”

## Ekstra

1. Birden çok DOM elementini seçmek
2. DevTools'da properties tab'ı ile çalışmak
3. Sayfaya JS dosyası eklemek

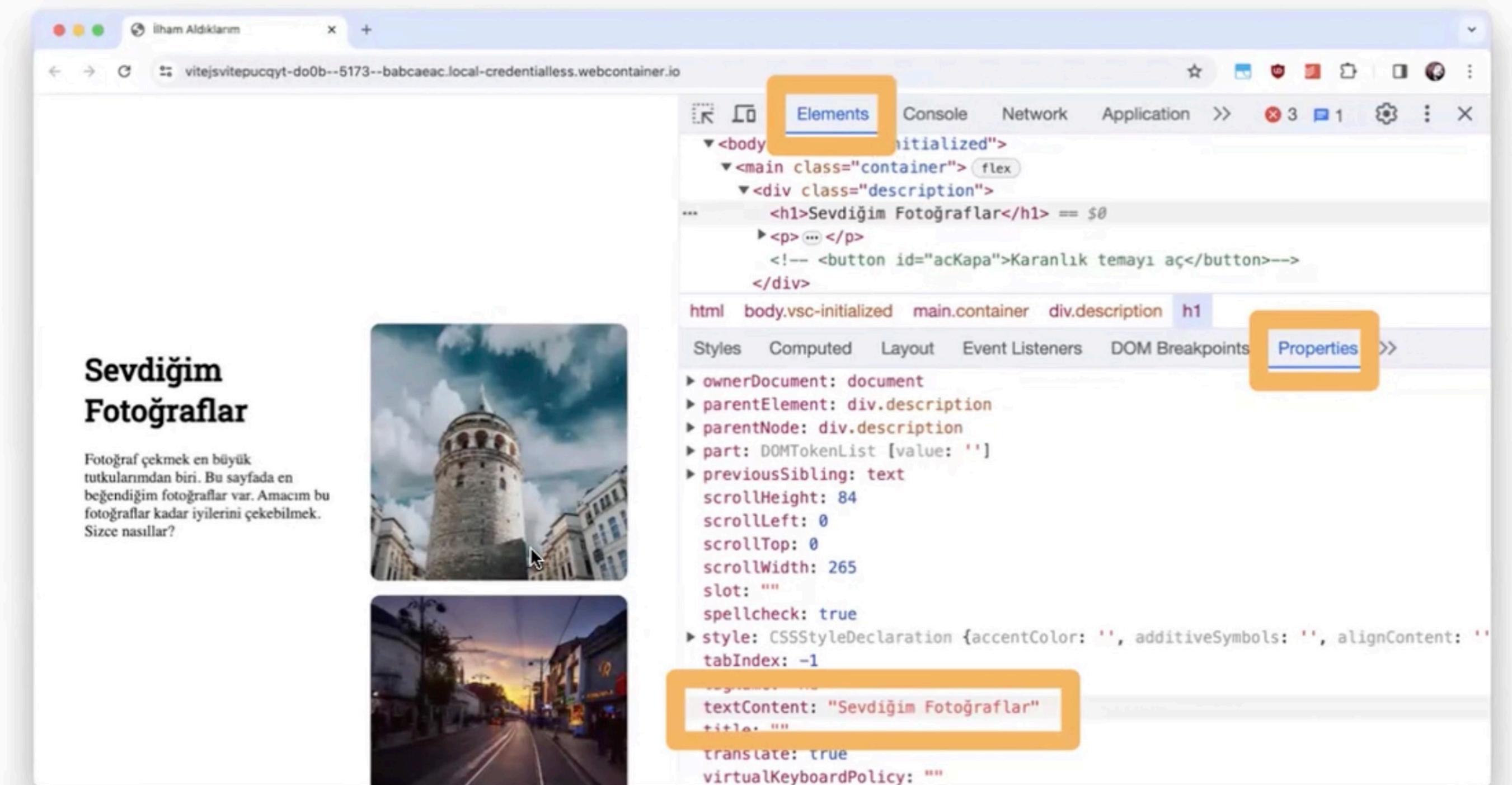
## Ekstra: Birden çok DOM elemanı seçmek

```
> document.querySelectorAll("img");
< ◀ ► NodeList(5) [img, img, img, img, img]
```

```
// tüm resimleri bir değişkende saklamak
const images = document.querySelectorAll("img");

// tüm resimleri döngüye sokup border eklemek
for (let i = 0; i < images.length; i++) {
  images[i].style.border = "2px solid white";
}
```

# Ekstra: DevTools > Elements > Properties tab'ı



## **Ekstra: Sayfaya js dosyası eklemek**

JavaScript kodlarınımız 2 farklı yere yazabiliriz:

- HTML dosyası içinde, `<script></script>` tagleri arasına,
- .js uzantılı bir dosyaya
  - HTML dosyasına `<script src="dosyaYolu/dosyaAdi.js"></script>` tagiyle bağlayarak.

### **Önemli Not:**

- Sayfamızın hızlı yüklenmesi için JS kodları body tagi kapanmadan hemen önce yazılır.

# DOM Event Kavramı

- **Events (Kullanıcıların sayfada yaptığı eylemler):**

Tıklamak, kaydırma, sağ tıklamak, mouse hareketleri, klavye tuşlarına basmak, vb.

- **EventListener'lar** bir **DOM eventini dinleyen** ve o event gerçekleştiğinde bir fonksiyonu tetikleyen metodlardır.

- **EventHandlers (Bu eylemler gerçekleştiğinde yapılması istenenler)**

*“...toggleTheme fonksiyonu çalışın”*

# DOM Event Kavramı [Event List -w3schools](#)

En sık kullanacağımız 5 tanesi:

- **click**: fare ile sol tıklama eventidir.
- **mousemove**: farenin hareket eventidir.
- **keyup**: klavyede bir tuşa basma eventidir.
- **change**: bir form elemanın değişmesini temsil eder.
- **submit**: bir formun gönderilmesi eventidir.

# addEventListener

eventListener kullanmak için 3 şey lazım:

- 1. Eleman,**
- 2. Event türü**
- 3. Çalışacak fonksiyon.**

The diagram shows a code snippet: `const button = document.getElementById("acKapa"); button.addEventListener("click", toggleTheme);`. Four red arrows point from labels below the code to specific parts of the code:

- An arrow points from the label "element" to the variable `button`.
- An arrow points from the label "event type" to the string `"click"`.
- An arrow points from the label "function" to the function name `toggleTheme`.
- An arrow points from the label "eventListener" to the method name `addEventListener`.

```
const button = document.getElementById("acKapa");
button.addEventListener("click", toggleTheme);
```

element    event type    function    eventListener

# Eleman Oluşturmak, Eklemek, Silmek

`document.createElement(tag)` ile yeni bir eleman oluşturabiliriz.

```
// yeni bir p tagi oluşturduk:  
const yeniParagraf = document.createElement("p");
```

`eleman.appendChild(elaman2)` ile bir elemana child olarak başka bir elemanı ekleyebiliriz.

```
// elemanı oluşturduk ama sayfaya ekleyene kadar görünmez:  
document.querySelector(".description").appendChild(yeniParagraf);
```

`eleman.remove()` ile bir elemanı DOM'dan çıkartabiliriz.

```
yeniParagraf.remove();
```

```
> document.addEventListener("click", (event)
  console.log(event);
})
<- undefined
```

```
▼ PointerEvent {isTrusted: true, pointerId: 1, ...
  1, pressure: 0, ...} i
  isTrusted: true
  altKey: false
  altitudeAngle: 1.5707963267948966
  azimuthAngle: 0
  bubbles: true
  button: 0
  buttons: 0
  cancelBubble: false
  cancelable: true
  clientX: 542
  clientY: 123
  composed: true
  ctrlKey: false
  currentTarget: null
  defaultPrevented: false
  detail: 1
  eventPhase: 0
  fromElement: null
  height: 1
  isPrimary: false
  layerX: 542
  layerY: 123
  metaKey: false
  movementX: 0
  movementY: 0
  offsetX: 542
  offsetY: 123
  pageX: 542
  pageY: 123
```

# Event objesi

**addEventListener** metodu, **bir eventin detaylarını içeren objeyi kendine verilen fonksiyona ileter.**

Bu sayede event detaylarına bağlı işlemler yapmak mümkün olur.

**Örnek:** bir fare tıklama olayı için, event nesnesi tıklamanın tam olarak nerede gerçekleştiğini (x ve y koordinatları), hangi düğmenin tıklanmış olduğunu (sol, sağ veya orta) ve tıklama sırasında hangi tuşların basılı olduğunu (Shift, Ctrl vb.) içerir.

# HTML Form Elemanları

**<form>**: kullanıcıdan veri almak için kullanılan elemanları grüplamak için kullanılır, kendine özgü bir görüntüsü yoktur.

**<label>**: form elemanlarına etiket eklemek için kullanılır. Bu etiketler, kullanıcıya hangi bilgiyi girmesi gerektiğini açıklar. Kullanıcı adı, şifre... gibi.

**<textarea>**: kullanıcının çok satırlı metin girişini yapmasını sağlar. Yorum gibi uzun metinler için idealdir.

**<button>**: kullanıcının tıklamasıyla bir işlemi başlatmak için kullanılır. Genellikle form verilerini göndermek için kullanılır ama form dışında da kullanılabilir.

form

Yazılı bir yorum ekleyin **label**

Nelerden hoşlandınız veya neleri beğenmediniz? Bu ürünü ne için kullandınız?

textarea

button Gönder

# Formlarda submit eventi ve .preventDefault()

**Formu submit etmek:** Gönder butonuna basmak.

**Default form davranışı:** Verileri bir yere gönder & sayfayı yenile

Submit eventinin ve diğer eventlerin eğer varsa default davranışını engellemek için **preventDefault()** metodu kullanılır.

```
const form = document.querySelector("form");

input.addEventListener("submit", (event) => {
  event.preventDefault();
  //istenen aksiyon
});
```

# input eventi

Textarea gibi alanların her değişiminde **input eventi** tetiklenir.

Input event'i ile yazılan bilgiyi(value) alabiliriz.

```
const input = document.querySelector("input");

input.addEventListener("input", (event) => {
  console.log(event.target.value)
});
```

# Disabled(Etkisiz) butonlar

**disabled** attribute'u ile butonu etkisiz hale getirip, zorunlu alanlar tamamlanmadan formun gönderilmesini engelleyebiliriz.

Disabled değeri **true** ise butona tıklanamaz, false ise tıklanabilir.

```
<button type="button" disabled>Kaydet</button>
```

Kaydet

```
<button type="button">Kaydet</button>
```

Kaydet

# Klavye eventleri ile çalışmak

Klavye eventleri genelde **direkt** document'a eklenir.

**keydown**: Kullanıcı bir tuşa bastı

**keyup**: Kullanıcı bir tuşa basmayı bıraktı

```
document.addEventListener('keyup', function(event) {  
  if (event.key === "B") {  
    // 'B' tuşuna basıldığında yapılacak işlemler yer alır  
  }  
});
```

# Component Kavramı

Birbirine benzeyen, yeniden kullanılabilen, tekrar eden şeyler. **Bileşen.**

- X(eski adıyla twitter)'daki postlar,



- amazon'un ana sayfasındaki reklam kutuları vs.



# Component Oluşturan Fonksiyonlar

- Component'ler tekrar eden yapılardır.
- JS'de tekrar eden kodlar için fonksiyonları kullanırız.
- Fonksiyonlar; string, number v.b. tipinde verileri  
dönebildiği(return) gibi elementleri de dönebilir.
- .createElement(tag) metodu ile yeni element yaratabilir,  
.append(element)/.appendChild(element)/.prepend(element)  
gibi metodlar ile yarattığımız elementleri birbirine  
icerisine ekleyebiliriz.

**Kısaca; bir component'i  
bir fonksiyon ile yaratıp,  
return edebiliriz.**

# Parametre Alarak Component Oluşturan Fonksiyonlar

- X'de post'lar benzer yapıdadır ama içindeki veriler farklıdır.
- Component'ı fonksiyon ile oluşturabiliriz.
- Fonksiyonlar da parametre alarak, farklı çıktılar üretebilir.

**Kısaca; fonksiyonlara gerekli verileri parametre olarak ileterek içerikleri farklı olan ama benzer yapıları(component'ları) oluşturabiliriz.**

```
function buttonCreator(buttonText){  
  const button = document.createElement('button');  
  button.textContent = buttonText;  
  button.classList.add('button');  
  
  button.addEventListener('click', (e) => {  
    console.log('clicked!');  
  });  
  
  return button;  
}  
  
let firstButton = buttonCreator('Button 1');  
let secondButton = buttonCreator('Button 2');  
  
parent.appendChild(firstButton);  
parent.appendChild(secondButton);
```

# Genel Veri Yapısı ve Parametrenin Veri Türü

- Parametreleri tek tek vermek pratik değildir.
- Parametreyi bir **obje** olarak vermek iyi bir pratiktir.

```
{  
  avatar: "https://i.pravatar.cc/150?img=3",  
  author: "Trey Martinez",  
  message: "Her fragrance of choice was fresh garlic.",  
}
```

# Component'ları Döngülerde Kullanmak: **for, .forEach(...), map(...)**

- Elimizdeki veri, objelerden oluşan bir array olabilir.
- Bu array'da döngü yaparak, her bir veri için ayrı ayrı 'component oluştururan fonksiyon'u çalıştırabiliriz.

# Component'ları Döngülerde Kullanmak: for, .forEach(...), map(...)

Hedef

Yorumlarınız



Troy Williams

Hepsi harika görünüyor,  
bravo!



Troy Williams

Hepsi harika görünüyor,  
bravo!

Elimizdeki data

```
const yorumlar = [
  {
    avatar: "https://i.pravatar.cc/150?img=3",
    author: "Trey Martinez",
    text: "Her fragrance of choice was fresh garlic.",
  },
  {
    avatar: "https://i.pravatar.cc/150?img=4",
    author: "Brigitte Kerr",
    text: "I'm a living furnace.",
  },
  {
    avatar: "https://i.pravatar.cc/150?img=5",
    author: "Thaddeus Walton",
    text: "The delicious aroma from the kitchen was ruined by cigarette smoke.",
  },
  {
    avatar: "https://i.pravatar.cc/150?img=6",
    author: "Aron Giles",
    text: "Jason didn't understand why his parents wouldn't let him sell his little sister at the garage sale.",
  },
];
```

Yapabileceğimiz döngüler

```
//for ve forEach() bir şey return etmez. Direk çalıştırırız
for ( let i = 0; i < yorumlar.length; i++){
  const element = createYorum(yorumlar[i]);

  document.querySelector(".yorumlar").appendChild(element);
}

yorumlar.forEach( (yorum) => {
  const element = createYorum(yorum);

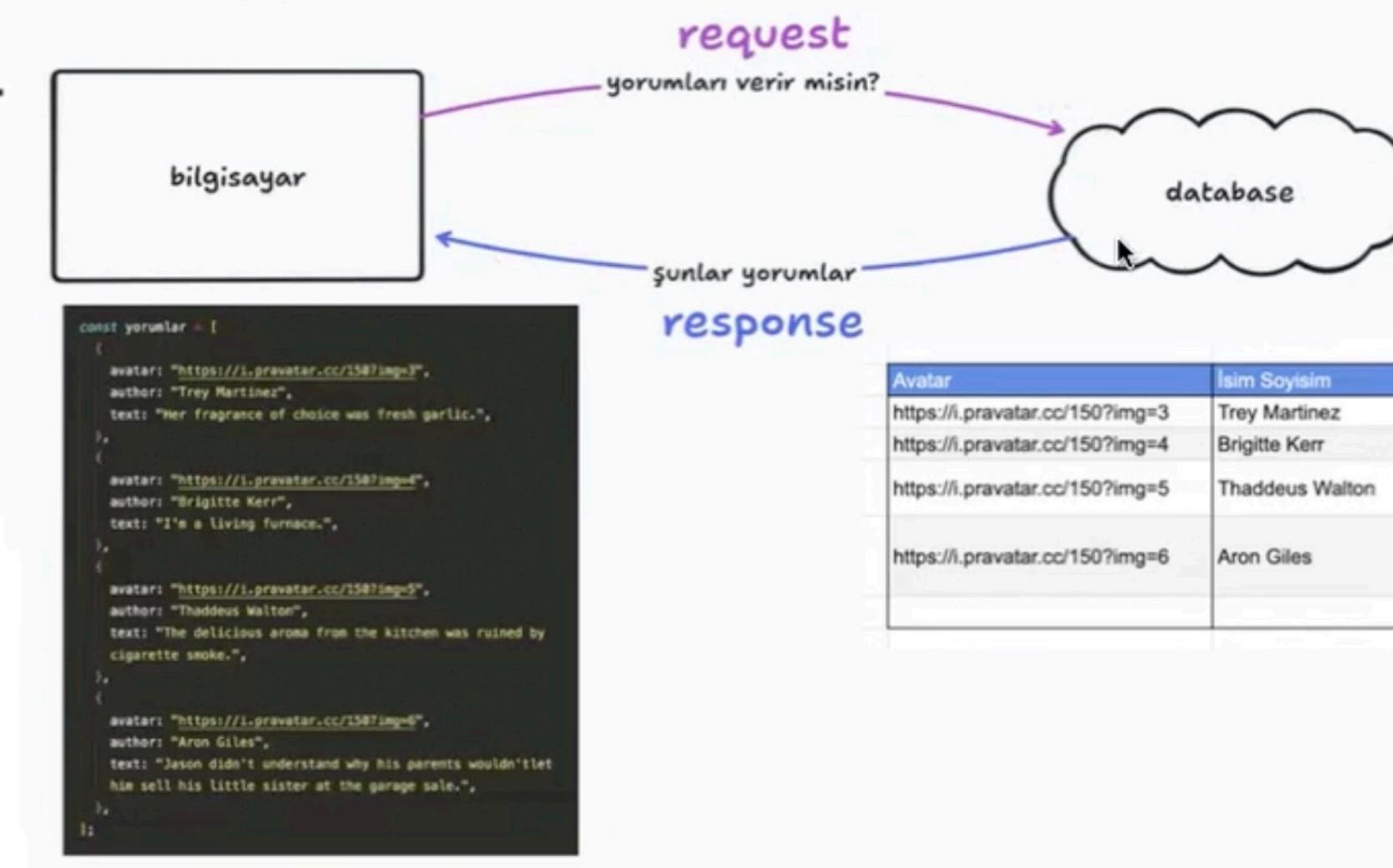
  document.querySelector(".yorumlar").appendChild(element);
})

//.map() array return eder. Bu yüzden bir değişkende saklanabilir.
const tumYorumlar = yorumlar.map( (yorum) => {
  const element = createYorum(yorum);
  return element;
})

document.querySelector(".yorumlar").append(...tumYorumlar)
```

# Ekstra: Array of objects, database ve component ilişkisi

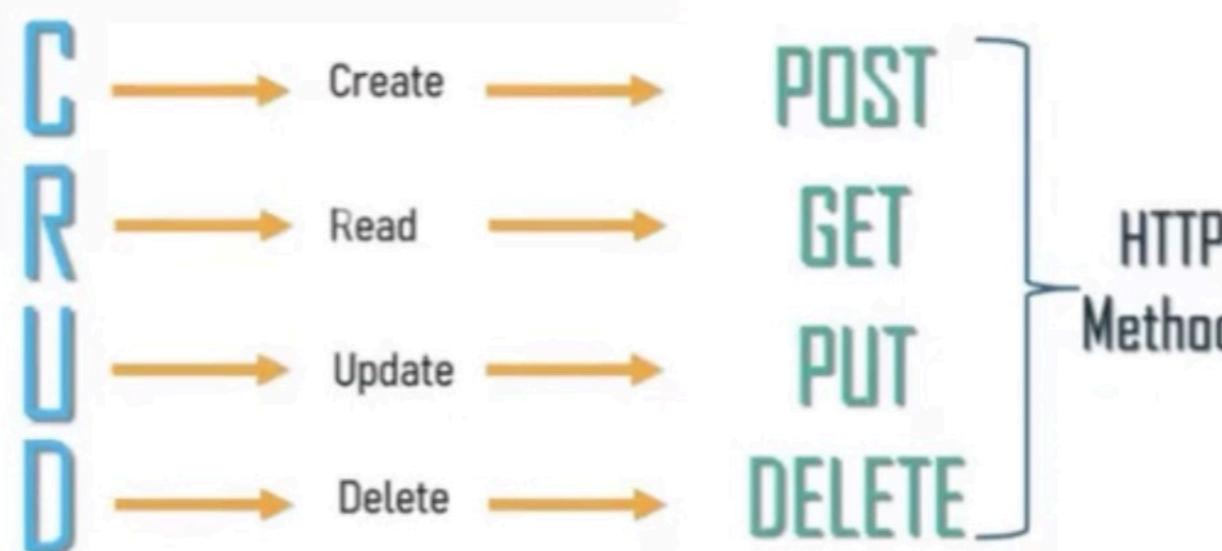
- Elimizdeki veriler bir veritabanının'da(database, kısaca DB'de) saklanır.
- Uygulamamız veritabanımızın bulunduğu server'a istek atar.
- Veriler, DB'den response olarak gelir.



# Request Tanımı ve Çeşitleri

Request, bir **istemcinin(client)** bir **sunucudan(server)** bilgi veya işlem talep ettiği bir işlemi ifade eder.

- **GET, POST, DELETE** ve **PUT**, HTTP protokolünün sağladığı dört temel request türüdür.



- **Request**'de temelde 3 bilgi olur:
  1. **Host:** Request hangi adrese yapılacak?
  2. **Method:** GET/POST/DELETE/PUT
  3. **Body?**: Request bilgi iletecekse gereklidir (yeni tweet atmak gibi, login bilgilerini göndermek gibi)

# Response Tanımı ve Status Kodları

Response objesi, gelen serverdan gelen cevap hakkında detaylı bilgi içerir.

- 2 tanesi çok önemli: **data** ve **status**
  1. **data:** Serverdan istediğimiz veriler bunun içindedir
  2. **status:** Serverdan gelen yanıtın niteliği

## Response objesi örneği

```
▼ {data: Array(2), status: 200, statusText: '', headers: n, config: {...}, ...} i
  ► config: {transitional: {...}, adapter: Array(2), transformRequest: Array(1), tr
  ▼ data: Array(2)
    ► 0: {id: '2', author: 'Alejandro Escamilla', width: 5000, height: 3333, url:
    ► 1: {id: '3', author: 'Alejandro Escamilla', width: 5000, height: 3333, url:
    ► length: 2
    ► [[Prototype]]: Array(0)
  ► headers: n {cache-control: 'private, no-cache, no-store, must-revalidate', co
  ► request: XMLHttpRequest {onreadystatechange: null, readyState: 4, timeout: 0,
  ▲ status: 200
  ▲ statusText: ""
  ► [[Prototype]]: Object
```

veriler burada

HTTP STATUS CODES	
2xx Success	
200	Success / OK
3xx Redirection	
301	Permanent Redirect
302	Temporary Redirect
304	Not Modified
4xx Client Error	
401	Unauthorized Error
403	Forbidden
404	Not Found
405	Method Not Allowed
5xx Server Error	
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout

# Senkron / Asenkron / Await Kavramları

- Yazılan kodlar yukarıdan aşağıya sırayla(**senkron**) çalışır. Çalışan fonksiyonun/döngünün -kodun- ne kadar uzun süreceği önemli değildir. Bitince sıradaki çalışır.
- Uygulamayı bekletmemek için bazı kodların paralelde(**asenkron**) çalışmasını isteyebiliriz. Server istekleri gibi... Ne kadar süreceğini bilemeyiz. (Server yoğunluğu, veri çokluğu v.s.)
  - Bazı durumlarda async bir kodu beklememiz gerekebilir. Bu durumda o kodun başına **await** ekleriz.
  - Await'i sadece async bir fonksiyonun içinde kullanabiliriz.

# Promise Kavramı

Asenkron kodları yönetmek için geliştirilmiş bir yapıdır. *callback*'lerin sıkıntılı yönlerini düzeltmek amacıyla geliştirilmiştir.

- Promise istenilen görevi yerine getirdiğinde değeri değişmez (*immutable*)
- Sadece bir kere **başarıya** (*resolved*) ulaşır, veya **başarısız** (*rejected*) olur.
- Öngörülemeyen hatalar otomatik olarak *Promise*'i **başarısız** (*rejected*) sonuca götürür.
- Yapısı gereği, gelecekteki bir değerin göstergesi olduğundan daha güvenilirdir.

Promise durumları: **pending - fulfilled - rejected**

# Promise Kavramı

Asenkron kodları yönetmek için geliştirilmiş bir yapıdır. *callback*'lerin sıkıntılı yönlerini düzeltmek amacıyla geliştirilmiştir.

- Promise istenilen görevi yerine getirdiğinde değeri değişmez (*immutable*)
- Sadece bir kere **başarıya** (*resolved*) ulaşır, veya **başarısız** (*rejected*) olur.
- Öngörülemeyen hatalar otomatik olarak *Promise*'i **başarısız** (*rejected*) sonuca götürür.
- Yapısı gereği, gelecekteki bir değerin göstergesi olduğundan daha güvenilirdir.

Promise durumları: **pending - fulfilled - rejected**

# Promise Kavramı

Serverlar response'ları promise olarak üretirler, **biz onları tüketiriz.**

```
aPromiseObject
  .then((response) => {
    // asenkron iş başarıyla tamamlandı!
  })
  .catch((error) => {
    // asenkron iş başarısız bir şekilde tamamlandı
  })
  .finally(() => {
    // asenkron tamamlandı
});
```

# API Kavramı

**API (Application Programming Interface)**, uygulamaların birbiriyle veri paylaşmasına izin veren arayüzlerdir.

- Ücretli, ücretsiz veya limitli(1 saatte 10 istek yapabilirsin) olabilirler.
- 2 uygulama arasında ortak bir dil oluştururlar. (Request formatı ve Response formatı)
- Kullanım alanlarından bazıları:
  - Sosyal medya kimlik bilgilerini kullanan uygulamalarda oturum açma izni verildiğinde
  - Facebook sunucularından size ait verileri çekerken
  - E-ticaret işlemlerinde ödeme yaparken banka ile iletişime geçildiğinde
  - Hava durumu uygulamalarında
  - Anlık borsa verilerinin takip edildiği uygulamalarda

# Axios Kütüphanesi <https://axios-http.com/>

Promise tabanlı bir HTTP client kütüphanesidir.

- Bir web sayfasına eklenerek, bu sayfanın sunucu ile iletişim kurmasını sağlar.
- Response **promise** olarak gelir.

# Axios Kütüphanesi **Kullanımı**

## 1. Projeye eklenir:

- HTML dosyasına script tagi ile

```
<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
```

- Spoiler:** ya da npm(node package manager) ile projeye eklenir ve import edilir.

```
› npm install axios
```

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

# Axios Kütüphanesi Kullanımı

2. "axios" anahtar kelimesi ile kullanılır.

a- **Promise** Kullanımı

```
axios
  .get(URL)
  .then(function (response) {
    // Başarılı olma durumunda burası çalışır
    console.log(response);
  })
  .catch(function (error) {
    // hata alma durumunda burası çalışır
    console.log(error);
  })
  .finally(function () {
    // her zaman çalışır.
    // Başarılı da olsa hata da verse...
  });

```

b- **await** ile kullanımı (Dikkat: async function içinde

```
async function getUser() {
  try {
    const response = await axios.get(URL);
    console.log(response);
  } catch (error) {
    console.error(error);
  }
}
```

# Ekstra: Dev Tools Network Tab'ı

Dev tools'daki network tab'ında yapılan isteklerin(API istekleri, web sayfası isteği, image istekleri v.b.) listesini bulabiliyoruz.

Fetch/XHR olarak filtreleyerek sadece API isteklerine bakabiliyoruz.

Dikkat: Bu tab açıldıktan sonra yapılan istekleri kaydeder.

The screenshot shows the Network tab in the Chrome DevTools. A specific request for '/v2/list?page=2&limit=2' is selected. The Headers section on the left shows various request headers like authority, method, path, scheme, Accept, etc. An arrow points from the 'Request Headers' section in the main area to a detailed view of the same headers on the right, which lists them with their corresponding values. The right panel also includes sections for Response Headers and Request Headers.

Name	Value
:authority	picsum.photos
:method	GET
:path	/v2/list?page=2&limit=2
:scheme	https
Accept	application/json, text/plain, */*
Accept-Encoding	gzip, deflate, br
Accept-Language	en-US,en;q=0.9,tr;q=0.8
Origin	https://vitejsvite3dcprf-d443--5173--ca1b5e18.local-credentialless.webcontainer.io
Referer	https://vitejsvite3dcprf-d443--5173--ca1b5e18.local-credentialless.webcontainer.io/
Sec-Ch-Ua	"Not_A Brand";v="8", "Chromium";v="120", "Google Chrome";v="120"
Sec-Ch-Ua-Mobile	?0
Sec-Ch-Ua-Platform	'macOS'
Sec-Fetch-Dest	empty
Sec-Fetch-Mode	cors
Sec-Fetch-Site	cross-site
User-Agent	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36

## Ekstra: REST API Kavramı

**REST**(Representational State Transfer), modern web tabanlı uygulamalar oluşturmak için sıkılıkla kullanılan bir mimari model biçimidir.

- Belli [prensiplere](#) uygun hazırlanması öğrenmeyi ve uygulamalara entegrasyonunu kolaylaştırır.
- Genelde dokümantasyonları da çok düzenli ve anlaşılırıdır.
- Bu yüzden oldukça kullanışlıdır.
- Hafif bir yapıda olması da onu en çok tercih edilen API türlerinden biri yapmıştır.