

WEB PROGRAMLAMA I ■ HAFTA 13

Asenkron JavaScript, Fetch API ve JSON

Senkron/asenkron farkı ■ AJAX ■ / ■ JSON

Prof. Dr. Özal YILDIRIM

Fırat Üniversitesi — Teknoloji Fakültesi

Yazılım Mühendisliği Bölümü

Sorun: Sayfanın Tamamını Yenileme Zorunluluğu

Web'in ilk yıllarında istemci–sunucu iletişimi tamamen **senkron** idi:

- 1 Kullanıcı bir linke tıklar; tarayıcı sunucuya HTTP isteği gönderir.
- 2 Tarayıcı **donar ve bekler** — kullanıcı hiçbir şey yapamaz.
- 3 Cevap gelince mevcut sayfa **tamamen silinir**, beyaz ekran görünür, yeni sayfa sıfırdan çizilir.

⚠ Dikkat

Somut örnek: BookVerse'te bir kitaba “beğen” butonu eklediğinizi düşünün. Bu eski dünyada, sadece **beğeni sayısını bir artırmak** için sayfanın tamamının yeniden yüklenmesi gerekirdi.

Bu “tıkla–bekle–yenile” döngüsü; yavaş, kesintili ve **kötü bir kullanıcı deneyimi** yaratıyordu.

- ▶ Senkron ve asenkron çalışma modelini karşılaştırabilecek,
- ▶ AJAX yaklaşımının ne olduğunu ve neden devrim olduğunu açıklayabilecek,
- ▶ `Promise`'in üç durumunu ve zincirleme kullanımını kavrayabilecek,
- ▶ `fetch()` ile sunucudan veri çekebilecek,
- ▶ `async/await` ile asenkron kodu okunaklı yazabilecek,
- ▶ `try/catch` ile hata yönetimi yapabilecek,
- ▶ JSON formatının kurallarını ve JavaScript nesnesinden farkını açıklayabilecek,
- ▶ `JSON.parse()` / `JSON.stringify()` dönüşümlerini uygulayabileceksiniz.

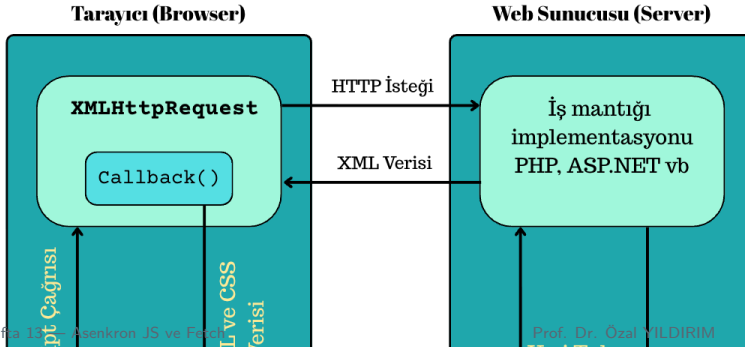
- 1 Asenkron Kavramı ve AJAX
- 2 Promise
- 3 Fetch API
- 4 JSON
- 5 Yapay Zeka ile Asenkron Kod
- 6 Özet ve Değerlendirme

Asenkron Kavramı ve AJAX

BÖLÜM 1

Tanım

AJAX (Asynchronous JavaScript and XML), kullanıcının mevcut sayfayla etkileşimini **kesintiye uğratmadan**, arka planda sunucuya küçük istekler gönderip **sadece gerekli veri parçalarını** alarak sayfayı anında güncelleme yaklaşımıdır.



AJAX Açılımı ve Restoran Analjisi

A — Asynchronous

En önemli anahtar kelime. JavaScript isteği gönderir ama **cevabı beklemez**. Kod çalışmaya devam eder. Cevap gelince yakalar ve sayfayı günceller.

J — JavaScript

Arka plan isteklerini başlatan ve gelen cevabı işleyerek **DOM'u manipüle eden** dil.

X — XML

İlk çıktığında veri formatı XML idi. Bugün yerini **JSON** aldı. Adı tarihsel olarak kaldı — aslında “AJAJ” demek daha doğru olurdu.

Restoran Analjisi

Senkron model:

Garson siparişinizi alır, mutfağa gider ve yemek hazırlanıp gelene kadar **yanınızdan hiç ayrılmadan bekler**. Bu sırada başka hiçbir şey yapamazsınız.

Asenkron (AJAX) model:

Garson siparişinizi alır, mutfağa iletir ve **siz sohbete devam ederken** başka masalarla ilgilenir. Yemek hazır olduğunda size getirir.

Senkron vs Asenkron: Kodda Görelim

```
1 console.log("1 - Baslangic");
2
3 setTimeout(() => {
4   console.log("2 - 2 saniye sonra");    // ASENKRON: beklemeye alinir
5 }, 2000);
6
7 console.log("3 - Bitis");
8
9 // CIKTI SIRASI:
10 // 1 - Baslangic
11 // 3 - Bitis      <- 2'yi BEKLEMEDI
12 // 2 - 2 saniye sonra
```

≡ Olay Döngüsü (Event Loop)

JavaScript **tek iş parçacıklıdır** (single-threaded). Asenkron işler “kenara alınır”, ana kod biter, **sonra** sırası gelenler çalışır.

⚠ Dikkat

Bu yüzden: Ağ isteğinin sonucunu **istekten hemen sonraki satırda** kullanamazsınız. Veri henüz gelmemiştir. Sonucu **callback**, **.then()** veya **await** ile beklemeniz gerekir.



Promise

BÖLÜM 2

Promise: Geleceğin Sonucunu Temsil Eden Nesne

Tanım

Promise (Söz), en basit tanımıyla **gelecekte tamamlanacak bir işlemin sonucunu** temsil eden bir nesnedir. Asenkron bir işlem başlattığınızda JavaScript size hemen bir “söz” verir.

Gelecekteki değer
erişilebilir olmadan önce

Beklemede
(Pending)

Asenkron görev bitti

Tamamlandı
(Settled)

Başarılı

Başarısız

.then() ve .catch() Zinciri

```
1 fetch("data/kitaplar.json")
2   .then(cevap => {
3     if (!cevap.ok) throw new Error(`HTTP hatasi: ${cevap.status}`);
4     return cevap.json();           // JSON'a cevir - BU DA bir Promise
5   })
6   .then(veri => {
7     console.log(veri);             // asil verimiz
8     listeyiCiz(veri);
9   })
10  .catch(hata => {
11    console.error("Veri alinamadi:", hata);
12    hataGoster("Kitaplar yuklenemedi. Lutfen tekrar deneyin.");
13  })
14  .finally(() => {
15    yukleniyorGizle();             // basarili da olsa, hata da olsa calisir
16  });
```

Bir işlem bittiğinde ne olacağını `.then()`, hata olursa ne yapılacağını `.catch()` ile **zincirleme ve net** bir şekilde belirtiriz.



Fetch API

BÖLÜM 3

fetch(): Modern Ağ İsteği

AJAX'in ilk yıllarında istekler XMLHttpRequest (XHR) ile yapılırdı — sözdizimi karmaşık, okunması zordu ve “**callback cehennemi**” denen iç içe fonksiyon yığınlarına yol açardı.

```
1 // En basit hali - GET istegi
2 fetch("data/kitaplar.json")
3   .then(cevap => cevap.json())
4   .then(veri => console.log(veri));
5
6 // Tam yazım - secenekli
7 fetch("https://api.ornek.com/kitaplar", {
8   method: "POST",
9   headers: {
10     "Content-Type": "application/json",
11     "Authorization": "Bearer TOKEN"
12   },
13   body: JSON.stringify({ ad: "Yeni Kitap", fiyat: 89.90 })
14 });
```

Fetch API, tarayıcılarda yerleşik gelir, **Promise tabanlıdır** ve XHR'a göre çok daha kolaydır.

fetch'in Bilinmesi Gereken Tuzağı

⚠ Dikkat

fetch, HTTP hata kodlarında (404, 500) reject ETMEZ! Sadece **ağ tamamen başarısız olursa** (internet yok, DNS hatası) reddeder. 404 bile “başarılı bir cevap” sayılır.

```
1 // YANLIŞ - 404 hatasını yakalamaz
2 fetch("olmayan-dosya.json")
3   .then(c => c.json())
4   .then(v => console.log(v))
5   .catch(h => console.error(h));    // BURAYA GİRMEZ
6
7 // DOĞRU - cevabi kontrol edin
8 fetch("olmayan-dosya.json")
9   .then(cevap => {
10     if (!cevap.ok) {
11       throw new Error(`HTTP ${cevap.status}: ${cevap.statusText}`);
12     }
13     return cevap.json();
14   })
15   .then(v => console.log(v))
16   .catch(h => console.error("Hata:", h.message));
```

Promise zincirlerini **senkron kod gibi** yazmanızı sağlar.

.then() zinciri

```
function kitaplariGetir() {  
  return fetch("data/kitaplar.json")  
    .then(c => {  
      if (!c.ok) throw new Error(c.status);  
      return c.json();  
    })  
    .then(v => {  
      listeyiCiz(v);  
      return v;  
    })  
    .catch(h => console.error(h));  
}
```

async/await — tercih edin

```
async function kitaplariGetir() {  
  try {  
    const cevap = await fetch("data/kitaplar.json");  
    if (!cevap.ok) throw new Error(cevap.status);  
  
    const veri = await cevap.json();  
    listeyiCiz(veri);  
    return veri;  
  } catch (hata) {  
    console.error(hata);  
  }  
}
```

💡 İpucu

Üç kural: (1) `await` sadece `async` fonksiyon içinde kullanılabilir. (2) `async` fonksiyon **her zaman bir Promise döndürür**. (3) Hata yönetimi için `try/catch` kullanılır.

Yükleniyor ve Hata Durumları

```
1 const liste      = document.querySelector(".kitap-listesi");
2 const yukleniyor = document.querySelector("#yukleniyor");
3
4 async function kitaplariYukle() {
5     yukleniyor.hidden = false;           // 1) Yukleniyor goster
6     liste.innerHTML = "";
7
8     try {
9         const cevap = await fetch("data/kitaplar.json");
10        if (!cevap.ok) throw new Error(`Sunucu hatasi (${cevap.status})`);
11
12        const kitaplar = await cevap.json();
13
14        if (kitaplar.length === 0) {       // 2) BOS durum
15            liste.innerHTML = "<p>Henuz kitap eklenmemis.</p>";
16            return;
17        }
18        listeyiCiz(kitaplar);              // 3) Basarili
19
20    } catch (hata) {                       // 4) HATA durumu
21        liste.innerHTML = `<p class="hata">Kitaplar yuklenemedi: ${hata.message}</p>`;
22    } finally {
23        yukleniyor.hidden = true;         // 5) Her durumda gizle
24    }
```

BookVerse - Kitap Tutkunların

127.0.0.1:53238/index.html

☆ 0

← → ↺ ⓘ

⌵



BookVerse

Menü

BookVerse'e Hoş Geldiniz

Burasi, kitaplar hakkında yorumlarinizi paylasabileceginiz, yeni yazarlar keşfedebileceginiz ve kitap tutkunlariyla bulusabileceginiz bir platformdur.

DevTools is now available in Turkish

Don't show again Always match Chrome's la

Console >> 1

top Filter

Default levels No Issues

Kitap verileri basariyla
cekildi: [main.js:26](#)

(3) [{...}, {...}, {...}]

0:
author: "Paulo Coelho"
rank: 1
title: "Simyaci"
[[Prototype]]: Object

1:
author: "José Mauro de Vasconcelos"
rank: 2
title: "Seker Portakali"
[[Prototype]]: Object

2:
author: "George Orwell"
rank: 3
title: "1984"
[[Prototype]]: Object

Web Programlama I | Hafta 13 — Asenkron JS ve Fetch

13 / 32



JSON

BÖLÜM 4

Tanım

JSON (JavaScript Object Notation), hem **insanlar tarafından okunabilen** hem de **makinelere tarafından ayrıştırılabilen** metin tabanlı bir veri alışverişi formatıdır.

```
{
  "siteAdi": "BookVerse",
  "toplamKitap": 4,
  "aktif": true,
  "guncelleme": null,
  "kitaplar": [
    { "id": 1, "ad": "Simyaci", "yazar": "Paulo Coelho", "fiyat": 89.90 },
    { "id": 2, "ad": "Kürk Mantolu", "yazar": "Sabahattin Ali", "fiyat": 45.00 }
  ]
}
```

Sözdizimi doğrudan **JavaScript nesne sözdiziminden** türetilmiştir. Hafif ve temiz yapısı, onu hızla **web API'lerinin fiili standardı** yaptı.

JSON Kuralları ve JS Nesnesinden Farkı

✓ JSON Kuralları

- ▶ Veriler **anahtar: değer** çiftleri halinde
- ▶ Anahtarlar **her zaman çift tırnak** içinde
- ▶ Çiftler **virgül** ile ayrılır
- ▶ Nesneler { } içinde
- ▶ Diziler [] içinde

✗ Geçersiz JSON

```
{
  ad: 'Simyaci',      // yorum
  fiyat: 89.90,
}
```

✗ JSON'da OLMAZ

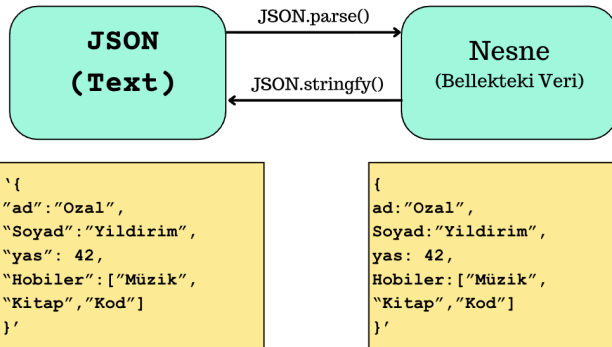
- ▶ **Yorum satırı** (// veya /* */)
- ▶ **Tek tırnak** ('ad')
- ▶ Tırnaksız anahtar (ad:)
- ▶ Son öğeden sonra **virgül**
- ▶ **undefined**, **fonksiyon**, **Date**

✓ Geçerli JSON

```
{
  "ad": "Simyaci",
  "fiyat": 89.90
}
```

Doğrulama aracı: jsonlint.com ■ VS Code .json dosyalarında hatayı **altını çizerek** gösterir.

JSON.parse() ve JSON.stringify()



```
1 // METIN -> JavaScript NESNESİ  
2 const metin = '{"ad": "Simyaci", "fiyat": 89.90}';  
3 const nesne = JSON.parse(metin);  
4 console.log(nesne.ad); // "Simyaci"  
5  
6 // JavaScript NESNESİ -> METİN  
7 const kitap = { ad: "Simyaci", fiyat: 89.90 };
```

Uygulama: BookVerse Verisini Dışarı Taşımak

1. Adım — data/kitaplar.json

```
[
  { "id": 1, "ad": "Simyaci",      "yazar": "Paulo Coelho",  "fiyat": 89.90,
    "puan": 8.4, "tur": "roman", "stok": 12, "kapak": "images/simyaci.jpg" },
  { "id": 2, "ad": "Kurk Mantolu", "yazar": "Sabahattin Ali", "fiyat": 45.00,
    "puan": 9.1, "tur": "roman", "stok": 0,  "kapak": "images/kurk.jpg" },
  { "id": 3, "ad": "Tutunamayanlar", "yazar": "Oguz Atay",    "fiyat": 120.00,
    "puan": 9.5, "tur": "roman", "stok": 3,  "kapak": "images/tutunamayanlar.jpg" }
]
```

2. Adım — js/main.js

```
1  let kitaplar = [];                // global veri
2
3  async function veriYukle() {
4    try {
5      const cevap = await fetch("data/kitaplar.json");
6      if (!cevap.ok) throw new Error(`HTTP ${cevap.status}`);
7      kitaplar = await cevap.json();
8      listeyiCiz(kitaplar);
9    } catch (hata) {
10     liste.innerHTML = `<p class="hata">Veriler yuklenemedi.</p>`;
11     console.error(hata);
12   }
13 }
14
15 document.addEventListener("DOMContentLoaded", veriYukle);
```

Gerçek Bir API'den Veri Çekmek

```
1 // Ucretsiz test API'leri
2 // https://jsonplaceholder.typicode.com/posts
3 // https://openlibrary.org/search.json?q=simyaci
4 // https://api.github.com/users/KULLANICI
5
6 async function kitapAra(terim) {
7   const url = `https://openlibrary.org/search.json?q=${encodeURIComponent(terim)}&limit=10`;
8
9   try {
10    const cevap = await fetch(url);
11    if (!cevap.ok) throw new Error(`API hatasi: ${cevap.status}`);
12
13    const veri = await cevap.json();
14    return veri.docs.map(k => ({
15      ad: k.title,
16      yazar: k.author_name?.[0] ?? "Bilinmiyor",
17      yil: k.first_publish_year ?? "-"
18    }));
19  } catch (hata) {
20    console.error(hata);
21    return [];
22  }
```

❌ Hata: “CORS policy has blocked”

`fetch`, `file://` **protokolünde çalışmaz**. HTML dosyanızı çift tıklayıp açarsanız bu hatayı alırsınız. **Çözüm:** Her zaman **Live Server** kullanın (`http://127.0.0.1:5500`). *Hafta 2'de bunu neden vurguladığımızı şimdi anladınız.*

🛡️ CORS Nedir?

Cross-Origin Resource Sharing. Bir tarayıcı güvenlik kuralıdır: Bir siteden **başka bir alan adına** istek gönderirken, hedef sunucunun **izin vermesi** gerekir.

💡 İpucu

Hedef sunucu şu başlığı göndermelidir:
`Access-Control-Allow-Origin: *`
Bu başlık **sunucu tarafında** ayarlanır — istemcide **çözülemez**. (Web-2'de öğreneceğiz.)

Network Sekmesi

- ▶ **Fetch/XHR** filtresini seçin
- ▶ İsteğe tıklayın:
- ▶ **Headers** — URL, metot, durum kodu
- ▶ **Payload** — gönderilen veri
- ▶ **Preview** — **JSON'u ağaç halinde** gösterir
- ▶ **Response** — ham cevap metni
- ▶ **Timing** — ne kadar sürdü

Gerçekçi Test

Network sekmesinde **"Slow 3G"** kısıtlamasını seçin. Yükleniyor göstergeniz **gerçekten çalışıyor mu?**

Kullanıcı 3 saniye boyunca boş ekran mı görüyor? **Bu test olmadan** yükleniyor durumunu doğru tasarlayamazsınız.

Hafta 1'e dönüş: Burada gördüğünüz istek/cevap, durum kodu ve başlıklar — ilk hafta öğrendiğimiz **HTTP döngüsünün** ta kendisi.

Birden Fazla İsteği Yönetmek

```
1 // SIRAYLA - toplam süre = 1s + 1s + 1s = 3 saniye
2 const kitaplar = await (await fetch("data/kitaplar.json")).json();
3 const yazarlar = await (await fetch("data/yazarlar.json")).json();
4 const turler = await (await fetch("data/turler.json")).json();
5
6 // PARALEL - toplam süre = en yavaş olan = 1 saniye
7 const [kitaplar2, yazarlar2, turler2] = await Promise.all([
8   fetch("data/kitaplar.json").then(c => c.json()),
9   fetch("data/yazarlar.json").then(c => c.json()),
10  fetch("data/turler.json").then(c => c.json())
11 ]);
12
13 // Biri hata verse bile diğerleri devam etsin
14 const sonuclar = await Promise.allSettled([istek1, istek2, istek3]);
15 sonuclar.forEach(s => {
16   if (s.status === "fulfilled") console.log(s.value);
17   else console.error(s.reason);
18 });
```



Dikkat

Debounce: Her Tuşa İstek Atmayın

Canlı aramada her tuş vuruşunda API'ye istek atmak, **sunucuyu boşuna yorar** ve kotanızı tüketir.

```
1 function debounce(fn, gecikme = 400) {
2   let zamanlayici;
3   return (...args) => {
4     clearTimeout(zamanlayici);
5     zamanlayici = setTimeout(() => fn(...args), gecikme);
6   };
7 }
8
9 const aramaKutusu = document.querySelector("#arama");
10
11 // Kullanıcı 400ms yazmayı bırakınca CALISIR
12 aramaKutusu.addEventListener("input", debounce(async (e) => {
13   const sonuclar = await kitapAra(e.target.value);
14   listeyiCiz(sonuclar);
15 }, 400));
```

“simyacı” yazmak **7 tuş vuruşu** → debounce olmadan **7 istek**; debounce ile **1 istek**.

Uygulama: Sepeti localStorage'da Saklamak

```
1  const SEPET_ANAHTAR = "bookverse-sepet";
2
3  function sepetiOku() {
4    try {
5      return JSON.parse(localStorage.getItem(SEPET_ANAHTAR)) ?? [];
6    } catch {
7      return []; // bozuk JSON varsa cokme
8    }
9  }
10
11 function sepetiYaz(sepet) {
12   localStorage.setItem(SEPET_ANAHTAR, JSON.stringify(sepet));
13   sepetSayisiGuncelle(sepet);
14 }
15
16 function sepeteEkle(kitapId) {
17   const sepet = sepetiOku();
18   const mevcut = sepet.find(s => s.id === kitapId);
19
20   if (mevcut) mevcut.adet += 1;
21   else sepet.push({ id: kitapId, adet: 1 });
22
23   sepetiYaz(sepet);
24 }
25
26 function sepetToplami() {
27   return sepetiOku().reduce((toplam, s) => {
28     const kitap = kitaplar.find(k => k.id === s.id);
29     return toplam + (kitap ? kitap.fiyat * s.adet : 0);
30   }, 0);
```

Sık Yapılan Asenkron Hataları

Belirti	Sebebi ve Çözümü
Konsolda <code>{<pending>}</code>	Promise <code>await</code> unutulmuş . Fonksiyon <code>async</code> mi?
<code>await</code> is only valid in <code>async</code> functions	<code>await</code> , <code>async</code> olmayan fonksiyonda kullanılmış.
Unexpected token <code><</code> in JSON	Sunucu JSON değil HTML döndürdü — genelde 404 sayfası. <code>cevap.ok</code> kontrolü ekleyin.
CORS hatası	<code>file://</code> ile açtınız. Live Server kullanın.
Veri geldi ama ekranda yok	<code>fetch</code> asenكرون — çizim kodu veri gelmeden çalışmış.
Hata mesajı yerine boş sayfa	<code>catch</code> bloğu yok veya sadece <code>console.error</code> yapıyor. Kullanıcıya da gösterin .
<code>JSON.parse</code> çöküyor	Gelen metin geçerli JSON değil. <code>try/catch</code> ile sarın.



Yapay Zeka ile Asenkron Kod

BÖLÜM 5

Asenkron Kodda Yapay Zekanın Zayıf Noktaları

Sık Ürettiği Sorunlar

- ▶ `cevap.ok` kontrolünü atlama — 404'ü başarı sayma
- ▶ `try/catch` yazmama
- ▶ Yükleniyor ve boş durumlarını ihmal etme
- ▶ `await` unutup `Promise` nesnesi basılması
- ▶ Kullanıcıya **ham hata mesajı** gösterme
- ▶ `encodeURIComponent` kullanmama

Prompt Kısıtlarınız

"cevap.ok kontrolü ekle." "Dört durumu da ele al: yükleniyor, başarılı, boş, hata." "Kullanıcıya teknik hata mesajı gösterme, anlaşılır metin yaz." "async/await kullan, .then() zinciri yazma."

En sinsi hata: `await` unutulduğunda kod **hata vermez** — konsola `Promise {<pending>}` basar. İş çalışıyor sanırsınız.

Yapay Zeka İstemi (Prompt) — Fetch Kodu İncelemesi

Asagidaki fetch kodunu denetle:

[KODUNUZU YAPISTIRIN]

Su kontrolleri yap, her birine [VAR] / [EKSIK] etiketi ver:

1) cevap.ok kontrolu yapiliyor mu? 2) try/catch veya .catch() var mi? 3) Yukleniyor gostergesi var mi? 4) Bos veri durumu ele alinmis mi? (dizi bos gelirse) 5) Kullaniciya gosterilen hata mesaji ANLASILIR mi, yoksa teknik terim mi iceriyor? 6) await unutulmus bir yer var mi? 7) URL parametreleri encodeURIComponent ile kodlanmis mi?
Her [EKSIK] icin, EKLENECEK satiri ve nereye eklenecegini goster. Tum kodu yeniden yazma.
Sonunda: "Bu kod internet baglantisi kesikken calisirsas kullanici ne gorur?" sorusunu yanitla.

Prompt 2 — JSON Doğrulama ve Dönüştürme

Yapay Zeka İstemi (Prompt) — JSON Yapısı Tasarımı

Bir kitap sitesi için JSON veri yapısı tasarlıyorum.

Su bilgileri saklamam gerekiyor: kitap adı, yazar(lar), ISBN, fiyat, indirimli fiyat, puan, yorum sayısı, tür(ler), stok durumu, kapak görseli, yayın yılı, yayınevi (ad + şehir), özet metni

GOREV: 1) Bu veriyi tutan ORNEK bir JSON dosyası oluşturun (3 kitap). 2) Her alan için doğru JSON veri tipini kullanın. (Birden fazla yazar/tür olabileceğini unutmayın.) 3) İç içe nesne ve dizi kullanımını gösterin. 4) Bu JSON'un GECERLİ olduğunu nasıl doğrularım? Aracın öner. 5) Aynı veriyi JavaScript nesnesi olarak yazsam ne DEĞİŞİRDİ? İki farkı maddeler halinde yazın.

KURAL: JSON içine YORUM SATIRI ekleme (gecersiz olur). Açıklamaları JSON'un DIŞINDA yazın.

Doğrulayın: Üretilen JSON'u jsonlint.com'a yapıştırın. Yapay zeka **sıkça** son öğeden sonra virgül bırakır.

Prompt 3 — Asenkron Akışı İzletme

Yapay Zeka İstemi (Prompt) — Çıktı Sırasını Tahmin Et

Bana asenkron JavaScript üzerine 6 adet "CIKTI SIRASI" sorusu hazırla.
Her soruda kısa bir kod parçası olsun; ben konsol çıktılarının HANGİ SIRAYLA yazılacağını tahmin edeceğim.
Konu dağılımı: - 2 soru: setTimeout ve senkron kod karışımı - 2 soru: Promise .then() zinciri - 2 soru: async/await ve await unutulmuş durumlar
KURALLAR: - Her kod en fazla 10 satır - Önce SADECE kodları yaz - Cevap anahtarında, her çıktının NEDEN o sırada olduğunu "olay döngüsü" (event loop) kavramıyla açıkla - En az bir soruda "await unutulmuş" durumu olsun ve konsola ne yazılacağını göster Türkçe yaz.

Yapay Zeka Çıktısını Doğrulama

Cevapları **konsolda gerçekten çalıştırın**. Asenkron sıralama, yapay zekanın **zaman zaman yanıldığı** bir konudur — özellikle mikrogörev/makrogörev sıralamasında.

Özet ve Değerlendirme

BÖLÜM 6

Asenkron

JavaScript tek iş parçacıklıdır; asenkron işler **beklemeye alınır**. AJAX, sayfa yenilenmeden veri almayı mümkün kıldı.

Promise

pending → fulfilled / rejected.
.then() başarı, .catch() hata, .finally()
her durumda.

Fetch & JSON

cevap.ok **kontrolü şarttır** — 404 reject etmez.
async/await + try/catch. JSON: anahtarlar
çift tırnaklı, yorum yok.

Yapay Zeka

cevap.ok ve hata durumlarını **atlar**. Prompt'a
“**dört durumu da ele al**” kısıtını ekleyin.

Altın Kural: Ağ isteği **her zaman** başarısız olabilir. Yükleniyor, başarılı, boş ve hata — **dört durumu da** tasarlayın.

Terim	Anlam	Terim	Anlam
Senkron	Sırayla, bekleyerek	<code>fetch()</code>	Modern ağ istek API'si
Asenkron	Beklemeden, arka planda	<code>cevap.ok</code>	Durum kodu 200–299 mu?
AJAX	Sayfa yenilemeden veri alma	<code>async</code>	Fonksiyonu Promise'e çevirir
Promise	Gelecekteki sonucu temsil eder	<code>await</code>	Promise'in sonucunu bekler
<code>pending</code>	Beklemede durumu	<code>try/catch</code>	Hata yakalama bloğu
<code>fulfilled</code>	Başarıyla tamamlandı	JSON	Metin tabanlı veri formatı
<code>rejected</code>	Hata ile sonuçlandı	<code>JSON.parse</code>	Metin → nesne
Event loop	Olay döngüsü	<code>JSON.stringify</code>	Nesne → metin
CORS	Kaynaklar arası paylaşım kuralı	Callback hell	İç içe geçmiş fonksiyonlar

Değerlendirme Soruları

1 Aşağıdaki kodun çıktı sırası nedir? Neden?

```
console.log("A");  
setTimeout(() => console.log("B"), 0);  
Promise.resolve().then(() => console.log("C"));  
console.log("D");
```

2 fetch bir 404 cevabı aldığında .catch() bloğu çalışır mı? Neden?

3 Promise'in üç durumunu ve her birinin ne anlama geldiğini yazınız.

4 async bir fonksiyon her zaman ne döndürür?

5 Aşağıdaki JSON'da **üç hata** vardır. Bulunuz:

```
{  
  ad: "Simyaci",  
  'yazar': "Coelho",    // yazar bilgisi  
  "fiyat": 89.90,  
}
```

6 JSON.parse() ile JSON.stringify() ne zaman kullanılır?

Laboratuvar Föyü 13

- ▶ Asenkron akış deneyleri
- ▶ `data/kitaplar.json` oluşturma
- ▶ `fetch` ile veri çekme
- ▶ Dört durumun (yükleniyor/başarılı/boş/hata) ele alınması
- ▶ Slow 3G altında test
- ▶ Gerçek bir API'den arama
- ▶ `localStorage` + JSON ile sepet
- ▶ Yapay zeka ile asenkron kod denetimi

Hazırlık

Okuma: Kitap Ünite 3.1–3.2 — “Asenkron Programlama ve AJAX”, “Veri Formatları”
Kaynaklar:

- ▶ developer.mozilla.org/docs/Web/API/Fetch_API
- ▶ javascript.info/async
- ▶ jsonplaceholder.typicode.com
- ▶ jsonlint.com

Gelecek hafta: **Sunucu tarafı** — PHP ile dönemin son halkası.

Haftaya Görüşmek Üzere!

Gelecek Hafta:

PHP ile Sunucu Tarafına Giriş

Neden sunucu gerekli? ■ PHP temelleri ■ Form işleme
() ■ Sunucu tarafı doğrulama ■ Web-2'ye köprü