WEB & YAZILIM GELİŞTİRME SERİSİ · MODÜL 7

Arkayüz & API

İstemci-sunucu modeli, HTTP ve istek-yanıt; REST API'ler, uç noktalar ve JSON; sunucu-veritabanı ilişkisi, kimlik doğrulama ve güvenlik; mimari ve üretim. Özgün diyagramlarla, dilden bağımsız.

HTTP · REST · JSON · Güvenlik · Eğitim amaçlıdır

Bu Kitap Hakkında

Bu modül, bir web uygulamasının görünmeyen ama her şeyi yöneten yüzünü — arkayüzü (backend) — herhangi bir programlama dilinden bağımsız olarak öğretir. Dört seviye ve yirmi altı bölüm boyunca istemci-sunucu modelinden HTTP, istek-yanıt ve JSON'a; REST API'lerden uç noktalara, kimlik doğrulama ve güvenliğe; oradan da önbellekleme, kuyruklar ve mimari desenlere kadar uzanır.

Her bölümde konuyu görselleştiren özgün bir diyagram (istemci-sunucu akışı, HTTP istek/yanıt panelleri, renkli API uç nokta tabloları), kavramsal kod örnekleri, 'sunucu ne yanıt döner?' kartı ve bir alıştırma yer alır. Güvenlik (HTTPS, girdi doğrulama, enjeksiyona karşı koruma, şifre hash'leme, yetki kontrolü) baştan sona vurgulanır. Bu, on altı modüllük 'Web & Yazılım Geliştirme' serisinin yedinci modülüdür ve ikinci fazı (sunucu tarafı) başlatır; buradaki kavramlar Veritabanı, PHP, Python ve C# modüllerinde somutlaşır. Bu seri eğitim amaçlıdır.

Web & Yazılım Geliştirme Serisi · Modül 7

İçindekiler

SEVİYE 1

Sunucu Tarafına Giriş

Temeller: arkayüz nedir, istemci-sunucu modeli, sunucu nasıl çalışır, HTTP, istek-yanıt, HTTP metotları, durum kodları ve JSON.

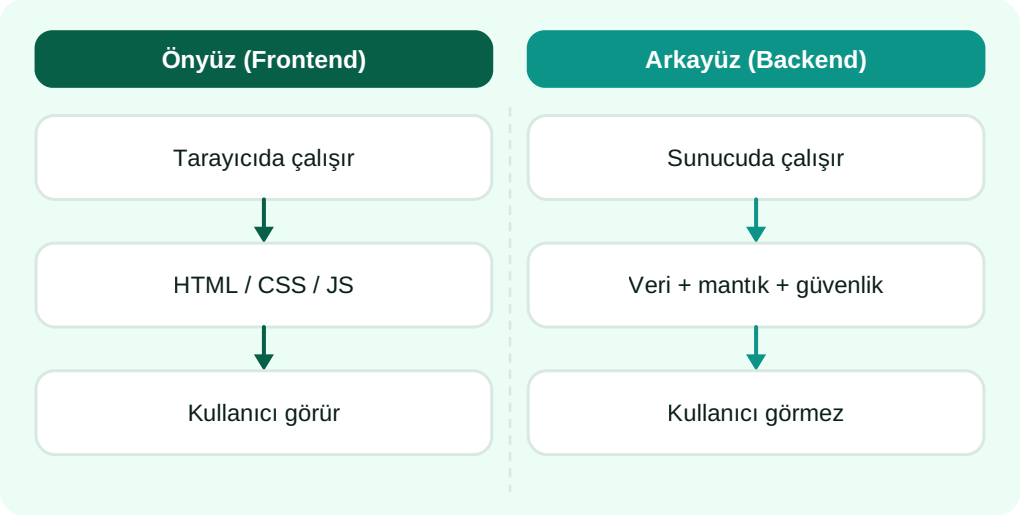
BÖLÜM 01

Arkayüz (Backend) Nedir?

Bir web uygulamasının iki yüzü vardır. Önyüz (frontend) kullanıcının tarayıcısında görünen kısımdır (HTML, CSS, JavaScript). Arkayüz (backend) ise sunucuda çalışan, görünmeyen kısımdır: veriyi saklar, mantığı yürütür, güvenliği sağlar.

Önyüz ve arkayüz

- **Önyüz:** kullanıcının gördüğü ve etkileş(tiği yüz (tarayıcıda).
- **Arkayüz:** veriyi, mantığı ve güvenliği yöneten kısım (sunucuda).
- İkisi **HTTP** üzerinden istek-yanıtla konuşur.



Şema 1.1 — Önyüz tarayıcıda, arkayüz sunucuda çalışır.

Arkayüz ne yapar?

- Veriyi **veritabanında** saklar ve getirir.
- İş mantığını çalıştırır (hesaplama, kural, doğrulama).
- Kimlik doğrulama ve **güvenliği** sağlar (gizli işler burada).

İPUCU

Basit bir kural: kullanıcıya **güvenmen gereken** her şey arkayüzde olmalıdır. Fiyat hesaplama, yetki kontrolü, ödeme — bunlar tarayıcıda (önyüzde) yapılırsa kullanıcı değiştirebilir. Bu modül, herhangi bir dilden bağımsız olarak arkayüzün "nasıl düşündüğünü" öğretir; PHP, Python ve C# modülleri bu kavramları somut dillere dökecek.

Sunucu ne yanıt döner? YANIT

Önyüz "Bana kullanıcı listesini ver" diye bir istek gönderir; arkayüz veritabanına bakar, listeyi hazırlar ve **JSON** olarak geri döner. Kullanıcı bu alışverişi görmez — sadece sonucu (ekrandaki listeyi) görür.

Alıştırma 8 dk

Önyüz/arkayüz ayır:

1. Bir alışveriş sitesinde hangi işler önyüzde, hangileri arkayüzde olmalı, listele.
2. "Ödeme tutarını hesaplama" neden arkayüzde olmalı, açıkla.
3. Arkayüzün üç temel görevini kendi cümlelerinle yaz.

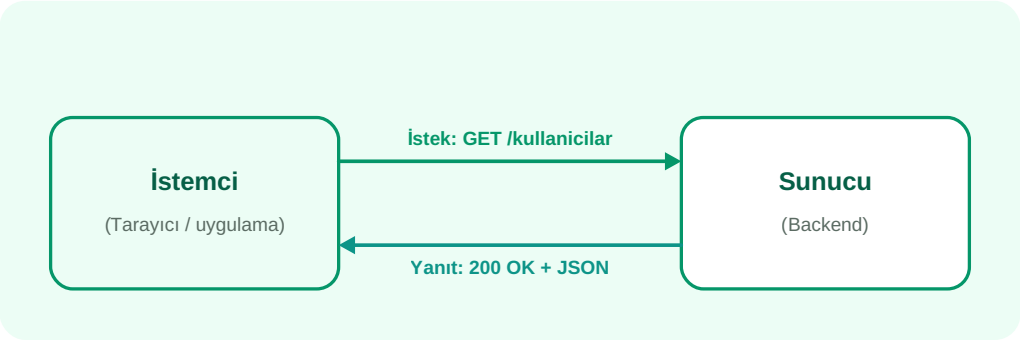
BÖLÜM 02

İstemci–Sunucu Modeli

Web, istemci–sunucu modeli üzerine kuruludur. İstemci (genelde tarayıcı) bir istek gönderir; sunucu bu isteği işler ve bir yanıt döner. Bu basit alışveriş, internetin temelidir.

İki taraf, bir konuşma

- **İstemci:** isteği başlatan taraf (tarayıcı, mobil uygulama).
- **Sunucu:** isteği karşılayıp yanıt veren taraf.
- Her şey bir **istek** ile başlar, bir **yanıt** ile biter.



Şema 2.1 — İstemci istek gönderir, sunucu yanıt döner.

Bir istek ve yanıt (kavramsal)

```
// İstemci:
GET /kullanicilar  ->  sunucuya gönderilir

// Sunucu yanıtı:
200 OK
[ { "ad": "Ayşe" }, { "ad": "Veli" } ]
```

İPUCU

Sunucu kendiliğinden istemciye bir şey göndermez; konuşmayı **her zaman istemci başlatır** (klasik web modelinde). İstemci sorar, sunucu yanıtlar. Bir sunucu aynı anda binlerce istemciye hizmet verebilir — bu yüzden sunucu kodu hız ve güvenlik açısından dikkatle yazılmalıdır.

Sunucu ne yanıt döner? YANIT

İstemci `GET /kullanicilar` der; sunucu veriyi hazırlayıp **200 OK** durum kodu ve JSON gövdesiyle yanıt verir. İstemci bu yanıtı alır ve ekranda gösterir. Konuşma tek bir tur halinde tamamlanır.

Alıştırma 8 dk

Modeli kavra:

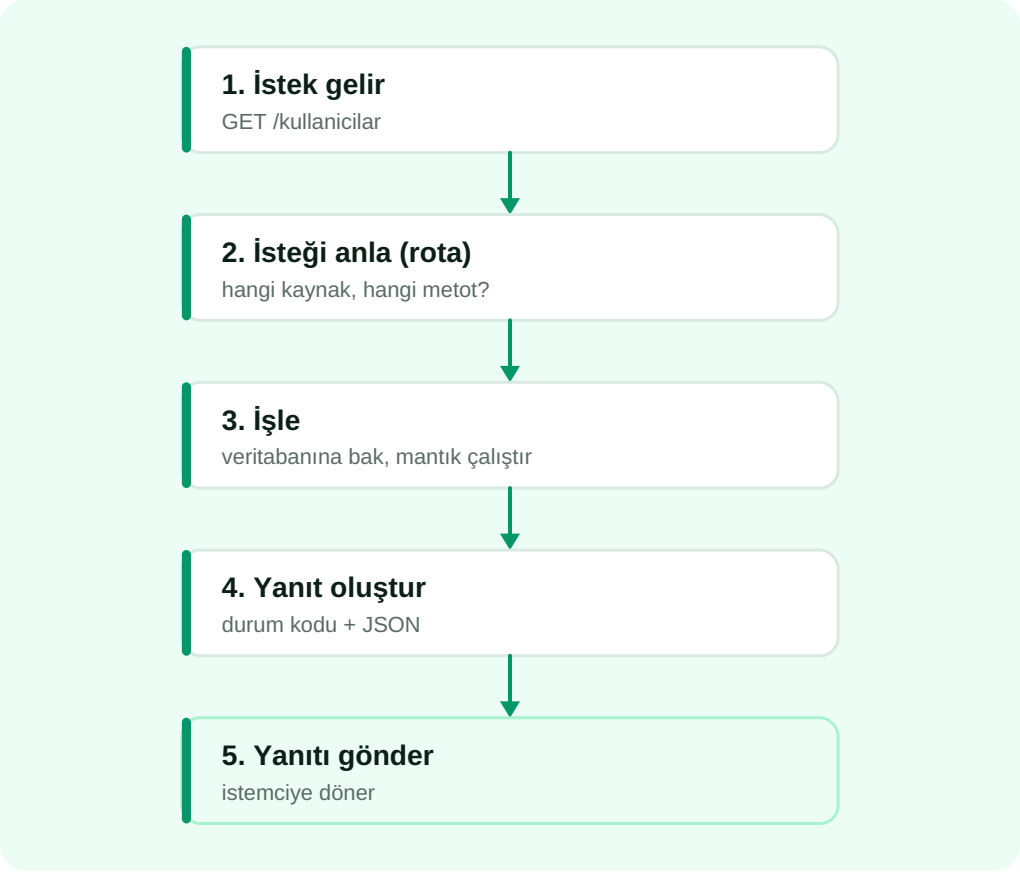
1. Günlük bir örnekle (lokanta?) istemci-sunucu ilişkisini anlat.
2. Bir web sayfası açtığında istemci ve sunucu kim, belirle.
3. İstek ve yanıtın neyi içerdiğini kısaca yaz.

BÖLÜM 03

Sunucu Nasıl Çalışır?

Bir sunucu, gelen istekleri dinleyen ve her birine yanıt üreten bir programdır. Her istek geldiğinde aynı döngüyü izler: al, anla, işle, yanıtla.

Sunucunun istek döngüsü



Şema 3.1 — Sunucu, her istek için bu beş adımı izler.

Bir rota işleyici (kavramsal)

```
 // "GET /kullanicilar" isteği gelince:
kullanicilar = veritabanindanGetir("kullanicilar")
yanıt.durum = 200
yanıt.govde = kullanicilar   // JSON olarak
yanıtı gönder
```

İPUCU

Sunucu "her zaman açık" bir programdır: çalışır, dinler ve istek geldikçe yanıtlar — günlerce, aylarca. Bu yüzden bir istekteki hata tüm sunucuyu çökertmemeli (hata yönetimi kritiktir, Seviye 3). Her isteği bağımsız ele almak (durumsuzluk), sunucuların ölçeklenmesini sağlar.

Sunucu ne yanıt döner? YANIT

İstek geldiğinde sunucu önce "bu hangi rota?" diye bakar (örn. `GET /kullanicilar`), uygun işleyiciyi çalıştırır, veritabanından veriyi alır ve bir yanıt (durum kodu + JSON) hazırlayıp gönderir. Sonra bir sonraki isteği bekler.

Alıştırma 10 dk

Döngüyü izle:

1. "GET /urunler" isteği geldiğinde sunucunun beş adımını sırayla yaz.
2. Hangi adımda veritabanı devreye girer?
3. Yanıtın iki temel parçasını (durum + gövde) belirt.

BÖLÜM 04

HTTP: Web'in Dili

HTTP (HyperText Transfer Protocol), istemci ile sunucunun konuştuğu ortak dildir. Her web isteği ve yanıtı, HTTP'nin belirlediği yapıda gönderilir. Bu yapıyı tanımak, arkayüzü anlamanın anahtarıdır.

Bir HTTP isteğinin yapısı

- **İstek satırı:** metot + yol (örn. `GET /api/kullanicilar`).
- **Başlıklar (headers):** ek bilgi (Host, Accept, Authorization...).
- **Gövde (body):** gönderilen veri (genelde POST/PUT'ta).

İSTEK (Request)

```
GET /api/kullanicilar HTTP/1.1
Host: ornek.com
Accept: application/json
Authorization: Bearer abc123...
```

Şema 4.1 — Bir HTTP isteğinin anatomisi.

İPUCU

HTTP **metinden ibarettir** ve okunabilir: bir istek aslında birkaç satır metindir. Başlıklar (headers) isteğe/yanıta dair "ek notlardır" — hangi formatta veri istendiği (`Accept`), kimlik bilgisi (`Authorization`), içeriğin türü (`Content-Type`) gibi. Tarayıcının geliştirici araçlarındaki "Network" sekmesinde gerçek HTTP isteklerini görebilirsin.

Sunucu ne yanıt döner? YANIT

İstemci bu HTTP isteğini gönderince sunucu, istek satırından "hangi kaynak, hangi metot" olduğunu, başlıklardan ise istemcinin ne istediğini (örn. JSON formatı) anlar ve buna göre bir yanıt hazırlar.

Alıştırma

10 dk

HTTP'yi incele:

1. Tarayıcının "Network" sekmesini açıp bir isteği incele.
2. İstek satırını, birkaç başlığı ve (varsa) gövdeyi bul.
3. Bir GET isteğinin neden genelde gövdesiz olduğunu açıkla.

BÖLÜM 05

İstek ve Yanıt (Request / Response)

Her HTTP alışverişi bir istek ve bir yanıttan oluşur. İsteği istemci hazırlar; yanıtı sunucu üretir. Yanıtın yapısı, isteğinkine benzer ama bir durum kodu içerir.

Bir HTTP yanıtının yapısı

- **Durum satırı:** protokol + durum kodu (örn. `200 OK`).
- **Başlıklar:** içerik türü, uzunluk, önbellek bilgisi...
- **Gövde:** asıl veri (genelde JSON).

YANIT (Response)

```
HTTP/1.1 200 OK
Content-Type: application/json

{
"ad": "Ayşe",
"yas": 30
}
```

Şema 5.1 — Bir HTTP yanıtının anatomisi: durum + başlıklar + JSON gövde.

İPUCU

Yanıtın en önemli parçası **durum kodudur**: istemci, gövdeyi okumadan önce isteğin başarılı olup olmadığını durum kodundan anlar (200 başarılı, 404 bulunamadı, 500 sunucu hatası). Doğru durum kodunu döndürmek, iyi bir API'nin işaretidir — istemci buna göre davranır.

Sunucu ne yanıt döner? YANIT

Sunucu, isteği başarıyla işlerse **200 OK** durumuyla ve istenen veriyi JSON gövdesinde döner. İstemci önce durum koduna bakar (başarılı mı?), sonra gövdedeki JSON'u ayrıştırıp kullanır.

◎ Alıştırma

10 dk

Yanıtı oku:

1. Bir 200 yanıtının üç parçasını (durum, başlık, gövde) işaretle.
2. Gövdedeki JSON'u kendi sözcüklerinle açıkla.
3. İstemcinin neden önce durum koduna baktığını yaz.

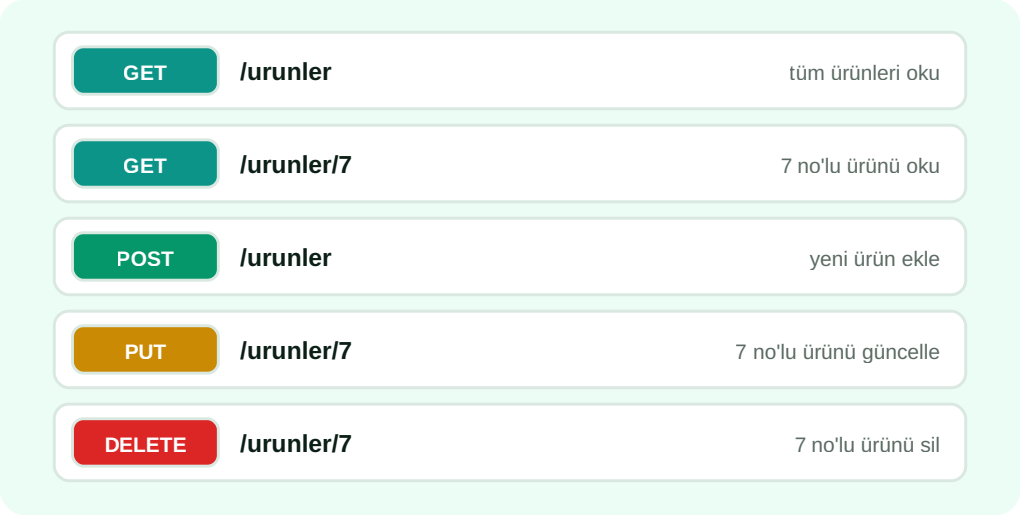
BÖLÜM 06

HTTP Metotları

HTTP metotları (fiilleri), bir kaynak üzerinde ne yapmak istediğini belirtir: okumak mı, eklemek mi, güncellemek mi, silmek mi? Doğru metodu kullanmak, anlaşılır ve standart bir API'nin temelidir.

Dört temel metot

- **GET:** veri oku (yan etkisiz, güvenli).
- **POST:** yeni veri oluştur.
- **PUT:** var olan veriyi güncelle.
- **DELETE:** veriyi sil.



Şema 6.1 — Aynı kaynak (/urunler), farklı metotlarla farklı işler.

İPUCU

Metotların "anlamına" uy: **GET asla veri değiştirmemeli** (sadece okur), silme işini DELETE yapmalı. Bir bağlantıya tıklamak veya sayfayı yenilemek GET tetikler; bu yüzden GET'in güvenli (yan etkisiz) olması önemlidir — yoksa yanlışlıkla veri silinebilir. Doğru metot, API'ni hem standart hem güvenli kılar.

Sunucu ne yanıt döner? YANIT

`GET /urunler` ürün listesini döner (200 + JSON); `POST /urunler` yeni ürün ekler ve genelde **201 Created** döner; `DELETE /urunler/7` siler ve **204 No Content** döner. Metot, sunucuya niyetini söyler.

Alıştırma

10 dk

Metot seç:

1. "Bir blog yazısı eklemek" için hangi metot? "Okumak" için?
2. "Bir yorumu silmek" ve "güncellemek" için metotları yaz.
3. GET'in neden veri değiştirmemesi gerektiğini açıkla.

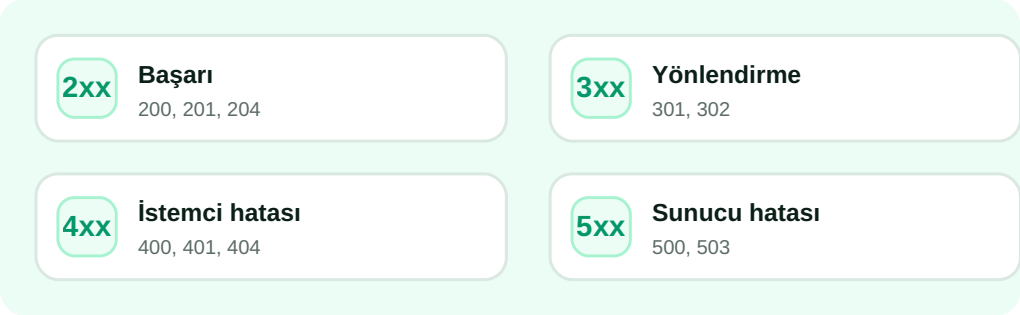
BÖLÜM 07

Durum Kodları (Status Codes)

Her HTTP yanıtı, isteğin sonucunu özetleyen üç haneli bir durum kodu içerir. Bu kodlar ailelere ayrılır; ailesini bilmek, sorunun nerede olduğunu anında söyler.

Durum kodu aileleri

- **2xx (Başarı):** 200 OK, 201 Created, 204 No Content.
- **3xx (Yönlendirme):** 301/302 başka adrese yönlendir.
- **4xx (İstemci hatası):** 400 hatalı istek, 401 yetkisiz, 404 bulunamadı.
- **5xx (Sunucu hatası):** 500 sunucu hatası, 503 hizmet yok.



Şema 7.1 — Durum kodu aileleri ve anlamları.

İPUCU

Kuralı hatırla: **4xx senin (istemcinin) hatan, 5xx sunucunun hatasıdır**. 404 "böyle bir kaynak yok" der; 401/403 "giriş/yetki gerekiyor" der; 500 ise "sunucuda bir şey patladı" demektir. Doğru kodu döndürmek, hata ayıklamayı kolaylaştırır: kullanıcıya 404 yerine 200 döndürüp boş veri göndermek, hataları gizler ve kafa karıştırır.

Sunucu ne yanıt döner? YANIT

İstemci her yanıtta önce durum koduna bakar: 2xx ise gövdeyi güvenle kullanır; 4xx ise isteğini düzeltmesi gerekir (örn. eksik bilgi); 5xx ise sunucu tarafında bir sorun vardır ve sonradan tekrar denemek gerekebilir.

◎ Alıştırma

8 dk

Kodları eşle:

1. "Olmayan bir sayfa" hangi kodu döner?
2. "Giriş yapmadan korumalı veriye erişmek" hangi kodu döner?
3. Bir 500 hatasının kimin sorumluluğunda olduğunu yaz.

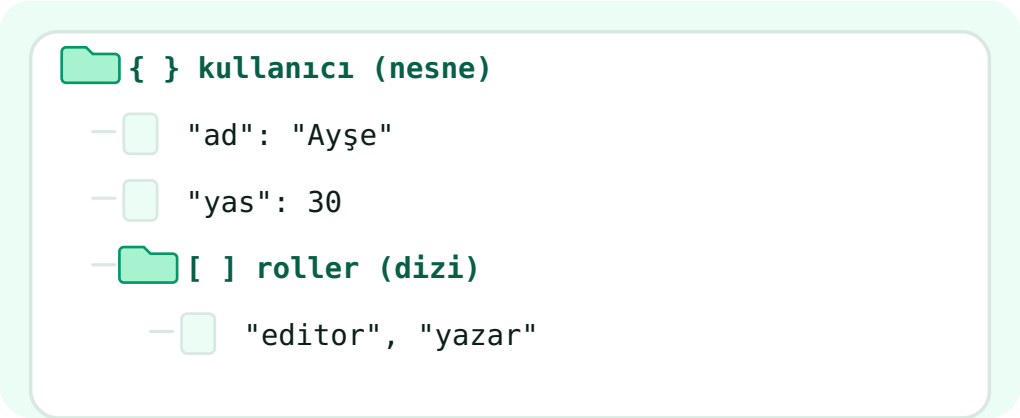
BÖLÜM 08

JSON: Veri Alışverişi

JSON (JavaScript Object Notation), web'de veri taşımanın standart biçimidir. Hem insanın okuyabileceği hem makinenin kolayca ayrıştırabileceği, sade bir metin formatıdır. Neredeyse tüm API'ler JSON konuşur.

JSON'un yapısı

- **Nesne:** süslü parantez `{ }`, anahtar–değer çiftleri.
- **Dizi:** köşeli parantez `[ ]`, sıralı liste.
- Değerler: metin, sayı, true/false, null, nesne, dizi.



Şema 8.1 — Bir JSON nesnesinin yapısı: anahtarlar, değerler, iç içe dizi.

JSON örneği

```
{
  "ad": "Ayşe",
  "yas": 30,
  "aktif": true,
  "roller": ["editor", "yazar"]
}
```

İPUCU

JSON, JavaScript nesnelerine çok benzer ama bir farkla: **anahtarlar çift tırnak içinde olmalıdır** (`"ad"`) ve sona virgül konmaz. En sık JSON hatası, fazladan virgül veya tek tırnak kullanmaktır. Sunucu ile istemci, veriyi her zaman JSON'a çevirip (serialize) gönderir, alınca geri ayrıştırır (parse).

Sunucu ne yanıt döner? YANIT

Sunucu veriyi JSON metnine çevirip yanıt gövdesinde gönderir; istemci bu metni alıp bir nesneye ayrıştırır (`JSON.parse`) ve `kullanici.ad` gibi alanlara erişir. JSON, iki taraf arasındaki ortak veri dilidir.

Alıştırma 10 dk

JSON yaz:

1. Kendini tanımlayan bir JSON nesnesi yaz (ad, yas, sehir, hobiler dizisi).
2. İç içe bir yapı kur (bir nesne içinde bir dizi).
3. Geçersiz JSON'da (fazladan virgül) hatayı bul.

SEVİYE 2

API'ler ve REST

Sistemlerin konuşma biçimi: API nedir, REST ilkeleri, uç noktalar ve rotalar, sorgu ve parametreler, bir API'yi tüketmek ve bir API tasarlamak.

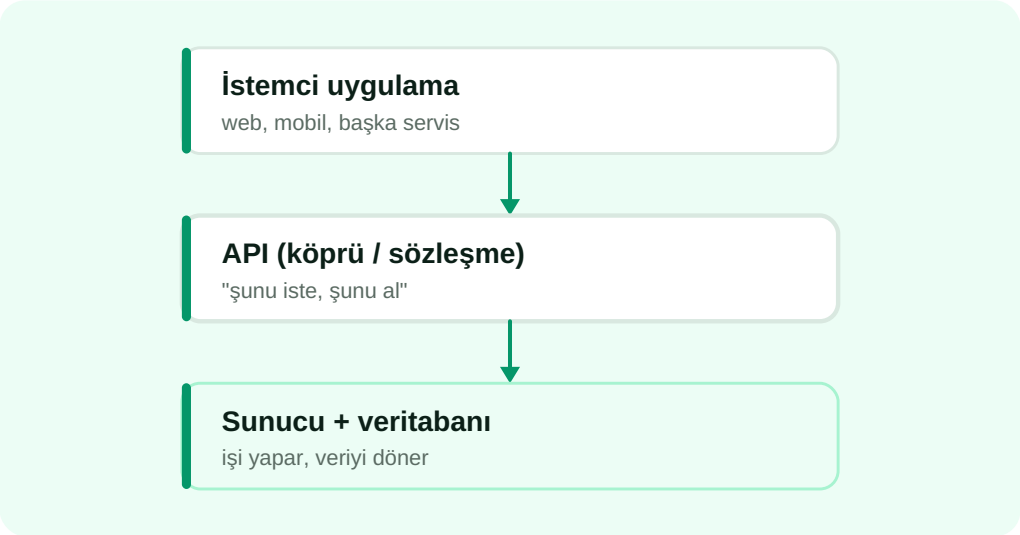
BÖLÜM 09

API Nedir?

API (Application Programming Interface), iki yazılımın birbiriyle konuşmasını sağlayan bir arayüzdür. Bir programın "şu işi yap" demesinin ve diğerinin yanıt vermesinin standart yoludur. Web'de API'ler genelde HTTP üzerinden çalışır.

API bir sözleşmedir

- Bir uygulamanın sunduğu "hizmet menüsüdür": neyi nasıl isteyebilirsin.
- İç ayrıntıları gizler; sadece "ne isteyip ne alacağını" bilirsin.
- Farklı sistemler (web, mobil) aynı API'yi kullanabilir.



Şema 9.1 — API, istemci ile sunucu arasındaki standart köprüdür.

İPUCU

Bir benzetme: API bir restoran menüsüdür. Mutfağın (sunucu) içini görmen gerekmez; menüden (API) sipariş verir (istek) ve yemeğini (yanıt) alırsın. Bu sayede bir hava durumu, harita veya ödeme servisini, iç işleyişini hiç bilmeden kendi uygulamanda kullanabilirsin — tek ihtiyacın API'nin nasıl çağrıldığını bilmek.

Sunucu ne yanıt döner? YANIT

İstemci API'ye "kullanıcı 5'in bilgisini ver" der; API bunu sunucuya iletir, sunucu veritabanından alır ve API üzerinden istemciye JSON döner. İstemci, sunucunun nasıl çalıştığını bilmeden sadece sözleşmeye (API'ye) uyar.

Alıştırma

8 dk

API'yi kavra:

1. Günlük hayattan bir "arayüz" örneği ver (priz? menü?).
2. Bir hava durumu uygulamasının neden bir API kullandığını açıkla.
3. API'nin "iç ayrıntıyı gizleme" özelliğini soyutlama ile ilişkilendir.

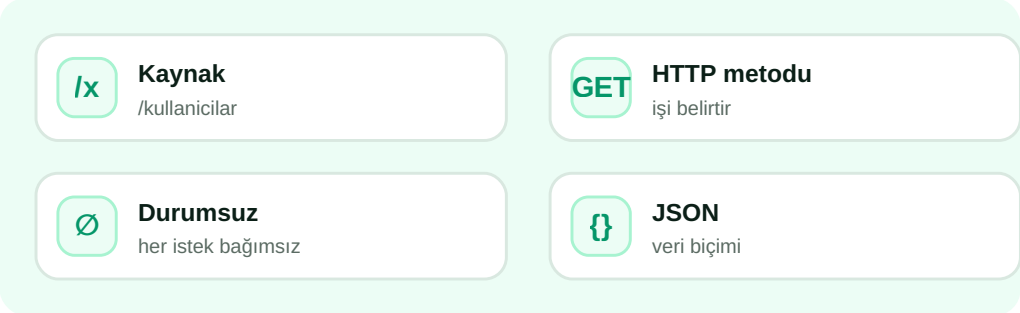
BÖLÜM 10

REST API İlkeleri

REST, web API'leri tasarlamanın en yaygın stilidir. Birkaç basit ilkeye dayanır ve bu ilkeler API'leri öngörülebilir, anlaşılır ve ölçeklenebilir kılar. Çoğu modern API RESTful'dur.

Temel REST ilkeleri

- **Kaynak odaklı:** her şey bir "kaynaktır" (kullanıcılar, ürünler) ve bir URL'si vardır.
- **HTTP metotları:** işlemi metot belirtir (GET/POST/PUT/DELETE).
- **Durumsuz (stateless):** her istek kendi kendine yeter; sunucu önceki isteği hatırlamaz.
- **JSON:** veri genelde JSON ile taşınır.



Şema 10.1 — REST'in dört temel ilkesi.

İPUCU

"Durumsuzluk" REST'in en önemli ve en çok yanlış anlaşılan ilkesidir: sunucu, istekler arasında istemciyi "hatırlamaz". Bu yüzden kimlik gibi bilgiler **her istekte** (örn. bir token başlığıyla) yeniden gönderilir. Durumsuzluk, sunucuların kolayca çoğaltılıp ölçeklenmesini sağlar — herhangi bir sunucu herhangi bir isteği karşılayabilir.

Sunucu ne yanıt döner? YANIT

RESTful bir API'de `GET /kullanicilar/5` her zaman "5 numaralı kullanıcıyı getir" anlamına gelir — öngörülebilirdir. Sunucu bu isteği, önceki isteklerden bağımsız olarak yanıtlar; ihtiyaç duyduğu kimlik bilgisini istekteki başlıktan okur.

◎ Alıştırma 10 dk

REST düşün:

1. Bir blog için kaynakları (yazılar, yorumlar) ve URL'lerini tasarla.
2. "Durumsuzluk" neden ölçeklenmeyi kolaylaştırır, açıkla.
3. Bir kaynağın dört temel işlemini (CRUD) metotlarla eşle.

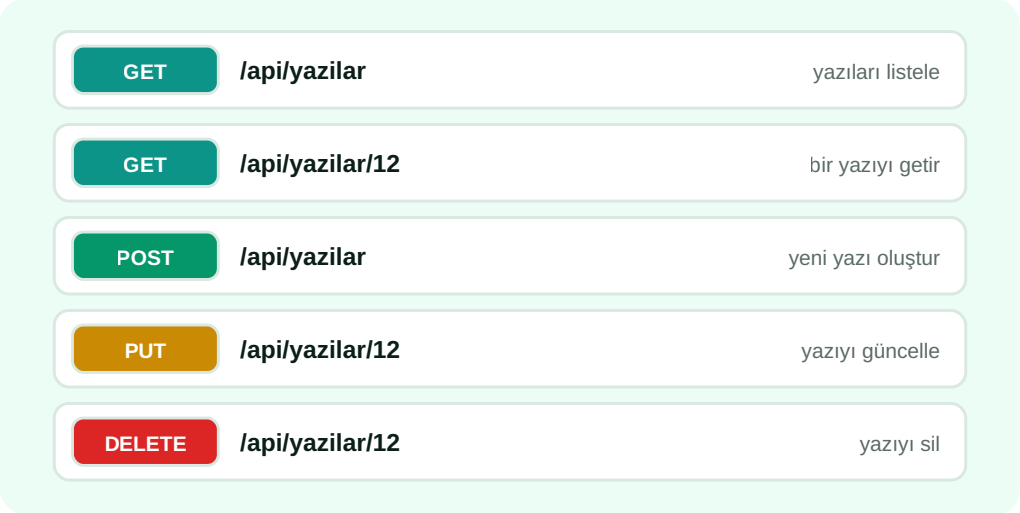
BÖLÜM 11

Uç Noktalar (Endpoints) ve Rotalar

Bir uç nokta (endpoint), API'nin belirli bir işlevine erişilen adrestir: metot + yol. Sunucu, gelen her isteği doğru işleyiciye yönlendirir; buna yönlendirme (routing) denir.

Uç nokta = metot + yol

- Her uç nokta bir **metot** ve bir **yol**tan oluşur.
- Sunucu, isteği eşleşen **işleyiciye (handler)** yönlendirir.
- Aynı yol, farklı metotlarla farklı işler yapar.



Şema 11.1 — Bir kaynağın (yazilar) uç noktaları.

Rota tanımı (kavramsal)

```
// Sunucu hangi isteği nereye yönlendireceğini bilir:
GET    /api/yazilar      -> yazilariListele()
POST   /api/yazilar      -> yaziOlustur()
GET    /api/yazilar/:id  -> yaziGetir(id)
DELETE /api/yazilar/:id  -> yaziSil(id)
```

İPUCU

İyi uç nokta isimleri **isimden (kaynaktan) oluşur, fiilden değil**: `GET /yazilar` doğru, `GET /yazilariGetir` yanlıştır — çünkü fiili (getir) zaten metot (GET) söyler. Çoğul isimler (`/yazilar`) ve `:id` gibi parametreler standarttır. Tutarlı isimlendirme, API'ni tahmin edilebilir kılar.

Sunucu ne yanıt döner? YANIT

İstemci `GET /api/yazilar/12` der; sunucu bu yolu "`/api/yazilar/:id → yaziGetir(id)`" kuralıyla eşler, `id=12` ile işleyiciyi çalıştırır ve 12 numaralı yazıyı JSON olarak döner.

Alıştırma 10 dk

Uç nokta tasarla:

1. Bir "yorumlar" kaynağı için beş uç nokta yaz (metot + yol).
2. URL'lerde fiil değil, isim kullandığından emin ol.
3. `:id` parametresinin ne işe yaradığını açıkla.

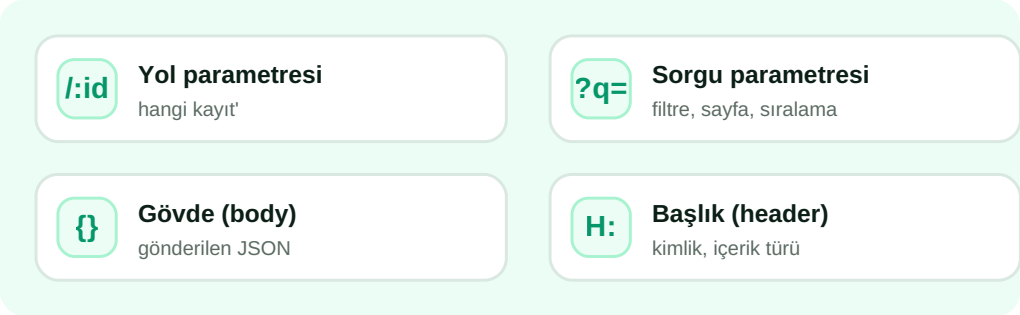
BÖLÜM 12

Sorgu ve Parametreler

İstemci, isteğine ek bilgi eklemek için parametreler kullanır: hangi kaydı istediğini, nasıl filtreleneceğini veya gönderdiği veriyi. Bu bilgilerin yol, sorgu ve gövde olmak üzere farklı yerleri vardır.

Bilgiyi nereye koyarsın?

- **Yol parametresi:** hangi kayıt — `/urunler/7`.
- **Sorgu parametresi:** filtre/sıralama — `/urunler?kategori=kitap&sayfa=2`.
- **Gövde (body):** gönderilen veri — POST/PUT'ta JSON.
- **Başlık (header):** meta bilgi — kimlik, içerik türü.



Şema 12.1 — İstekte bilginin taşındığı dört yer.

Farklı parametre yerleri

```
GET /urunler/7                    // yol: 7 no'lu ürün
GET /urunler?kategori=kitap       // sorgu: filtre
POST /urunler                     // gövde: yeni ürün JSON'u
   { "ad": "Kalem", "fiyat": 15 }
```

İPUCU

Doğru yeri seç: **belirli bir kaydı** işaret ediyorsan yol parametresi (`/urunler/7`); **listeyi filtreliyor/sıralıyorsan** sorgu parametresi (`?kategori=...`); **yeni/güncel veri gönderiyorsan** gövde. Hassas verileri (şifre vb.) asla URL'ye (yol/sorgu) koyma — URL'ler loglanır ve geçmişte saklanır; onlar gövdede ve HTTPS üzerinden gider.

Sunucu ne yanıt döner? YANIT

`GET /urunler?kategori=kitap&sayfa=2` isteğinde sunucu, sorgu parametrelerini okuyup "kitap kategorisindeki ürünlerin 2. sayfasını" döner. Yol "hangi kaynak", sorgu ise "o kaynağı nasıl süzeceğini" söyler.

Alıştırma

10 dk

Parametre yerleştir:

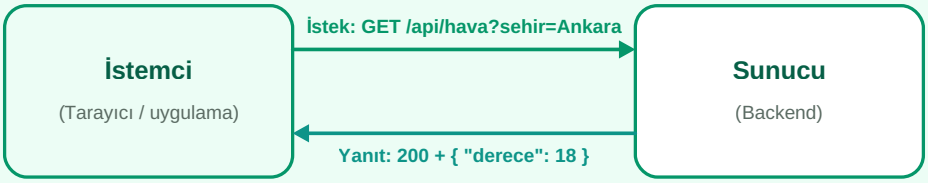
1. "42 numaralı kullanıcıyı getir" isteğini yol parametresiyle yaz.
2. "Aktif kullanıcıları isme göre sırala" isteğini sorgu parametresiyle yaz.
3. Bir şifrenin neden URL'de değil gövdede gitmesi gerektiğini açıkla.

BÖLÜM 13

Bir API'yi Tüketmek

API'leri yalnızca sunmazsın; başkalarının API'lerini de kullanırsın (tüketirsin). Önyüzden veya başka bir servisten bir API'ye istek gönderip yanıtı işlemek, modern geliştirmenin günlük işidir.

Bir API çağrısının akışı



Şema 13.1 — Bir API'yi tüketmek: istek gönder, JSON yanıtı kullan.

fetch ile API tüketmek (JavaScript)

```
async function havaGetir(sehir) {
  const yanit = await fetch("/api/hava?sehir=" + sehir);
  if (!yanit.ok) throw new Error("İstek başarısız: " + yanit.status);
  const veri = await yanit.json();
  return veri.derece;
}
```

İPUCU

Bir API'yi tüketirken üç şeyi her zaman ele al: **(1)** durum kodunu kontrol et (`yanit.ok`), **(2)** JSON'u ayrıştır, **(3)** hataları yakala (ağ kopabilir, API çökebilir). Dış bir API'ye **asla körü körüne güvenme**: gelen veriyi doğrula, beklediğin alanların var olduğundan emin ol. Modül 5'teki `fetch` ve `try/catch` bilgilerin burada doğrudan işe yarar.

Sunucu ne yanıt döner? YANIT

`havaGetir("Ankara")` çağrısı bir GET isteği gönderir; sunucu 200 koduyla `{ "derece": 18 }` döner; fonksiyon JSON'u ayrıştırıp **18** değerini döndürür. İstek başarısız olursa (örn. 500) hata fırlatılır ve çağıran taraf bunu ele alır.

Alıştırma

12 dk

API tüket:

1. Bir API'ye `fetch` ile GET isteği gönderen kod yaz.
2. Durum kodunu kontrol et ve hatayı ele al.
3. Gelen JSON'dan tek bir alanı çıkarıp kullan.

BÖLÜM 14

Bir API Tasarlamak

İyi bir API tasarlamak, kaynaklarını belirlemek ve her biri için tutarlı uç noktalar oluşturmaktır. İyi tasarlanmış bir API'yi kullanmak kolaydır; kötü tasarlanmış olanı ise her adımda şaşırtır.

Tasarım adımları

- **Kaynakları belirle:** sistemdeki "şeyler" (kullanıcı, sipariş...).
- **Uç noktaları tanımla:** her kaynak için CRUD.
- **Tutarlı ol:** isimlendirme, durum kodları, yanıt biçimi.

Şema 14.1 — Bir "siparişler" kaynağı için tutarlı API tasarımı.

İPUCU

Tutarlılık, iyi API tasarımının kalbidir: tüm kaynaklar aynı kalıbı izlerse (çoğul isimler, aynı CRUD yapısı, aynı hata biçimi), geliştiriciler bir kaynağı öğrenince diğerlerini tahmin edebilir. Yanıtların biçimini de standartlaştır: başarıda hangi yapı, hatada hangi yapı dönecek? Önceden karar ver ve her yerde aynısını kullan.

Sunucu ne yanıt döner? YANIT

İyi tasarlanmış bir API'de `POST /siparisler` yeni sipariş oluşturup **201 Created** döner; bulunamayan bir kaynak **404** döner; silme **204 No Content** döner. Bu tutarlılık sayesinde istemci, her uç noktanın nasıl davranacağını önceden bilir.

◎ Alıştırma

12 dk

API tasarla:

1. Bir "kütüphane" sistemi için kaynakları belirle (kitaplar, üyeler).
2. "kitaplar" kaynağı için beş CRUD uç noktası tasarla.
3. Her uç noktanın döneceği durum kodunu belirt.

SEVİYE 3

Veri ve Güvenlik

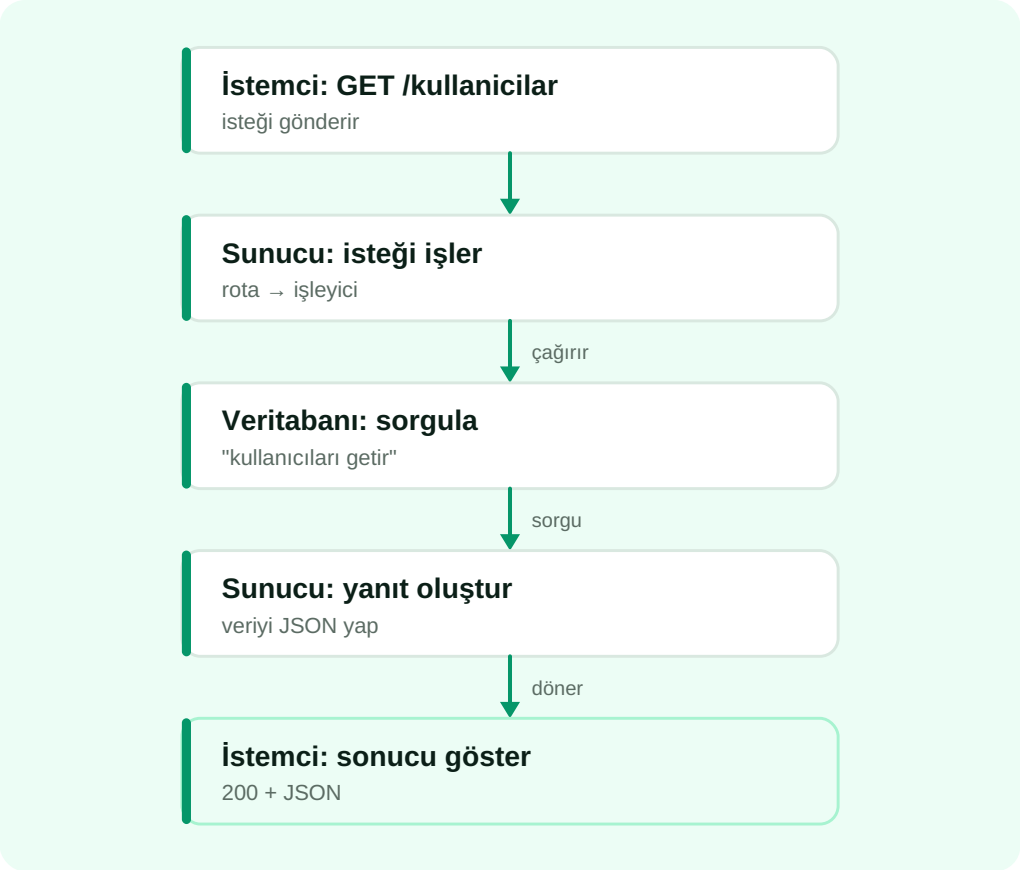
Arkayüzün ciddi işleri: sunucu-veritabanı ilişkisi, CRUD işlemleri, kimlik doğrulama, yetkilendirme, güvenlik temelleri ve hata yönetimi.

BÖLÜM 15

Sunucu ve Veritabanı

Sunucunun belleği geçicidir; kalıcı veriyi bir veritabanı tutar. Arkayüzün en önemli işlerinden biri, istekleri veritabanı işlemlerine çevirmek ve sonucu istemciye döndürmektir.

Veri nasıl akar?



Şema 15.1 — İstemci → sunucu → veritabanı → sunucu → istemci akışı.

İPUCU

Veritabanı, uygulamanın "kalıcı hafızasıdır": sunucu yeniden başlasa bile veri kaybolmaz. Sunucu ile veritabanı arasındaki konuşma genelde **SQL** (sonraki modülün konusu) ile yapılır. Önemli bir güvenlik kuralı: kullanıcıdan gelen veriyi **asla doğrudan** bir veritabanı sorgusuna gömme — bu, "SQL enjeksiyonu" denen ciddi bir açığa yol açar (Bölüm 19).

Sunucu ne yanıt döner? YANIT

İstemci kullanıcı listesini ister; sunucu veritabanına "kullanıcıları getir" sorgusu gönderir, dönen satırları JSON'a çevirir ve istemciye 200 ile döner. Veritabanı verinin kalıcı kaynağı, sunucu ise çevirmen ve bekçidir.

Alıştırma 10 dk

Akışı izle:

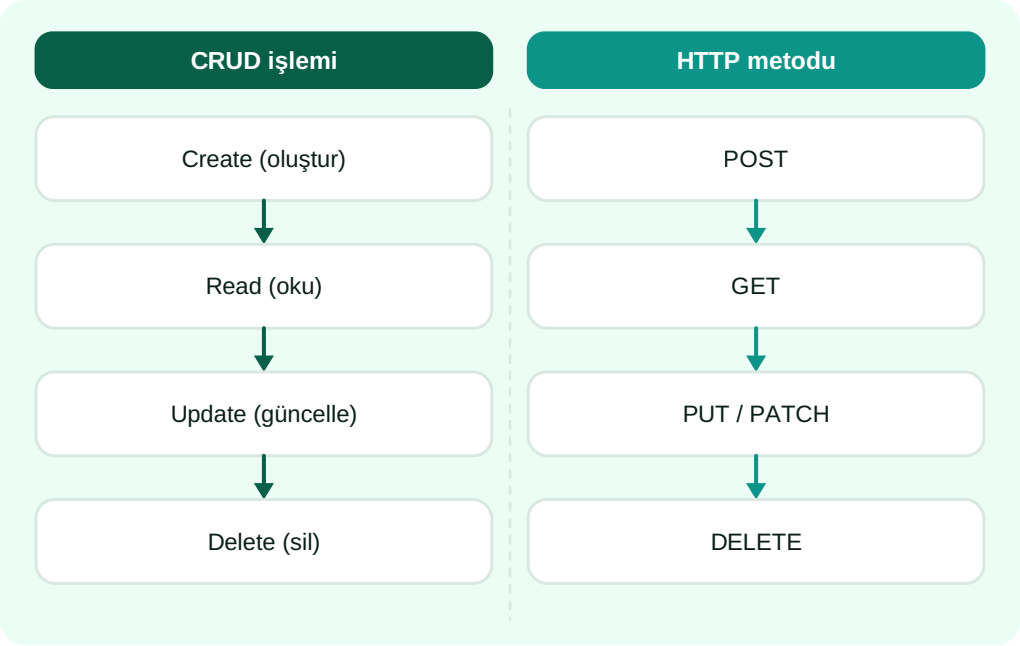
1. "Yeni bir sipariş oluştur" isteğinin veritabanına kadar olan yolunu yaz.
2. Veritabanının neden gerekli olduğunu (sunucu belleği yerine) açıkla.
3. Kullanıcı verisinin neden doğrudan sorguya gömülmemesi gerektiğini not et.

BÖLÜM 16

CRUD İşlemleri

Çoğu uygulamanın özünde dört işlem vardır: Oluştur, Oku, Güncelle, Sil (Create, Read, Update, Delete — CRUD). REST API'lerde bu dört işlem, dört HTTP metoduna düzenli biçimde karşılık gelir.

CRUD ve HTTP eşleşmesi



Şema 16.1 — Dört CRUD işlemi, dört HTTP metoduna karşılık gelir.

CRUD uç noktaları (kavramsal)

```
 POST   /gorevler        // Create: yeni görev
GET    /gorevler        // Read:   tümünü oku
GET    /gorevler/:id    // Read:   birini oku
PUT    /gorevler/:id    // Update: güncelle
DELETE /gorevler/:id    // Delete: sil
```

İPUCU

CRUD, neredeyse her veri odaklı uygulamanın iskeletidir: bir blog (yazılar), bir mağaza (ürünler), bir görev uygulaması (görevler) — hepsi CRUD'dur. Bu kalıbı bir kez öğrenince, çoğu yeni API "tanıdık" gelir. `PUT` tüm kaydı değiştirir, `PATCH` ise sadece bazı alanları günceller — ikisi arasındaki farkı bilmek işine yarar.

Sunucu ne yanıt döner? YANIT

Bir görev uygulamasında: `POST /gorevler` yeni görev ekler (201), `GET /gorevler` hepsini listeler (200), `PUT /gorevler/3` 3 numaralıyı günceller (200), `DELETE /gorevler/3` siler (204). Dört işlem, dört metot — düzenli ve tahmin edilebilir.

Alıştırma 10 dk

CRUD eşle:

1. Bir "not defteri" uygulaması için CRUD uç noktalarını yaz.
2. Her CRUD işlemini doğru HTTP metoduyla eşle.
3. PUT ile PATCH arasındaki farkı bir örnekle açıkla.

BÖLÜM 17

Kimlik Doğrulama (Authentication)

Kimlik doğrulama "Sen kimsin?" sorusunu yanıtlar. Kullanıcı giriş yapar, sunucu kimliğini doğrular ve ona bir "anahtar" (token) verir; sonraki isteklerde bu anahtarı göstererek kim olduğunu kanıtlar.

Giriş ve token akışı

Şema 17.1 — Kimlik doğrulama: giriş → token → sonraki isteklerde kanıt.

Token ile istek (kavramsal)

```
// Girişten sonra her korumalı istekte:
GET /api/profil
Authorization: Bearer eyJhbGci...   // kimlik anahtarı
```

İPUCU

İki altın kural: **(1)** şifreleri asla düz metin saklama — her zaman güçlü bir algoritmayla (bcrypt gibi) **hash'le**; veritabanı çalınsa bile şifreler okunamasın. **(2)** Token'lar ve şifreler yalnızca **HTTPS** üzerinden gitmeli. Token, "durumsuz" REST'te kimliği her istekte taşımanın yoludur — sunucu seni hatırlamaz, sen her seferinde anahtarını gösterirsin.

Sunucu ne yanıt döner? YANIT

Kullanıcı doğru şifreyle giriş yapınca sunucu imzalı bir token döner. İstemci bunu saklar ve sonraki her korumalı istekte `Authorization` başlığında gönderir. Sunucu token'ı doğrular ve "evet, bu Ayşe" diyerek isteği işler. Yanlış/eksik token → **401 Unauthorized**.

Alıştırma 12 dk

Kimliği doğrula:

1. Giriş → token → korumalı istek akışını adım adım yaz.
2. Şifrelerin neden hash'lenerek saklandığını açıkla.
3. Eksik token durumunda hangi durum kodu dönmeli?

BÖLÜM 18

Yetkilendirme (Authorization)

Kimlik doğrulama "kim olduğunu", yetkilendirme ise "ne yapabileceğini" belirler. Bir kullanıcı giriş yapmış olabilir (kimliği doğru) ama yine de bir işlemi yapmaya izni olmayabilir.

Kimlik ≠ yetki

- **Kimlik doğrulama:** "Sen Ayşe'sin." (kim?)
- **Yetkilendirme:** "Ayşe bu işi yapabilir mi?" (ne?)
- Roller/izinler ile yönetilir (admin, editör, kullanıcı).

Şema 18.1 — Yetkilendirme: kimlik doğru olsa bile izin gerekir.

İPUCU

Yetki kontrolü **her zaman sunucuda** yapılmalı — asla önyüze güvenme. Bir düğmeyi tarayıcıda gizlemek "güvenlik" değildir; kullanıcı isteği elle de gönderebilir. "En az ayrıcalık" ilkesini uygula: her kullanıcıya yalnızca ihtiyacı olan izinleri ver. Yetkisiz erişimde **403 Forbidden** döndür (401 "giriş yap", 403 "girdin ama iznin yok" demektir).

Sunucu ne yanıt döner? YANIT

Bir editör `DELETE /kullanicilar/9` denerse: token geçerlidir (kimlik tamam) ama silme yalnızca admin'e açıksa sunucu **403 Forbidden** döner. Aynı isteği admin yaparsa işlem gerçekleşir. Yetki, kimlikten ayrı bir kapıdır.

Alıştırma 10 dk

Yetki tasarla:

1. Bir blog için roller (okuyucu, yazar, admin) ve izinlerini listele.
2. "Bir yazıyı silme" için hangi rol yetkili olmalı?
3. 401 ile 403 arasındaki farkı bir örnekle açıkla.

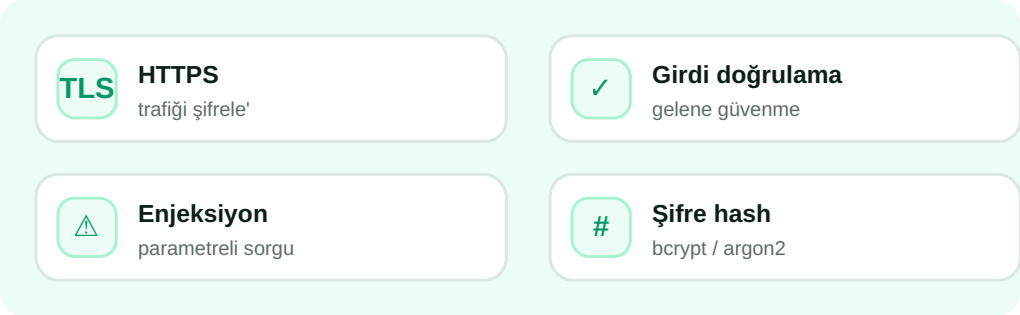
BÖLÜM 19

Güvenlik Temelleri

Arkayüz, sistemin güvenlik kalesidir. Birkaç temel önlem, en yaygın saldırıların çoğunu engeller. Güvenlik sonradan eklenen bir özellik değil, baştan benimsenen bir alışkanlık olmalıdır.

Olmazsa olmaz önlemler

- **HTTPS:** tüm trafiği şifrele (asla düz HTTP değil).
- **Girdi doğrulama:** gelen her veriyi denetle, asla güvenme.
- **Enjeksiyona karşı:** sorgularda parametre kullan, veri gömme.
- **Şifre hash'leme:** şifreleri bcrypt gibi yöntemlerle sakla.



Şema 19.1 — Arkayüz güvenliğinin dört temel taşı.

İPUCU

En önemli güvenlik ilkesi: **kullanıcıdan gelen hiçbir veriye güvenme**. Her girdiyi doğrula (tür, uzunluk, biçim), veritabanı sorgularında **asla** kullanıcı verisini metin olarak birleştirme — bunun yerine "parametreli sorgu" kullan (Veritabanı modülünde göreceksin). Bu tek alışkanlık, en tehlikeli açıklardan biri olan enjeksiyon saldırılarını engeller. Güvenlik, hatadan sonra değil, baştan tasarlanır.

Sunucu ne yanıt döner? YANIT

Bir saldırgan bir form alanına zararlı kod yazmayı denerse: girdi doğrulama bunu reddeder, parametreli sorgu onu "veri" olarak görüp çalıştırmaz ve HTTPS trafiği dinlenmekten korur. Bu katmanlı savunma, sistemini gerçek dünya saldırılarına karşı ayakta tutar.

Alıştırma

12 dk

Güvenliği uygula:

1. Bir kayıt formuna gelen veride hangi doğrulamaları yapardın, listele.
2. HTTPS'in neden zorunlu olduğunu açıkla.
3. Şifrelerin düz metin yerine neden hash'lendiğini yaz.

BÖLÜM 20

Hata Yönetimi ve Loglama

Gerçek sistemlerde hatalar kaçınılmazdır: ağ kopar, veritabanı yanıt vermez, kullanıcı hatalı veri gönderir. İyi bir arkayüz, hataları nazikçe ele alır, anlamlı yanıtlar döner ve sorunları izlemek için loglar.

Hatayı doğru ele almak

Şema 20.1 — Hata yönetimi: yakala, logla, anlamlı yanıt dön.

Hata ele alma (kavramsal)

```
try {
  kullanici = veritabanindanGetir(id)
  EĞER kullanici yok İSE  yanıt(404, "Kullanıcı bulunamadı")
  DEĞİLSE  yanıt(200, kullanici)
} catch (hata) {
  logla(hata)               // ayrıntı sunucuda kalır
  yanıt(500, "Bir hata oluştu")   // kullanıcıya genel mesaj
}
```

İPUCU

İki ilke: **(1)** hata ayrıntılarını (yığın izi, SQL hatası) asla kullanıcıya gösterme — bu, saldırganlara ipucu verir; ayrıntıyı **logla**, kullanıcıya genel bir mesaj dön. **(2)** Doğru durum kodunu kullan: bulunamadı 404, hatalı girdi 400, sunucu hatası 500. İyi loglama, üretimde "neyin neden bozulduğunu" anlamanın tek yoludur — sessizce yutulan hatalar en tehlikelisidir.

Sunucu ne yanıt döner?

YANIT

Veritabanı yanıt vermezse: sunucu hatayı yakalar, ayrıntıyı kendi loglarına yazar ve kullanıcıya **500** ile "Bir hata oluştu" gibi genel bir mesaj döner — çökmeden, ipucu sızdırmadan. İstenen kayıt yoksa **404**, eksik bilgi varsa **400** döner.

Alıştırma

10 dk

Hatayı yönet:

1. "Olmayan bir kullanıcıyı getirme" için hangi kod ve mesaj dönmeli?
2. Hata ayrıntısının neden kullanıcıya değil loga gitmesi gerektiğini açıkla.
3. Bir try/catch ile bir veritabanı çağrısını sözde kodla sar.

SEVİYE 4

Mimari ve Üretim

Gerçek sistemler: ortam değişkenleri, önbellekleme, asenkron işler ve kuyruklar, mimari desenler, API belgeleme-test ve küçük bir API tasarlamak.

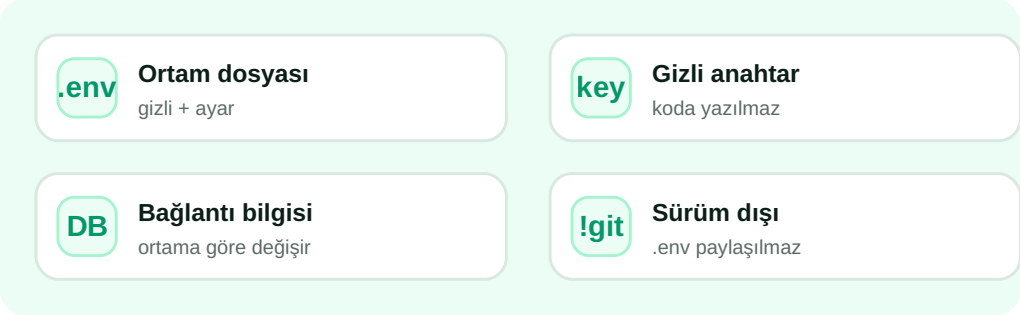
BÖLÜM 21

Ortam Değişkenleri ve Yapılandırma

Bir uygulama farklı ortamlarda çalışır: geliştirme, test, üretim. Veritabanı adresleri, gizli anahtarlar ve ayarlar koda gömülmemeli; bunun yerine ortam değişkenlerinde tutulmalıdır.

Yapılandırmayı koddan ayır

- **Gizli anahtarlar** (API anahtarı, şifre) asla kaynak koda yazılmaz.
- Ortam değişkenleri (`.env`) ile dışarıdan verilir.
- Farklı ortamlar (geliştirme/üretim) farklı değerler kullanır.



Şema 21.1 — Yapılandırmayı koddan ayırmak.

İPUCU

Gizli anahtarları asla kod deposuna (Git) gönderme — bu, en sık yapılan ve en pahalı güvenlik hatalarından biridir. `.env` dosyasını `.gitignore`'a ekle. Kuralı hatırla: kod herkese açık olabilir, sırlar olmamalı. Aynı kod, ortam değişkenleri sayesinde geliştirme makinende test veritabanına, üretimde gerçek veritabanına bağlanır — kodu değiştirmeden.

Sunucu ne yanıt döner? YANIT

Sunucu başlarken yapılandırmayı ortam değişkenlerinden okur: `VERITABANI_URL`, `API_ANAHTARI` gibi. Bu sayede aynı kod, farklı ortamlarda farklı (ve gizli) değerlerle çalışır; sırlar kaynak kodda görünmez.

Alıştırma

8 dk

Yapılandır:

1. Bir uygulamanın hangi değerlerini ortam değişkeninde tutardın, listele.
2. `.env` dosyasının neden Git'e gönderilmemesi gerektiğini açıkla.
3. Aynı kodun farklı ortamlarda nasıl çalıştığını anlat.

BÖLÜM 22

Önbellekleme (Caching)

Bazı veriler sık istenir ama nadiren değişir. Her seferinde sıfırdan hesaplamak veya veritabanına gitmek yerine, sonucu geçici olarak saklamak (önbellek) sistemi belirgin biçimde hızlandırır.

Önbellek mantığı

Şema 22.1 — Önbellek: varsa hızlı dön, yoksa hesapla ve sakla.

İPUCU

Önbellek hızı artırır ama bir bedeli vardır: **eskime (stale) sorunu**. Veri değişince önbellek güncellenmezse, kullanıcılara eski veri gösterilir. Bu yüzden önbelleğe bir "son kullanma süresi" verilir veya veri değişince önbellek temizlenir. Kural: önce doğru çalışsın, sonra (gerçekten gerekiyorsa) önbellekle. Erken önbellekleme, çözmediğin bir sorunu karmaşıklaştırır.

Sunucu ne yanıt döner? YANIT

Bir ürün listesi sık istenip nadiren değişiyorsa: ilk istekte veritabanından alınıp önbelleğe konur; sonraki istekler önbellekten anında yanıtlanır (veritabanına gitmeden). Ürün güncellenince önbellek temizlenir, böylece bir sonraki istek taze veriyi alır.

Alıştırma 10 dk

Önbelleği düşün:

1. Hangi tür verilerin önbelleğe uygun olduğunu (sık okunan, az değişen) örnekle.
2. Önbelleğin "eskime" sorununu bir örnekle açıkla.
3. Bir veri güncellenince önbellekle ne yapılmalı?

BÖLÜM 23

Asenkron İşler ve Kuyruklar

Bazı işler uzun sürer: e-posta gönderme, video işleme, rapor üretme. Kullanıcıyı bekletmemek için bu işler "arka plana" atılır: bir kuyruğa konur ve sırayla, ayrı çalışanlar tarafından işlenir.

Arka plan iş akışı

Şema 23.1 — Uzun işler kuyruğa atılır, arka planda işlenir.

İPUCU

Temel fikir: kullanıcının yanıtını **uzun bir işe bağlama**. İstek geldiğinde işi kuyruğa koy ve hemen "isteğin alındı" (örn. 202 Accepted) dön; ağır işi arka plandaki çalışanlar yapsın. Bu, hem kullanıcı deneyimini (bekletme yok) hem de güvenilirliği artırır (iş başarısız olursa kuyruk tekrar deneyebilir). Kuyruklar, Modül 6'daki FIFO mantığının gerçek dünyadaki en yaygın uygulamasıdır.

Sunucu ne yanıt döner? YANIT

Kullanıcı "raporu oluştur" der; sunucu işi kuyruğa atıp anında "rapor hazırlanıyor" yanıtı döner. Arka plandaki bir çalışan raporu üretir ve bitince kullanıcıya bildirir. Kullanıcı, dakikalarca süren bir işlemi ekran başında beklemez.

◎ Alıştırma

10 dk

Arka plana taşı:

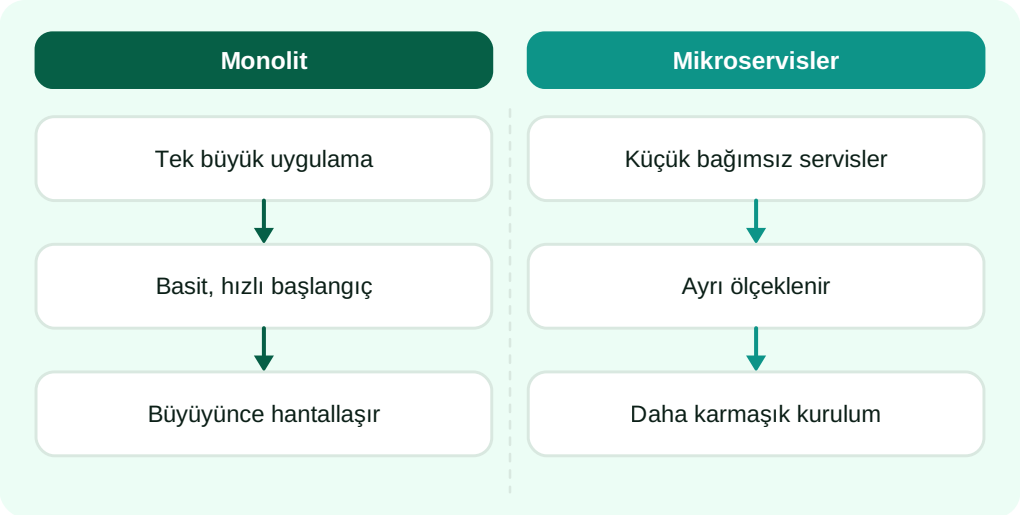
1. Hangi işlerin kuyruğa uygun olduğunu (uzun, hemen sonuç gerekmeyen) örnekle.
2. Kuyruğun hangi veri yapısına (Modül 6) dayandığını söyle.
3. İşi kuyruğa atmanın kullanıcı deneyimine faydasını açıkla.

BÖLÜM 24

Mimari Desenler

Bir uygulamayı nasıl yapılandıracağın, büyüdükçe önem kazanır. İki yaygın yaklaşım vardır: her şeyi tek bir parçada tutan monolit ve sistemi küçük bağımsız servislere bölen mikroservis mimarisi.

Monolit ve servisler



Şema 24.1 — Monolit ile mikroservis: iki farklı yapılandırma yaklaşımı.

İPUCU

Yaygın bir tuzak: küçük bir proje için mikroservislerle başlamak. Çoğu durumda **iyi yapılandırılmış bir monolit** ile başlamak daha doğrudur — basit, hızlı ve anlaşılır. Sistem gerçekten büyüdüğünde ve farklı parçalar ayrı ölçeklenmeyi gerektirdiğinde servislere geçmeyi düşünürsün. "Önce çalışan basit çözüm, gerçek ihtiyaç doğunca karmaşıklık" ilkesi burada da geçerli. Çoğu uygulamanın iç düzeni ise **katmanlıdır**: rota → iş mantığı → veri erişimi.

Sunucu ne yanıt döner? YANIT

Yeni başlayan bir proje genelde tek bir monolit olarak kurulur — geliştirmesi ve dağıtması en kolayıdır. Trafik ve ekip büyüdüğünde, ödeme veya bildirim gibi parçalar bağımsız servislere ayrılabilir; böylece her servis kendi ihtiyacına göre ölçeklenir. Mimari, soruna göre seçilir; her zaman "en yeni" en iyi değildir.

◎ Alıştırma

10 dk

Mimariyi değerlendir:

1. Küçük bir proje için monolit mi mikroservis mi? Gerekçeni yaz.
2. Mikroservislerin bir avantajını ve bir dezavantajını söyle.
3. Katmanlı yapının (rota → mantık → veri) faydasını açıkla.

BÖLÜM 25

API Belgeleme ve Test

Bir API ne kadar iyi olursa olsun, nasıl kullanılacağı belgelenmemişse işe yaramaz. İyi belgeleme ve düzenli test, bir API'yi hem kullanılabilir hem güvenilir kılar.

İyi bir API'nin iki desteği

- **Belgeleme:** her uç nokta, parametreleri ve örnek istek/yanıt.
- **Test:** uç noktaların beklendiği gibi çalıştığını otomatik doğrula.
- **Sürümleme:** değişiklikler eski istemcileri bozmasın (`/v1/`).

Şema 25.1 — Belgeleme, test ve sürümleme: sürdürülebilir API.

İPUCU

İyi belgeleme her uç nokta için "ne gönder, ne alırsın"ı örneklerle gösterir — geliştiriciler kodu okumadan API'ni kullanabilmeli. Otomatik testler ise bir değişikliğin mevcut davranışı bozmadığını garanti eder (özellikle çok kişi çalışırken). API'ni değiştirip eski istemcileri bozmamak için **sürümleme** kullan: `/api/v1/` dururken yeni davranışı `/api/v2/` altında sun.

Sunucu ne yanıt döner? YANIT

İyi belgelenmiş bir API'de her uç nokta için örnek bir istek ve örnek bir yanıt bulunur; geliştirici bunu kopyalayıp deneyebilir. Otomatik testler, "GET /kullanicilar 200 ve bir dizi dönmeli" gibi beklentileri her değişiklikte yeniden doğrular — böylece sürprizler erken yakalanır.

Alıştırma

10 dk

Belgele ve test et:

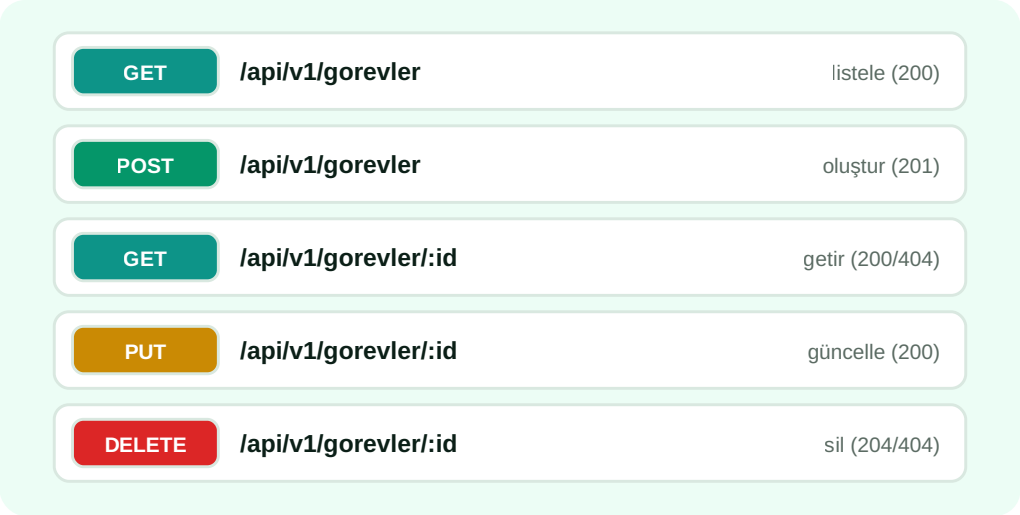
1. Bir uç nokta için örnek istek ve yanıt yaz (belgeleme gibi).
2. O uç nokta için hangi davranışı test ederdin?
3. Sürümlemenin (`/v1/`) neden gerekli olduğunu açıkla.

BÖLÜM 26

Bitirme: Küçük Bir API Tasarlamak

Tüm öğrendiklerini birleştirip baştan sona küçük bir REST API tasarlıyorsun: kaynaklar, uç noktalar, durum kodları, güvenlik ve hata yönetimi bir arada. Bu, gerçek bir arkayüz projesinin iskeletidir.

Örnek: bir "görevler" API'si



Şema 26.1 — Baştan sona tasarlanmış küçük bir REST API.

Tasarım kontrol listesi

```
// 1. Kaynak: gorevler  (çoğul isim)
// 2. CRUD uç noktaları + doğru durum kodları
// 3. Güvenlik: HTTPS + token + yetki kontrolü
// 4. Girdi doğrulama: her POST/PUT gövdesini denetle
// 5. Hata yönetimi: 400/404/500 + loglama
// 6. Sürümleme: /api/v1/
```

İPUCU

Gerçek bir API tasarlarken bu kontrol listesini izle: kaynakları belirle, CRUD uç noktalarını tutarlı kur, doğru durum kodlarını dön, güvenliği (HTTPS, kimlik, yetki, doğrulama) baştan ekle, hataları nazikçe yönet ve belgele. Bu modülü tamamladıysan, artık bir arkayüzün "nasıl düşündüğünü" biliyorsun — sıradaki adım bunu bir veritabanıyla (Modül 8) ve gerçek bir dille (PHP/Python/C#) hayata geçirmek.

Sunucu ne yanıt döner? YANIT

Tasarladığın "görevler" API'si: `POST /api/v1/gorevler` ile yeni görev oluşturulur (önce gövde doğrulanır, sonra 201 döner); yetkisiz bir istek 401/403 alır; olmayan görev 404 döner; beklenmeyen bir hata loglanıp 500 ile yanıtlanır. Tutarlı, güvenli ve öngörülebilir — iyi bir arkayüzün işareti.

Alıştırma 20 dk

API'ni tasarla:

1. Bir "etkinlikler" sistemi seç; kaynaklarını belirle.
2. CRUD uç noktalarını, her birinin durum kodlarıyla yaz.
3. Güvenlik (kimlik/yetki/doğrulama) ve hata yönetimini tasarıma ekle.
4. API'ni kısaca belgele: bir uç nokta için örnek istek/yanıt yaz.

EK

Backend & API Terimleri Sözlüğü

En sık kullanılan arkayüz ve API terimleri. Bir başvuru kaynağı olarak saklayabilirsin.

İstemci / Sunucu	`İsteyen / yanıtlayan`	HTTP	`Web'in protokolü`
GET / POST	`Oku / oluştur`	PUT / DELETE	`Güncelle / sil`
200 / 404 / 500	`Başarı / yok / hata`	JSON	`Veri alışveriş biçimi`
API	`Uygulama arayüzü`	REST	`API mimari stili`
Endpoint	`Erişim uç noktası`	Token	`Kimlik anahtarı`
CRUD	`Oluştur-Oku-Güncelle-Sil`	HTTPS	`Şifreli HTTP`

Arkayüzün özeti YANIT

Arkayüz, bir **istek** alır (HTTP metot + yol), onu **işler** (rota → mantık → veritabanı) ve bir **yanıt** döner (durum kodu + JSON). REST API'ler bu alışverişi standart ve öngörülebilir kılar; kimlik doğrulama ve güvenlik ise bu kapının bekçisidir. Önyüz görünen yüzse, arkayüz güvenilen beyindir.