WEB & YAZILIM GELİŞTİRME SERİSİ · MODÜL 13

Mobil

Web'i telefona taşımak: mobil-öncelikli ve duyarlı tasarım, viewport, medya sorguları, esnek düzen, görseller, tipografi, dokunmatik ve mobil performans; PWA ile web'i kurulabilir uygulamaya dönüştürmek; native, çapraz platform ve hibrit uygulama yolları; mağazalar, mobil UX, erişilebilirlik, güvenlik ve test. Özgün mobil diyagramlarıyla.

Duyarlı · Dokunmatik · PWA · Uygulama · Eğitim amaçlıdır

Bu Kitap Hakkında

Bu modül, bu seride öğrendiğin web becerisini mobilin gerçeğine taşır: kullanıcıların çoğu telefonda olduğu için, deneyimi önce orada mükemmelleştirmeyi öğretir. Daha çok kavram, tasarım ilkesi ve pratik kodla ilerler; her konuyu net mobil diyagramlarla (telefon önizlemesi, duyarlı ekranlar, uygulama karşılaştırmaları) açıklar. Dört seviye ve yirmi altı bölüm boyunca mobil-öncelikli düşünceden ve viewport'tan, duyarlı tasarıma, medya sorgularına, esnek düzene (Flexbox/Grid), görsel ve tipografiye, dokunmatik etkileşime ve mobil performansa; PWA'lardan (manifest, service worker, çevrimdışı, önbellekleme, bildirimler) gerçek mobil uygulama yaklaşımlarına (native, çapraz platform, hibrit) ve seçim ölçütlerine; uygulama mağazalarından, mobil UX'e, erişilebilirliğe, güvenlik ve gizliliğe (KVKK) ve teste kadar uzanır.

Her bölümde konuyu görselleştiren özgün bir diyagram (telefon önizleme çerçevesi, telefon/tablet/masaüstü duyarlı ekranlar, uygulama yolları karşılaştırması, PWA ve dağıtım şemaları), gerçek HTML/CSS/JS örnekleri, 'telefonda nasıl görünür?' kartı ve bir alıştırma yer alır. Sorumlu geliştirme baştan sona vurgulanır: erişilebilirlik (kontrast, dokunma hedefi, ekran okuyucu, ölçeklenir metin), izinlerde en az ilkesi, HTTPS, hassas veri güvenliği ve KVKK'ya uygun gizlilik. Modül, web becerisini mobile taşımanın bütünsel bir kavrayışıyla ve mobil-hazır kontrol listesiyle kapanır. Bu, on altı modüllük 'Web & Yazılım Geliştirme' serisinin on üçüncü modülü ve FAZ 3'ün ilk adımıdır. Bu seri eğitim amaçlıdır; kod örnekleri modern web standartlarına dayanır.

Web & Yazılım Geliştirme Serisi · Modül 13

İçindekiler

MOBİL-ÖNCELİKLİ WEB

01 Neden Mobil Önce? 6
02 Viewport ve Mobil Ekran 8
03 Responsive (Duyarlı) Tasarım 10
04 Medya Sorguları (Media Queries) 12
05 Esnek Düzen: Flexbox & Grid Mobilde 14
06 Görseller ve Tipografi Mobilde 16
07 Dokunmatik ve Mobil Etkileşim 18
08 Mobil Performans 20

PWA — WEB'İ UYGULAMAYA DÖNÜŞTÜRMEK

09 PWA Nedir? 23
10 Manifest ve Kurulabilirlik 25
11 Service Worker ve Çevrimdışı 27
12 Önbellekleme Stratejileri 29
13 Bildirimler ve Cihaz Özellikleri 31
14 PWA'yı Yayınlamak 33

MOBİL UYGULAMA YAKLAŞIMLARI

15 Mobil Uygulama Yolları 36
16 Native Uygulamalar 38
17 Çapraz Platform (React Native, Flutter) 40
18 Hibrit Uygulamalar 42
19 Hangi Yolu Seçmeli? 44
20 Mobilde API ve Veri 46

YAYINLAMA VE KALİTE

21 Uygulama Mağazaları 49
22 Mobil UX İlkeleri 51
23 Erişilebilirlik (Mobilde) 53
24 Mobil Güvenlik ve Gizlilik 55
25 Test ve Cihaz Uyumluluğu 57
26 Bitirme: Mobil-Hazır Bir Proje 59

★ Mobil Terimleri Sözlüğü 61

SEVİYE 1

Mobil-Öncelikli Web

Temeller: neden mobil önce, viewport ve mobil ekran, duyarlı tasarım, medya sorguları, esnek düzen, görseller ve tipografi, dokunmatik etkileşim, mobil performans.

BÖLÜM 01

Neden Mobil Önce?

Bugün internete giren her on kişiden çoğu telefondan geliyor. "Mobil-öncelikli" (mobile-first) yaklaşım, tasarıma küçük ekrandan başlamak ve oradan büyütmek demektir. Bu, hem kullanıcıların çoğunluğuna öncelik verir hem de daha sağlam bir tasarım üretir.

İki yaklaşım

Şema 1.1 — Mobil-önce: küçükten başla, büyüdükçe geliştir.

- **Çoğunluk mobil:** ziyaretçilerin büyük kısmı telefondan gelir.
- **Kısıt netleştirir:** küçük ekran, önceliği zorunlu kılar.
- **Büyütmek kolay:** küçükten büyüğe gitmek, tersinden kolaydır.

> **İPUCU**
>
> Mobil-öncelikli düşünmek bir **disiplindir**: küçük ekranda yerin sınırlı olduğu için, gerçekten önemli olanı (ana içerik, ana eylem) öne çıkarmak **zorunda** kalırsın — bu da her ekranda daha net bir tasarım üretir. Masaüstünden başlayıp telefona "sıkıştırmak" ise genelde taşan, okunmaz, kullanılamaz sonuçlar verir. Bu yaklaşım, Modül 4'teki (CSS) duyarlı tasarımın felsefesidir: temel düzeni mobil için kur, sonra `@media` sorgularıyla büyük ekranlara **ekle**. Unutma: bir site telefonda iyi çalışıyorsa, masaüstünde de iyi çalışır; tersi her zaman doğru değildir.

Telefonda nasıl görünür? MOBİL

Mobil-öncelikli bir site telefonda açıldığında, içerik tek bir sütunda, büyük ve okunur biçimde, başparmakla rahatça kullanılabilir şekilde dizilir. Aynı site masaüstünde açıldığında, ek alan kullanılarak içerik yan yana yerleşir. Kullanıcı hangi cihazda olursa olsun, tasarım ona uygun ve kullanışlı gelir.

Alıştırma 8 dk

Mobil-önceyi kavra:

1. Mobil-önce ile masaüstü-önce yaklaşımları arasındaki farkı yaz.
2. Sevdiğin bir siteyi telefonda aç; mobil-öncelikli mi, değil mi, değerlendir.
3. Küçük ekranla başlamanın bir avantajını kendi cümlenle açıkla.

BÖLÜM 02

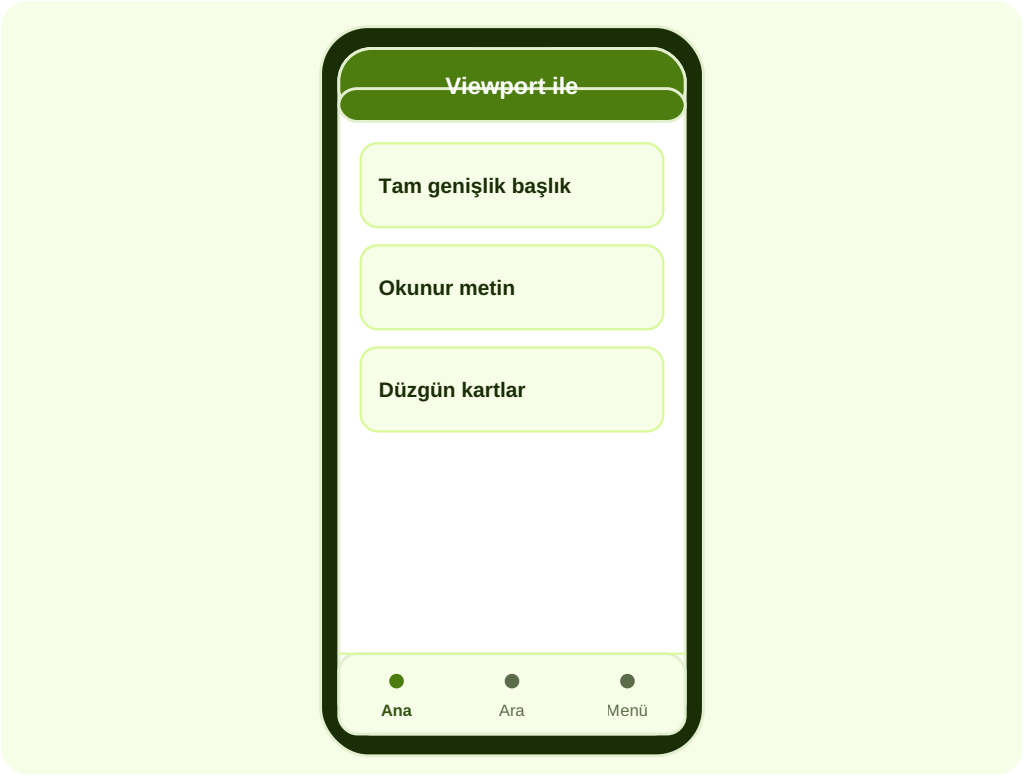
Viewport ve Mobil Ekran

Bir web sayfasının telefonda doğru görünmesi için tek bir satır şarttır: viewport meta etiketi. Bu etiket olmadan telefonlar sayfayı masaüstü genişliğinde gösterip küçültür — yazılar minik, kullanım zor olur. Etiketle ise sayfa gerçek ekran genişliğine uyar.

Viewport meta etiketi

Her sayfanın bölümünde olmalı

```
<meta name="viewport"
      content="width=device-width, initial-scale=1">
```



Şema 2.1 — Viewport etiketiyle sayfa, ekran genişliğine tam oturur.

- `width=device-width` — sayfa genişliği = ekran genişliği.
- `initial-scale=1` — başlangıç yakınlaştırması normal.
- Bu etiket olmadan mobil tasarım düzgün çalışmaz.

İPUCU

Viewport etiketi, duyarlı tasarımın **ön koşuludur** — en güzel mobil CSS'i bile yazsan, bu etiket yoksa telefon sayfayı masaüstü gibi gösterip küçültür ve tüm emeğin boşa gider. Bu yüzden her HTML sayfasının `<head>` bölümüne mutlaka eklenir (Modül 3 ve 4'te gördüğün şablonların hepsinde vardır). `width=device-width` "sayfayı cihazın gerçek genişliğine göre düzenle" der; `initial-scale=1` ise kullanıcı yakınlaştırmadan, normal boyutta başlamasını sağlar. Kullanıcının yakınlaştırmasını engelleyen ayarlardan (`user-scalable=no`) kaçın — bu, görme zorluğu olan kullanıcılar için bir erişilebilirlik sorunudur.

Telefonda nasıl görünür? MOBİL

Viewport etiketi olan bir sayfa telefonda açıldığında, içerik ekran genişliğine tam oturur: yazılar okunur boyutta, kartlar düzgün, kaydırma yalnızca dikey olur. Etiket olmadan ise sayfa minik görünür, kullanıcı yatay kaydırmak ve yakınlaştırmak zorunda kalır. Tek bir satır, mobil deneyimi baştan kurtarır.

Alıştırma 8 dk

Viewport ekle:

1. Viewport meta etiketinin iki parçasının (width, scale) görevini yaz.
2. Bu etiket olmadan bir sayfa telefonda nasıl görünür, açıkla.
3. user-scalable=no kullanmaktan neden kaçınmalı, belirt.

BÖLÜM 03

Responsive (Duyarlı) Tasarım

Duyarlı tasarım, tek bir sitenin her ekran boyutuna uyum sağlamasıdır: telefonda tek sütun, tablette iki, masaüstünde üç sütun. Aynı içerik, ekrana göre yeniden akar. Ayrı bir "mobil site" yapmana gerek kalmaz.

Tek site, her ekran

Şema 3.1 — Duyarlı tasarım: aynı içerik, ekrana göre yeniden düzenlenir.

- **Akışkan düzen:** sabit piksel yerine esnek genişlikler (%/fr).
- **Esnek görseller:** kapsayıcıya sığacak şekilde ölçeklenir.
- **Kırılma noktaları:** belli genişliklerde düzen değişir.

İPUCU

Duyarlı tasarımın özü, **sabit ölçüler yerine esnek ölçüler** kullanmaktır: `width: 600px` demek yerine `width: 100%` veya `max-width` dersin; sütunları piksel yerine yüzde veya grid `fr` birimiyle tanımlarsın. Böylece içerik, ekran ne olursa olsun uyum sağlar. Düzenin köklü değiştiği yerlerde (örneğin tek sütundan iki sütuna geçiş) **kırılma noktaları** (breakpoints) kullanırsın — bunları bir sonraki bölümde göreceğiz. "Mobil için ayrı, masaüstü için ayrı site" yapmak eski ve zahmetli bir yöntemdir; modern yaklaşım **tek duyarlı site**'dir — bakımı kolay, tutarlı ve arama motoru dostudur.

Telefonda nasıl görünür? MOBİL

Duyarlı bir site, telefonda içeriği tek sütunda dikey dizer; tablette ekran genişledikçe iki sütuna, masaüstünde üç sütuna geçer — hep aynı HTML, hep aynı içerik. Kullanıcı hangi cihazda olursa olsun, içerik o ekrana en uygun biçimde yerleşir. Tek bir site, her yerde düzgün çalışır.

◎ Alıştırma

12 dk

Duyarlı düşün:

1. Duyarlı tasarımın "tek site, her ekran" ilkesini kendi cümlenle açıkla.
2. Sabit piksel yerine neden esnek ölçü kullanılır, yaz.
3. Bir haber sitesinin telefon/tablet/masaüstünde nasıl akacağını tarif et.

BÖLÜM 04

Medya Sorguları (Media Queries)

Medya sorguları, "ekran şu genişlikten büyükse şu kuralları uygula" demenin CSS yoludur. Mobil-öncelikli yaklaşımda temel stili telefon için yazar, sonra @media ile büyük ekranlara ekleme yaparsın. Bunlar duyarlı tasarımın motorudur.

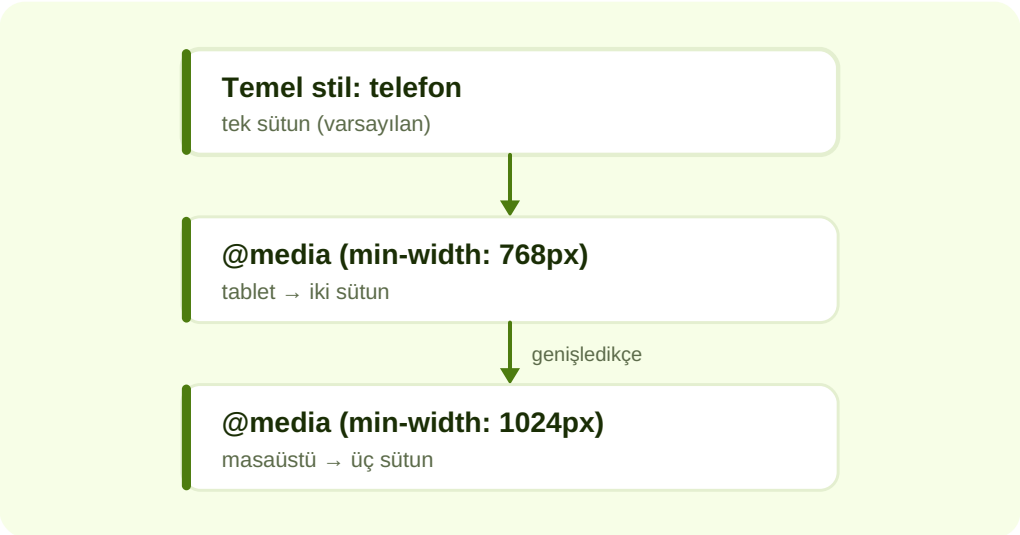
@media ile kırılma noktaları

Mobil-öncelikli medya sorgusu (CSS)

```
 /* Temel: telefon (tek sütun) */
.kartlar { display: grid; grid-template-columns: 1fr; }

/* Tablet ve üzeri: iki sütun */
@media (min-width: 768px) {
  .kartlar { grid-template-columns: 1fr 1fr; }
}

/* Masaüstü: üç sütun */
@media (min-width: 1024px) {
  .kartlar { grid-template-columns: 1fr 1fr 1fr; }
}
```



Şema 4.1 — Mobil-öncelikli medya sorguları: temelden büyüğe ekle.

İPUCU

min-width ile yazmak, mobil-öncelikli yaklaşımın özüdür: temel (varsayılan) kuralların telefon içindir, sonra ekran **büyüdükçe** kuralları geliştirirsin. Bu, `max-width` ile (masaüstünden başlayıp küçülterek) yazmaktan daha temiz ve sürdürülebilirdir. **Kırılma noktalarını içeriğe göre seç**, belirli cihazlara göre değil — "tasarım şu genişlikte bozuluyor, burada bir sütun ekleyeyim" diye düşün (768px ve 1024px yaygın başlangıç değerleridir ama kuralı içeriğin belirler). Çok fazla kırılma noktası karmaşa yaratır; genelde iki-üç tanesi (telefon/tablet/masaüstü) yeterlidir. Bu, Modül 4'teki CSS bilginin doğrudan mobil uygulamasıdır.

Telefonda nasıl görünür? MOBİL

Bu CSS ile `.kartlar` telefonda tek sütun olur (temel kural); ekran 768px'i geçince iki sütuna, 1024px'i geçince üç sütuna döner. Kullanıcı tarayıcı penceresini büyütüp küçülttükçe düzen anında uyum sağlar. Tek bir stil dosyası, üç farklı ekran düzenini yönetir.

Alıştırma 12 dk

Medya sorgusu yaz:

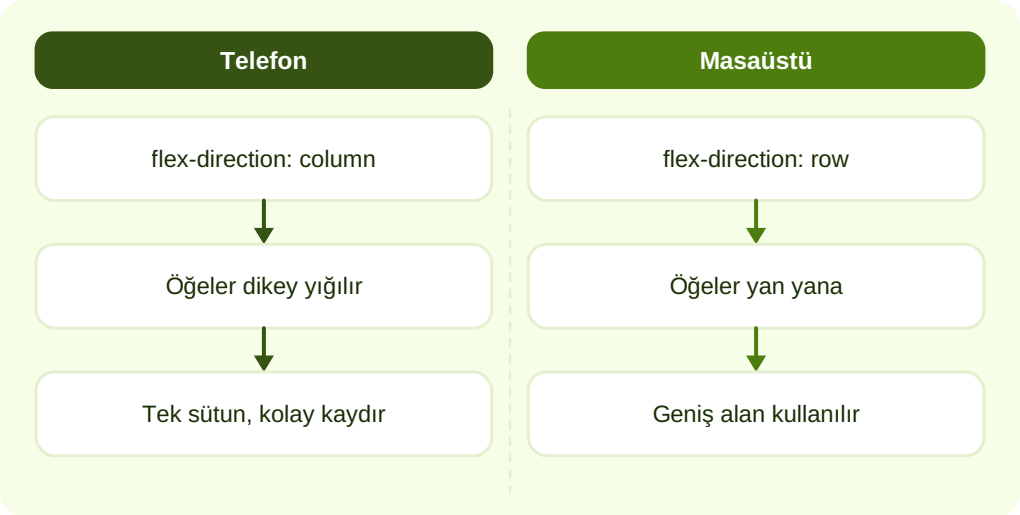
1. Telefonda tek, masaüstünde iki sütun olan bir düzen için CSS taslağı yaz.
2. min-width ile max-width yaklaşımı arasındaki farkı açıkla.
3. Kırılma noktasını neye göre seçmek gerektiğini belirt.

BÖLÜM 05

Esnek Düzen: Flexbox & Grid Mobilde

Flexbox ve Grid (Modül 4), duyarlı düzenin en güçlü araçlarıdır. Mobilde içeriği dikey yığar, ekran genişleyince yan yana dizersin. Çoğu zaman medya sorgusuna bile gerek kalmadan, bu araçlar kendiliğinden uyum sağlar.

Dikeyden yataya



Şema 5.1 — Flexbox: mobilde dikey yığ, masaüstünde yan yana diz.

Otomatik duyarlı grid (medya sorgusu olmadan)

```
.galeri {
  display: grid;
  /* en az 220px, sığdığı kadar sütun */
  grid-template-columns:
    repeat(auto-fit, minmax(220px, 1fr));
  gap: 16px;
}
```

İPUCU

Modern CSS, duyarlılığı **kendiliğinden** sağlayan güçlü desenler sunar. Yukarıdaki `auto-fit` + `minmax` kalıbı sihirlidir: "her öğe en az 220px olsun, ekrana kaç tane sığarsa o kadar sütun yap" der — telefonda tek, tablette iki, geniş ekranda dört sütun, hepsi tek satır CSS ile, medya sorgusu olmadan. Flexbox'ta `flex-wrap: wrap` ile öğeler sığmadığında alt satıra geçer. Bu araçları kullanınca çoğu düzen, az kodla ve doğal olarak duyarlı olur. Modül 4'teki Flexbox ve Grid bilgin burada doğrudan işe yarıyor — mobilde fark, "varsayılanı dikey/tek sütun tutmak", genişledikçe açmaktır.

Telefonda nasıl görünür? MOBİL

Bu grid kuralıyla galeri, telefonda tek sütun (220px sığar), tablette iki-üç sütun, geniş ekranda dört-beş sütun olur — sen sayıyı belirtmeden, ekran kendiliğinden karar verir. Flexbox'ta ise `column` yönü mobilde dikey yığar, `row` masaüstünde yan yana dizer. İçerik her ekranda doğal ve düzenli akar.

Alıştırma 12 dk

Esnek düzen kur:

1. auto-fit + minmax kalıbının ne yaptığını kendi cümlenle açıkla.
2. Flexbox ile mobilde dikey, masaüstünde yatay düzen nasıl kurulur, yaz.
3. flex-wrap: wrap ne işe yarar, belirt.

BÖLÜM 06

Görseller ve Tipografi Mobilde

Mobilde iki şey kritiktir: görseller ekranı taşırmadan, hızlı yüklenecek; yazılar ise yakınlaştırmaya gerek kalmadan okunacak. Birkaç basit kural, mobil okunabilirliği ve hızı dönüştürür.

Esnek görsel, okunur metin

Şema 6.1 — Mobil görsel ve tipografinin dört temel kuralı.

Görseli taşırmadan ölçekle (CSS)

```
img {
  max-width: 100%;   /* kapsayıcıyı aşma */
  height: auto;      /* oranı koru */
}
body { font-size: 16px; line-height: 1.6; }
```

İPUCU

Görseller mobilde en sık taşma ve yavaşlık sebebidir. `max-width: 100%` ile görselin kapsayıcısını aşmasını engellersin; `srcset` ile tarayıcının ekran boyutuna uygun (küçük ekrana küçük dosya) sürümü seçmesini sağlarsın — bu, mobil veri ve hız için büyük fark yaratır. **Tipografide**: gövde metni için en az **16px** kullan (daha küçüğü telefonda yakınlaştırma gerektirir ve okumayı zorlaştırır), satır aralığını rahat tut (1.5–1.6) ve metin ile arka plan arasında **yeterli kontrast** bırak (erişilebilirlik için). Satır uzunluğunu da gözet — bir satırda çok fazla karakter, mobilde okumayı yorar. Bu kurallar hem okunabilirliği hem erişilebilirliği yükseltir.

Telefonda nasıl görünür? MOBİL

`max-width: 100%` kuralıyla bir görsel, telefonda kapsayıcısından taşmaz; ekran küçüldükçe görsel de orantılı küçülür. 16px gövde metni ve rahat satır aralığıyla yazılar yakınlaştırmaya gerek kalmadan okunur. Sonuç: hızlı yüklenen, taşmayan, rahatça okunan bir mobil sayfa.

Alıştırma

10 dk

Mobile uyarla:

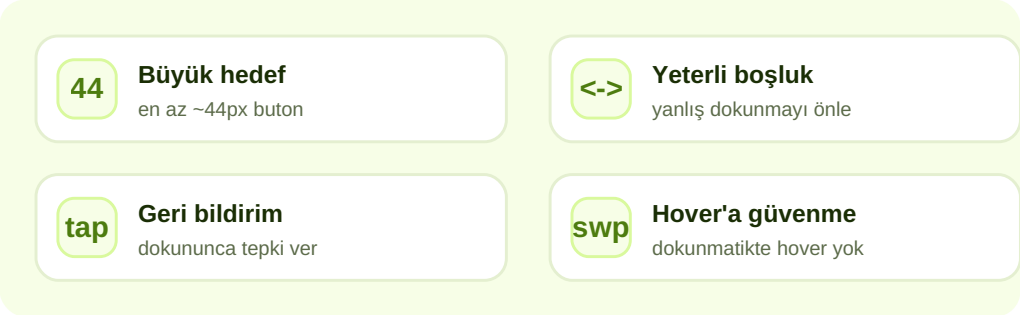
1. Bir görselin telefonda taşmaması için hangi CSS kuralını kullanırsın?
2. Mobil gövde metni için neden en az 16px önerilir, açıkla.
3. srcset ne işe yarar, kendi cümlenle yaz.

BÖLÜM 07

Dokunmatik ve Mobil Etkileşim

Telefonda kullanıcı fareyle değil parmağıyla etkileşir. Bu, tasarımda farklar gerektirir: dokunma hedefleri yeterince büyük olmalı, "üzerine gelince" (hover) etkilerine güvenilmemeli ve kaydırma/dokunma akıcı olmalı. Parmak, fare imlecinden çok daha kalındır.

Parmak dostu tasarım



Şema 7.1 — Dokunmatik etkileşimin dört temel kuralı.

- **Dokunma hedefi:** butonlar/bağlantılar en az ~44×44px olmalı.
- **Boşluk:** tıklanabilir öğeler arasında yeterli aralık bırak.
- **Hover yok:** telefonda "üzerine gelme" etkisi çalışmaz.
- **Geri bildirim:** dokunmaya görsel tepki ver (renk değişimi vb.).

İPUCU

Parmak, fare imlecinden çok daha geniş bir alana dokunur; bu yüzden butonlar ve bağlantılar yeterince **büyük** (yaklaşık 44×44 piksel, bir parmak ucu kadar) ve aralarında yeterli **boşluk** olmalıdır — yoksa kullanıcı yanlış öğeye dokunur ve sinirlenir. **Hover (üzerine gelme) etkilerine güvenme**: dokunmatik ekranda "üzerine gelmek" diye bir şey yoktur, bu yüzden önemli bilgiyi yalnızca hover'da gösterme. Dokunmaya **anında görsel geri bildirim** ver (rengin değişmesi, hafif bir gölge) ki kullanıcı dokunuşunun algılandığını anlasın. Kaydırma ve sürükleme gibi mobil jestleri doğal hisler verir ama beklenmedik yerlerde kullanmak kafa karıştırır — alışılmış kalıplara sadık kal.

Telefonda nasıl görünür? MOBİL

Parmak dostu bir arayüzde butonlar telefonda rahatça basılacak kadar büyüktür, aralarında yanlış dokunmayı önleyecek boşluk vardır ve her dokunuş anında bir tepki (renk değişimi gibi) verir. Hover'a bağlı menüler yerine, dokunmayla açılan net öğeler kullanılır. Kullanıcı, küçük hedefleri zorlamak yerine akıcı ve hatasız bir deneyim yaşar.

◎ Alıştırma

10 dk

Dokunmatiğe uyarla:

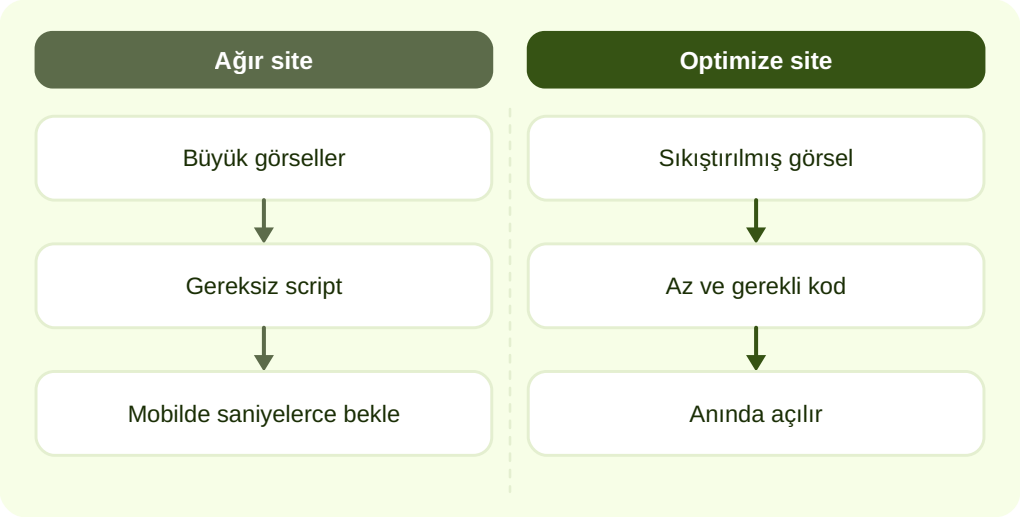
1. Bir butonun mobilde ne kadar büyük olması gerektiğini ve nedenini yaz.
2. Neden hover etkilerine güvenmemeli, açıkla.
3. Dokunmaya geri bildirim vermenin önemini kendi cümlenle belirt.

BÖLÜM 08

Mobil Performans

Mobil kullanıcılar genelde daha yavaş bağlantı ve daha az güçlü cihazlarla gelir. Ağır bir site masaüstünde sorunsuz açılırken telefonda saniyelerce bekletebilir. Mobil performans, kullanıcıyı kaybetmemenin anahtarıdır: yavaş açılan siteyi insanlar terk eder.

Hafif ve hızlı



Şema 8.1 — Mobil performans: hafif site, hızlı açılır ve kullanıcıyı tutar.

- **Görselleri optimize et:** en büyük ağırlık genelde görsellerdir.
- **Kodu küçült:** gereksiz script ve stilden kaçın.
- **Önbellek ve CDN:** tekrar eden yüklemeleri hızlandır (Modül 12).

İPUCU

Mobil performansın en büyük kaldıracı genelde **görsellerdir**: doğru boyut (telefona dev bir masaüstü görseli gönderme), modern biçimler (WebP/AVIF) ve sıkıştırma, sayfa ağırlığını dramatik düşürür. İkinci kaldıraç **JavaScript**: gereksiz veya büyük scriptler, özellikle düşük güçlü telefonlarda sayfayı yavaşlatır — yalnızca gerekeni yükle. Modül 12'deki **önbellekleme ve CDN** burada da kritiktir. Performansı tahmin etme, **ölç**: tarayıcıların ve ücretsiz araçların mobil performans testleri, neyin yavaşlattığını tam söyler. Unutma: kullanıcılar yavaş açılan siteleri saniyeler içinde terk eder; hız, doğrudan kullanıcı kaybı veya kazancı demektir. Mobil-öncelikli düşünmek, hafif düşünmektir.

Telefonda nasıl görünür?

MOBİL

Optimize edilmiş bir site telefonda, yavaş bir bağlantıda bile hızla açılır: görseller küçük ve sıkıştırılmıştır, gereksiz kod yoktur, tekrar eden kaynaklar önbellekten gelir. Ağır bir site ise aynı koşulda kullanıcıyı saniyelerce bekletir — ve çoğu kullanıcı beklemez, çıkar. Hız, mobilde lüks değil, zorunluluktur.

Alıştırma

10 dk

Hızı artır:

1. Mobilde sayfayı yavaşlatan en büyük etkenleri yaz.
2. Görsel optimizasyonunun neden bu kadar önemli olduğunu açıkla.
3. Performansı "tahmin etmek yerine ölçmek" ne demek, belirt.

SEVİYE 2

PWA — Web'i Uygulamaya Dönüştürmek

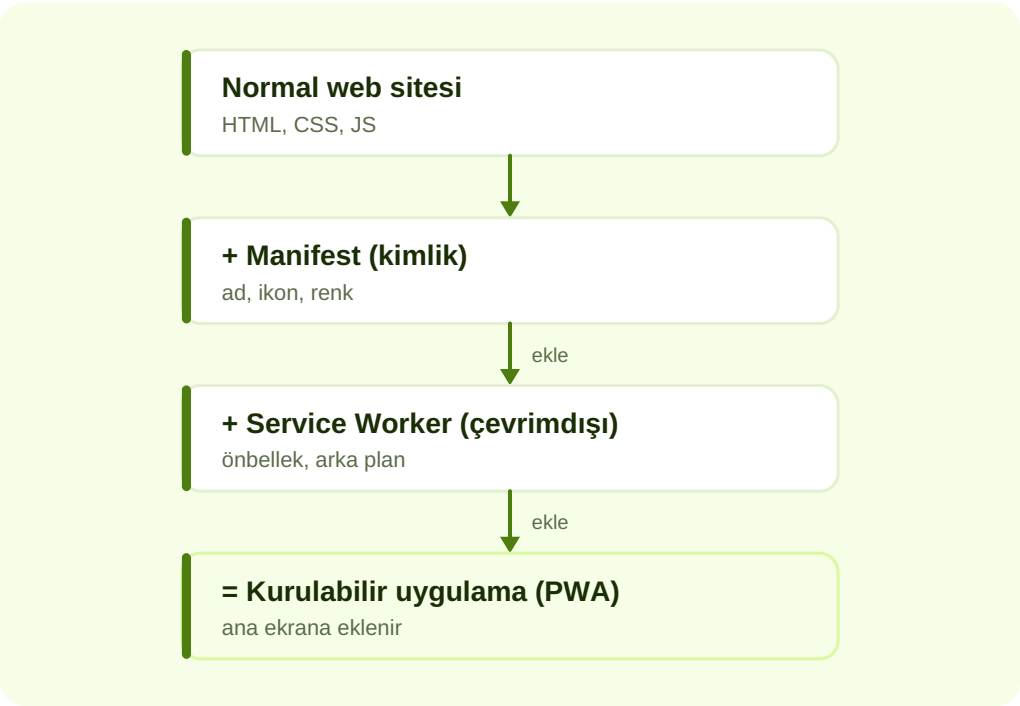
Web sitesini kurulabilir, çevrimdışı çalışan bir uygulamaya çevirmek: PWA nedir, manifest ve kurulabilirlik, service worker, önbellekleme stratejileri, bildirimler ve yayınlama.

BÖLÜM 09

PWA Nedir?

PWA (Progressive Web App / Aşamalı Web Uygulaması), normal bir web sitesini, kurulabilen ve uygulama gibi davranan bir şeye dönüştüren tekniklerin bütünüdür. Kullanıcı onu ana ekrana ekler, tam ekran açar, hatta çevrimdışı kullanır — ama altında hâlâ web vardır.

Web + uygulama yetenekleri



Şema 9.1 — PWA: web sitesi + manifest + service worker = kurulabilir uygulama.

- **Kurulabilir:** ana ekrana eklenir, ikonu olur.
- **Çevrimdışı çalışabilir:** service worker sayesinde.
- **Tek kod tabanı:** hem web hem "uygulama" aynı koddan.

İPUCU

PWA'nın güzelliği, **mağazaya gerek kalmadan** uygulama deneyimi sunmasıdır: kullanıcı siteni tarayıcıda açar, "ana ekrana ekle" der ve artık bir uygulama ikonuna sahip olur — indirme, mağaza onayı, güncelleme derdi yok. Bu seri boyunca yaptığın tüm eğitim sitelerinin (bu dahil) birer PWA olduğunu fark etmişsindir: çevrimdışı açılır, kurulabilir. PWA, web ile native uygulama arasında pratik bir orta yoldur — web'in **erişilebilirliği ve kolay dağıtımı** ile uygulamanın **kurulabilirlik ve çevrimdışı** avantajlarını birleştirir. Her şey için yeterli olmasa da (çok ağır oyunlar, derin donanım erişimi gerektiren uygulamalar hariç), şaşırtıcı sayıda proje için PWA fazlasıyla yeterlidir ve en hızlı yoldur.

Telefonda nasıl görünür? MOBİL

Bir PWA telefonda açıldığında normal bir site gibi görünür; ama kullanıcı "ana ekrana ekle" dediğinde, bir uygulama ikonu oluşur ve sonraki açılışlarda adres çubuğu olmadan, tam ekran bir uygulama gibi başlar. İnternet kesilse bile (service worker sayesinde) açılmaya devam edebilir. Kullanıcı için bu, indirilen bir uygulamadan ayırt edilemez.

Alıştırma 10 dk

PWA'yı kavra:

1. PWA'yı normal bir web sitesinden ayıran iki yeteneği yaz.
2. PWA'nın native uygulamaya göre bir avantajını açıkla.
3. Bu eğitim sitesinin neden bir PWA olduğunu kendi cümlenle belirt.

BÖLÜM 10

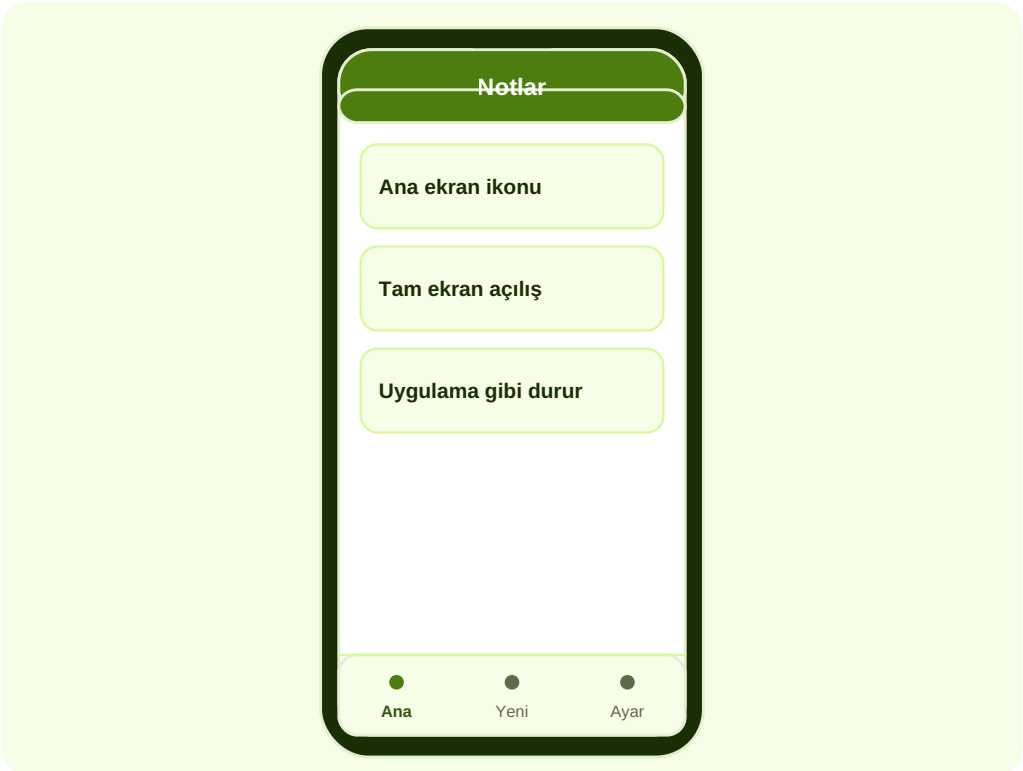
Manifest ve Kurulabilirlik

Web app manifest, PWA'nın kimlik kartıdır: uygulamanın adını, ikonlarını, renklerini ve nasıl açılacağını tanımlayan küçük bir JSON dosyası. Bu dosya sayesinde tarayıcı, siteni "kurulabilir bir uygulama" olarak tanır ve ana ekrana eklemeyi önerir.

manifest.json

Basit bir web app manifest

```
{
  "name": "Notlarım",
  "short_name": "Notlar",
  "start_url": "index.html",
  "display": "standalone",
  "background_color": "#ffffff",
  "theme_color": "#4D7C0F",
  "icons": [
    { "src": "icon-192.png", "sizes": "192x192" },
    { "src": "icon-512.png", "sizes": "512x512" }
  ]
}
```



Şema 10.1 — Manifest ile uygulama ana ekrana eklenir, tam ekran açılır.

İPUCU

Manifest dosyası, HTML'in `<head>` bölümüne `<link rel="manifest">` ile bağlanır. Kurulabilir bir PWA için manifest'in iki kritik alanı vardır: **ikonlar** (en az 192px ve 512px boyutlarında, ana ekranda görünecek) ve `display: "standalone"` (uygulamanın adres çubuğu olmadan, tam ekran açılmasını sağlar). `theme_color` ve `background_color`, açılış ekranının ve durum çubuğunun rengini belirler — uygulamana markalı bir his katar. `start_url`, uygulama açıldığında hangi sayfadan başlayacağını söyler. Bu dosyayı bir kez doğru kurduğunda (artı bir service worker ve HTTPS), tarayıcılar kullanıcıya "ana ekrana ekle" seçeneğini otomatik sunar.

Telefonda nasıl görünür? MOBİL

Manifest dosyasını ekleyip siteyi açtığında, tarayıcı uygulamayı kurulabilir olarak tanır ve "ana ekrana ekle" önerir. Kullanıcı eklediğinde, telefonun ana ekranında manifest'teki ikon ve adla bir uygulama belirir; dokununca `standalone` sayesinde tam ekran, markalı renklerle açılır. Web sitesi, görsel olarak gerçek bir uygulamaya dönüşür.

Alıştırma 12 dk

Manifest yaz:

1. Bir uygulama için ad, ikon ve renkleri olan kısa bir manifest taslağı yaz.
2. display: "standalone" ne işe yarar, açıkla.
3. Kurulabilir bir PWA için manifest'te hangi iki alan kritiktir, belirt.

BÖLÜM 11

Service Worker ve Çevrimdışı

Service worker, sitenle ağ arasında duran ve istekleri yakalayabilen özel bir scripttir. Asıl gücü budur: ziyaret edilen sayfaları önbelleğe alıp, internet kesildiğinde onları önbellekten sunarak siteyi çevrimdışı çalıştırabilir.

Araya giren katman

Şema 11.1 — Service worker: istekleri yakalar, önbellekten sunar, çevrimdışı çalıştırır.

Service worker'ı kaydetmek (JS)

```
if ("serviceWorker" in navigator) {
  navigator.serviceWorker
    .register("service-worker.js");
}
```

İPUCU

Service worker, bir kez kaydedildikten sonra **arka planda** çalışır ve sayfa kapansa bile yaşamını sürdürebilir. En yaygın kullanımı **çevrimdışı destektir**: ilk ziyarette önemli dosyaları (HTML, CSS, JS, ikonlar) önbelleğe alır; internet kesildiğinde tarayıcı yerine service worker yanıt vererek siteyi açık tutar. Bu seri sitelerinin metroda, uçakta, kapsama dışında bile açılmasının sebebi budur. Service worker **yalnızca HTTPS'te** (ve geliştirme için localhost'ta) çalışır — bu bir güvenlik gereğidir (Modül 12'deki HTTPS'in önemi burada da geçerli). Güçlü bir araçtır ama dikkat ister: yanlış önbellek yönetimi kullanıcılara **eski sürümü** gösterebilir; bu yüzden önbellek sürümleme (cache versioning) önemlidir.

Telefonda nasıl görünür? MOBİL

Service worker kaydedildikten sonra, kullanıcı siteyi her açtığında istekler önce ondan geçer: ihtiyaç duyulan dosya önbellekteyse anında (ve çevrimdışı bile) sunulur; değilse ağdan getirilir ve bir sonraki sefer için saklanır. Sonuç: site hem daha hızlı açılır hem de internet olmadan çalışabilir. Kullanıcı, "bağlantı yok" hatası yerine çalışan bir uygulama görür.

Alıştırma 12 dk

Çevrimdışını kavra:

1. Service worker'ın temel görevini kendi cümlenle açıkla.
2. Bir sitenin çevrimdışı açılması nasıl mümkün olur, yaz.
3. Service worker'ın neden HTTPS gerektirdiğini belirt.

BÖLÜM 12

Önbellekleme Stratejileri

Service worker'la neyi, ne zaman önbellekten sunacağına sen karar verirsin. Farklı içerikler farklı strateji ister: değişmeyen dosyalar için "önce önbellek", güncel veri için "önce ağ". Doğru stratejiyi seçmek, hız ile tazelik arasındaki dengeyi kurar.

İki temel strateji

Şema 12.1 — Önbellekleme: statik için "önce önbellek", güncel için "önce ağ".

- **Önce önbellek (cache-first):** CSS, JS, ikon gibi sabit dosyalar.
- **Önce ağ (network-first):** haber, fiyat gibi güncel veri.
- **Sürümleme:** dosya güncellenince önbelleği tazele.

İPUCU

Strateji seçimi, içeriğin **ne sıklıkla değiştiğine** bağlıdır. Logon, stil dosyan, scriptlerin gibi **nadiren değişen** şeyler için "önce önbellek" idealdir — anında, çevrimdışı, hızlı. Haber akışı, stok durumu, kullanıcı verisi gibi **güncel olması gereken** şeyler için "önce ağ" uygundur (ağ yoksa son bilinen sürümü önbellekten gösterir). Bu dengeyi kurmak performansın anahtarıdır. En sık hata **önbellek tazeleme**yi unutmaktır: bir dosyayı güncellediğinde, eski önbellek silinmeli veya sürümlenmeli (örn. `cache-v2`) — yoksa kullanıcılar değişikliği görmez. Stratejini basit tut; karmaşık önbellek mantığı zor ayıklanan hatalara yol açar.

Telefonda nasıl görünür? MOBİL

"Önce önbellek" stratejisinde, sabit dosyalar (stil, script, ikon) önbellekten anında sunulur — sayfa şimşek hızında açılır. "Önce ağ" stratejisinde ise güncel veri (örneğin bir liste) önce ağdan istenir; bağlantı yoksa son kaydedilen sürüm gösterilir. Kullanıcı her durumda çalışan bir arayüz görür: hızlı açılan kabuk, güncel (ya da en azından son bilinen) içerik.

Alıştırma 10 dk

Strateji seç:

1. "Önce önbellek" hangi tür dosyalar için uygundur, örnek ver.
2. "Önce ağ" hangi tür içerik için tercih edilir, yaz.
3. Önbellek sürümlemenin neden gerekli olduğunu açıkla.

BÖLÜM 13

Bildirimler ve Cihaz Özellikleri

Modern web, eskiden yalnızca native uygulamalara özel olan birçok cihaz yeteneğine erişebilir: bildirim gönderme, konum, kamera, titreşim ve daha fazlası. Ama bu güçle birlikte sorumluluk gelir: her biri kullanıcı izni gerektirir ve dikkatli kullanılmalıdır.

Web'in cihaz yetenekleri

Şema 13.1 — Web'in cihaz yetenekleri: hepsi kullanıcı iznine bağlı.

- **Push bildirimleri:** kullanıcı kapalıyken bile haber ver (izinle).
- **Cihaz API'leri:** konum, kamera, titreşim, çevrimdışı.
- **İzin modeli:** her hassas yetenek kullanıcı onayı ister.

İPUCU

Bu yetenekler güçlüdür ama **izin ve güven** üzerine kuruludur. En sık yapılan hata, kullanıcı siteye girer girmez **bildirim izni istemektir** — bu rahatsız edici ve itici bir davranıştır; çoğu kullanıcı hemen reddeder. Bunun yerine izni, **kullanıcının bir fayda gördüğü anda** ve **neden gerektiğini açıklayarak** iste (örneğin "siparişin hazır olunca haber verelim mi?"). Konum, kamera gibi hassas erişimleri yalnızca **gerçekten gerektiğinde** kullan ve aldığın veriyi sorumlu biçimde işle. Bu, bir sonraki güvenlik/gizlilik bölümünün ve KVKK'nın doğrudan konusudur: **en az veri ilkesi** — ihtiyacın olmayan izni isteme, olmayan veriyi toplama. İzinler, kötüye kullanıldığında kullanıcı güvenini ve uygulamanı zedeler.

Telefonda nasıl görünür? MOBİL

Bir PWA, kullanıcı izin verdiğinde push bildirimi gönderebilir (uygulama kapalıyken bile), konumunu kullanabilir veya kamerasına erişebilir — tıpkı bir native uygulama gibi. Ama her hassas yetenek için tarayıcı, kullanıcıya açık bir izin sorusu gösterir; kullanıcı reddederse o özellik çalışmaz. Bu izin modeli, kullanıcıyı korur ve geliştiriciyi sorumlu davranmaya yönlendirir.

Alıştırma 10 dk

İzinleri yönet:

1. Web'in eriştiği üç cihaz yeteneği yaz ve hepsinin ortak koşulunu belirt.
2. Bildirim iznini ne zaman istemek doğru, ne zaman yanlış, açıkla.
3. "En az veri ilkesi"ni kendi cümlenle tanımla.

BÖLÜM 14

PWA'yı Yayınlamak

Bir PWA'yı yayınlamak, normal bir web sitesini yayınlamakla aynıdır (Modül 12): HTTPS bir sunucuya koyarsın. Kullanıcılar tarayıcıdan kurar. İstersen, aynı PWA'yı uygulama mağazalarına da gönderebilirsin — tek kod tabanı, birçok dağıtım kanalı.

Yayın yolları

Şema 14.1 — PWA yayını: HTTPS'e koy, kullanıcı kurar, isteğe bağlı mağaza.

İPUCU

PWA'nın en pratik yanı, dağıtımının **web kadar kolay** olmasıdır: Modül 12'de öğrendiğin gibi HTTPS bir sunucuya koyarsın, bağlantıyı paylaşırsın, kullanıcı tarayıcıdan "ana ekrana ekle" der — mağaza onayı, indirme süreci yoktur ve güncellemeler anında yayılır (kullanıcı uygulamayı her açtığında en güncel sürümü alır). Buna ek olarak, PWA'yı paketleyip **uygulama mağazalarına** da gönderebilirsin (araçlar bunu kolaylaştırır) — böylece hem web hem mağaza kanalını aynı koddan beslersin. **Kurulabilirlik kontrol listesi**: HTTPS aktif, geçerli manifest (ikonlar + standalone), kayıtlı service worker. Üçü tamamsa tarayıcılar kurulumu otomatik önerir. Bu, "uygulama yapmak" için en düşük engelli yoldur.

Telefonda nasıl görünür? MOBİL

Bir PWA'yı yayına aldığında (HTTPS sunucuya koyarak), kullanıcılar bağlantıya girip tarayıcıdan kurabilir — App Store/Play Store sürecine gerek yoktur ve güncellemeler anında ulaşır. İstersen aynı uygulamayı paketleyip mağazalara da koyabilirsin. Sonuç: tek bir kod tabanından hem web sitesi, hem kurulabilir uygulama, hem (istenirse) mağaza uygulaması elde edersin.

Alıştırma 10 dk

PWA'yı yayınla:

1. Bir PWA'nın yayınlanmasının normal web yayınıyla aynı olduğunu açıkla.
2. PWA güncellemelerinin mağaza uygulamalarına göre avantajını yaz.
3. Kurulabilirlik için gereken üç koşulu listele.

SEVİYE 3

Mobil Uygulama Yaklaşımları

Gerçek mobil uygulamalara giden yollar: native, çapraz platform ve hibrit yaklaşımlar, hangisini seçmeli ve mobilde API ile veri.

BÖLÜM 15

Mobil Uygulama Yolları

Gerçek bir mobil uygulama yapmanın tek yolu yoktur; üç ana yaklaşım vardır. Native en güçlü ama en pahalı; çapraz platform tek kodla iki platform; PWA en hızlı ve ucuz. Hangisinin doğru olduğu, projenin ihtiyaçlarına bağlıdır.

Üç ana yol

Şema 15.1 — Üç uygulama yolu: performans, maliyet ve erişim dengesi.

İPUCU

Bu üç yol bir **denge tablosudur**; "en iyi" yol yoktur, projeye uygun yol vardır. **Native** (Swift/iOS, Kotlin/Android) en yüksek performansı ve en derin donanım erişimini verir ama her platform için **ayrı kod** (ve ayrı ekip/maliyet) gerektirir. **Çapraz platform** (React Native, Flutter) tek kod tabanından hem iOS hem Android üretir — maliyeti düşürür, çoğu uygulama için yeterli performans sağlar. **PWA** (Seviye 2) en ucuz ve hızlı yoldur, web becerinle yaparsın, ama bazı derin donanım özelliklerine ve mağaza görünürlüğüne sınırlı erişimi vardır. Genel kural: **basit başla**. Çoğu fikir için PWA veya çapraz platform fazlasıyla yeterlidir; native'i yalnızca gerçekten gerektiğinde (üst düzey performans, özel donanım) seç.

Telefonda nasıl görünür? MOBİL

Bir mobil fikrini hayata geçirirken üç yoldan birini seçersin: en yüksek performans ve donanım erişimi için native (ama platform başına ayrı kod), maliyet/performans dengesi için çapraz platform (tek kod, iki platform), veya en hızlı ve ucuz yol için PWA (web becerinle). Doğru seçim; bütçe, performans ihtiyacı, ekip becerisi ve donanım gereksinimine göre değişir.

Alıştırma 12 dk

Yolları karşılaştır:

1. Üç uygulama yolunu performans, maliyet ve platform açısından karşılaştır.
2. Hangi durumda native'i tercih edersin, bir örnek ver.
3. Çoğu fikir için neden önce PWA/çapraz platform önerilir, açıkla.

BÖLÜM 16

Native Uygulamalar

Native uygulama, bir platformun kendi diliyle ve araçlarıyla yazılan uygulamadır: iOS için Swift, Android için Kotlin. Platforma özel olduğu için en yüksek performansı ve en tam donanım erişimini sunar — karşılığında her platform için ayrı geliştirme gerekir.

Platforma özel güç

Şema 16.1 — Native: en güçlü ama platform başına ayrı kod.

- **Avantaj:** en iyi performans, tam donanım erişimi, en akıcı his.
- **Dezavantaj:** platform başına ayrı kod, yüksek maliyet.
- **Kim için:** performans/donanım kritik uygulamalar (oyun, kamera vb.).

İPUCU

Native uygulamalar, bir platformun **tüm gücüne** doğrudan erişir: en yüksek hız, en akıcı animasyonlar, kameranın/sensörlerin/donanım hızlandırmanın tam kontrolü ve platformun en güncel özelliklerine anında erişim. Bu yüzden ağır oyunlar, gelişmiş kamera uygulamaları veya yüksek performans gerektiren araçlar genelde native yazılır. Bedeli ise **ikiye katlanan iş**: iOS ve Android için farklı diller, farklı araçlar, ayrı kod tabanları ve ayrı bakım — bu da daha fazla zaman, para ve ekip demektir. Bu maliyet, çoğu sıradan uygulama (içerik, e-ticaret, hizmet) için gereksizdir; o yüzden native'i bir **ihtiyaç** olduğunda seç, varsayılan olarak değil. "Önce en basit yeterli yolu dene" ilkesi burada da geçerlidir.

Telefonda nasıl görünür?

MOBİL

Native bir uygulama, cihazın tüm yeteneklerini en yüksek performansla kullanır; kullanıcı en akıcı, platforma en doğal hisseden deneyimi yaşar. Bunun bedeli, iOS ve Android için iki ayrı uygulamanın geliştirilip sürdürülmesidir — bu da en yüksek maliyetli yoldur. Performans veya donanım gerçekten kritik olduğunda native parlar; aksi halde daha ekonomik yollar genelde yeterlidir.

Alıştırma

10 dk

Native'i değerlendir:

1. Native uygulamanın iki avantajını ve iki dezavantajını yaz.
2. iOS ve Android için hangi diller kullanılır, belirt.
3. Hangi tür uygulamalar için native gereklidir, örnek ver.

BÖLÜM 17

Çapraz Platform (React Native, Flutter)

Çapraz platform yaklaşımı, tek bir kod tabanından hem iOS hem Android uygulaması üretmeni sağlar. React Native ve Flutter en popüler araçlardır. Native'e yakın performans sunarken, maliyeti ve geliştirme süresini önemli ölçüde düşürür.

Tek kod, iki platform

Şema 17.1 — Çapraz platform: tek koddan iki platforma.

- **React Native:** JavaScript/React bilenler için tanıdık.
- **Flutter:** Dart diliyle, akıcı arayüzler.
- **Avantaj:** tek kod, düşük maliyet, geniş donanım erişimi.

İPUCU

Çapraz platform, çoğu modern uygulama için **tatlı noktadır**: native'in maliyetine katlanmadan, native'e çok yakın bir deneyim sunar. **React Native** özellikle bu seride öğrendiğin JavaScript ve React bilgisini doğrudan kullandığı için web geliştiriciler için en doğal geçiştir — "bildiğin dille mobil uygulama" demektir. **Flutter** ise Dart diliyle, çok akıcı ve tutarlı arayüzler üretmesiyle öne çıkar. İkisi de tek kod tabanından iki platforma çıktığı için **geliştirme ve bakım maliyetini yarıya yakın** düşürür. Sınırı şudur: çok özel/derin donanım entegrasyonları veya en uç performans gerektiğinde bazen native koda inmen gerekebilir — ama çoğu uygulama (sosyal, içerik, e-ticaret, araçlar) bu sınıra hiç ulaşmaz. Web becerisinden mobile geçişin en mantıklı köprüsü genelde budur.

Telefonda nasıl görünür? MOBİL

Çapraz platform bir araçla (React Native veya Flutter) tek bir kod tabanı yazarsın; araç bundan hem iOS hem Android uygulamasını üretir. Kullanıcı, her iki platformda da native'e çok yakın, akıcı bir deneyim yaşar — ama sen iki yerine tek kod yazıp sürdürürsün. Bu, çoğu uygulama için maliyet ve performans arasındaki en dengeli noktadır.

Alıştırma 10 dk

Çaprazı kavra:

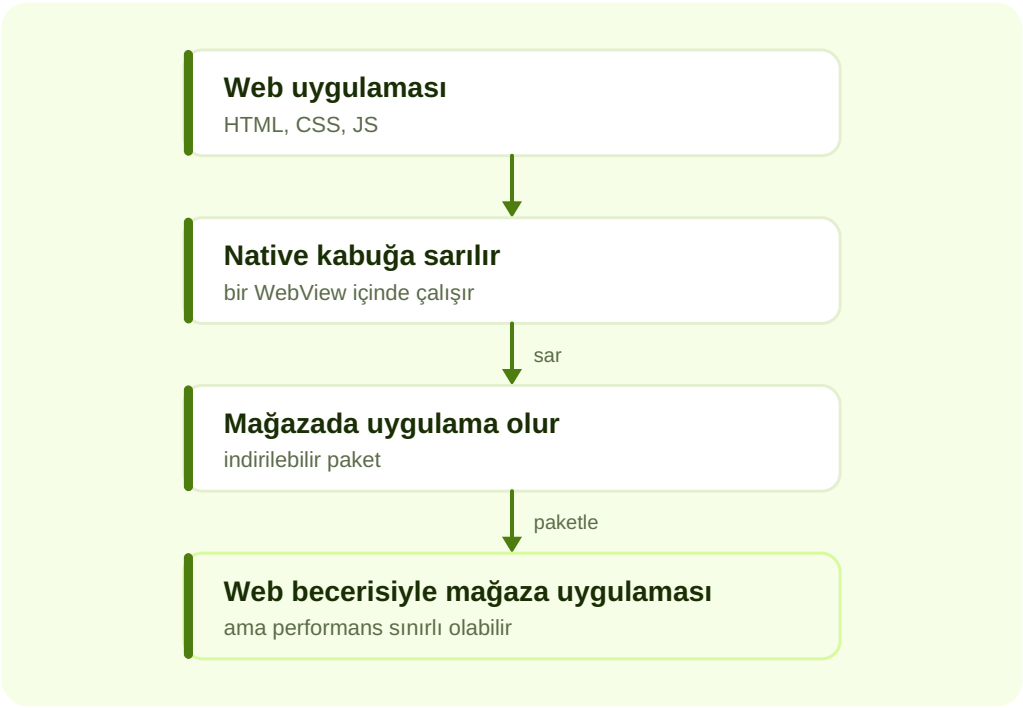
1. Çapraz platform yaklaşımının native'e göre temel avantajını yaz.
2. Web geliştiriciler için neden React Native doğal bir geçiştir, açıkla.
3. Çapraz platformun sınırının nerede olduğunu belirt.

BÖLÜM 18

Hibrit Uygulamalar

Hibrit uygulama, bir web uygulamasının (HTML/CSS/JS) native bir kabuğun içine sarılarak mağazada uygulama olarak sunulmasıdır. Web becerinle yaparsın, mağazada uygulama olarak görünür. Çapraz platformla PWA arasında bir köprüdür.

Web'i kabuğa sarmak



Şema 18.1 — Hibrit: web uygulamasını native kabuğa sarıp mağazaya koymak.

İPUCU

Hibrit yaklaşım (örneğin Capacitor/Cordova gibi araçlarla), mevcut web becerini ve hatta mevcut web uygulamanı **yeniden kullanarak** bir mağaza uygulaması çıkarmanı sağlar: uygulama, bir **WebView** (gömülü tarayıcı penceresi) içinde çalışır ama dışarıdan native bir uygulama gibi görünür ve mağazada yer alır. Avantajı, **tek web kod tabanı** ve mağaza erişimidir. Dezavantajı, performansın ve "his"in genelde native veya iyi bir çapraz platform çözümünün gerisinde kalmasıdır — çünkü altında bir web sayfası çalışır. Bugün çoğu durumda, mağaza gerekmiyorsa **PWA**, gerekiyorsa **çapraz platform** daha iyi sonuç verir; hibrit ise mevcut bir web uygulamasını hızlıca mağazaya taşımak isteyen ekipler için pratik bir ara çözüm olarak yerini korur.

Telefonda nasıl görünür? MOBİL

Hibrit bir uygulamada, yazdığın web uygulaması bir native kabuğun (WebView'in) içine yerleştirilir ve mağazaya yüklenebilen bir paket hâline gelir. Kullanıcı bunu mağazadan indirir ve bir uygulama gibi kullanır — ama içeride aslında web çalışır. Web becerini mağaza uygulamasına dönüştürmenin hızlı bir yoludur; performans en üst düzey olmasa da birçok uygulama için yeterlidir.

Alıştırma 10 dk

Hibridi kavra:

1. Hibrit uygulamanın temel fikrini ("web + kabuk") kendi cümlenle açıkla.
2. Hibridin avantajını ve dezavantajını yaz.
3. PWA, hibrit ve çapraz platformu kısaca sırala (en web'den en native'e).

BÖLÜM 19

Hangi Yolu Seçmeli?

Doğru yaklaşımı seçmek, birkaç pratik soruyu yanıtlamakla başlar: Mağazada olmak şart mı? Performans ne kadar kritik? Ekibin hangi becerilere sahip? Bütçe ne? Bu sorular, seni native, çapraz platform, hibrit veya PWA'ya yönlendirir.

Karar soruları

Şema 19.1 — Yol seçimi: birkaç soru seni doğru yaklaşıma götürür.

İPUCU

Karar verirken pragmatik ol ve **en basit yeterli yoldan** başla. **Mağaza görünürlüğü şart değilse** ve web becerin varsa, PWA çoğu zaman en hızlı, en ucuz ve bakımı en kolay yoldur — bu seride yaptığın işlerle zaten bu yola hâkimsin. **Mağazada olmak gerekiyorsa** ve tek kodla iki platform istiyorsan, çapraz platform (özellikle React Native, mevcut JS bilginle) güçlü bir seçimdir. **En üst performans veya derin donanım** kritikse native'e geç. **Elinde çalışan bir web uygulaması varsa** ve hızla mağazaya taşımak istiyorsan hibrit pratik olabilir. Yaygın hata, "herkes native yapıyor" diye en pahalı yola gereksiz yere girmektir. Ekip becerisi, bütçe ve gerçek ihtiyaçlar kararı belirlemeli — moda değil. İlerde yol değiştirmek de mümkündür: birçok ürün PWA olarak başlayıp ihtiyaç doğunca büyür.

Telefonda nasıl görünür?

MOBİL

Sorulara verdiğin yanıtlar seni doğru yola götürür: mağaza gerekmiyorsa ve web becerin varsa PWA; tek kodla iki platform istiyorsan çapraz platform; en üst performans/donanım gerekiyorsa native; mevcut web'i hızla mağazaya taşıyacaksan hibrit. Doğru seçim; ihtiyaç, ekip ve bütçeyle örtüşen, gereksiz karmaşık olmayan seçimdir.

Alıştırma

12 dk

Yolunu seç:

1. Bir uygulama fikri hayal et; mağaza, performans ve ekip ihtiyacını yaz.
2. Bu ihtiyaçlara göre hangi yolu seçerdin, gerekçelendir.
3. "En basit yeterli yoldan başla" ilkesini kendi cümlenle açıkla.

BÖLÜM 20

Mobilde API ve Veri

Hangi yolu seçersen seç, çoğu mobil uygulama verisini bir sunucudan alır: kullanıcı bilgileri, içerik, listeler. Bu, Modül 7'deki (Backend & API) ve Modül 8'deki (Veritabanı) bilginin doğrudan mobil karşılığıdır. Uygulama bir istemci, sunucu bir API olur.

Uygulama ↔ sunucu

Şema 20.1 — Mobil uygulama, API üzerinden sunucudan veri alır (Modül 7).

İPUCU

Mobil uygulamalar genelde verinin kendisini taşımaz; bir **API** üzerinden sunucudan ister (tıpkı Modül 7'de gördüğün gibi — istek gönder, JSON yanıt al). Bu yüzden backend ve API bilgin, mobilde doğrudan işe yarar: aynı API'yi web siten ve mobil uygulaman **paylaşabilir**. Mobilde bazı ek konulara dikkat edersin: **ağ her zaman olmayabilir** (çevrimdışı durumu ele al, veriyi yerelde önbellekle), **bağlantı yavaş olabilir** (az ve gerektiği kadar veri çek), ve **güvenlik kritiktir** (Modül 7-12'deki ilkeler: HTTPS kullan, kimlik doğrulama belirteçlerini güvenli sakla, API anahtarlarını uygulamaya gömme). Kullanıcı verisini işlerken yine **KVKK** ve en az veri ilkesi geçerlidir. Kısacası: mobil uygulama bir "ön yüz"dür; gerçek veri ve mantık çoğu zaman tanıdık backend dünyasında yaşar.

Telefonda nasıl görünür? MOBİL

Bir mobil uygulama veri gösterdiğinde (ürün listesi, mesajlar, profil), bunu genelde bir API'den ister: uygulama bir istek gönderir, sunucu veritabanından veriyi alıp JSON olarak döner, uygulama da bunu ekrana çizer. Web ile mobil aynı API'yi paylaşabilir. Mobilde ek olarak çevrimdışı durumu, yavaş ağı ve güvenliği düşünmek gerekir — ama temel mantık Modül 7'dekiyle aynıdır.

Alıştırma 10 dk

Veriyi bağla:

1. Bir mobil uygulamanın API'den veri alma akışını adım adım yaz.
2. Web ile mobilin aynı API'yi paylaşabilmesinin avantajını açıkla.
3. Mobilde API kullanırken dikkat edilecek üç ek konuyu belirt.

SEVİYE 4

Yayınlama ve Kalite

Uygulamayı dünyaya açmak ve iyi tutmak: uygulama mağazaları, mobil UX ilkeleri, erişilebilirlik, güvenlik ve gizlilik, test ve mobil-hazır bir proje.

BÖLÜM 21

Uygulama Mağazaları

Native, çapraz platform ve hibrit uygulamalar genelde App Store (iOS) ve Google Play (Android) üzerinden dağıtılır. Mağazalar erişim ve güven sağlar ama kendi kuralları, inceleme süreçleri ve ücretleri vardır. Yayınlamadan önce bunları bilmek gerekir.

Mağazaya giden yol

Şema 21.1 — Mağaza süreci: geliştir → gönder → incele → yayında.

- **Geliştirici hesabı:** genelde yıllık/tek seferlik ücret gerekir.
- **İnceleme:** uygulama mağaza kurallarına göre denetlenir.
- **Mağaza kesintisi:** ödemelerden bir pay alınır.

İPUCU

Mağazalar büyük bir **erişim ve güven** sağlar (milyonlarca kullanıcı, güvenli ödeme, otomatik güncelleme) ama bedelleri vardır: **geliştirici hesabı ücreti**, **inceleme süreci** (uygulaman kurallara uymazsa reddedilebilir — gizlilik, içerik, işlevsellik kuralları katıdır) ve **ödeme kesintisi** (mağaza, dijital satışlardan bir yüzde alır). İnceleme reddi yaygındır; mağaza kurallarını önceden okumak zaman kazandırır. İyi bir mağaza listesi için **net ikon, çekici ekran görüntüleri, açık bir açıklama** ve dürüst meta veriler önemlidir — bunlar indirme oranını etkiler. Hatırla: **PWA bu sürecin tamamını atlar** (Seviye 2) — mağaza görünürlüğüne ihtiyacın yoksa, doğrudan web'den dağıtmak çok daha hızlıdır. Mağaza, görünürlük ve güven gerçekten gerektiğinde değerlidir.

Telefonda nasıl görünür?

MOBİL

Bir uygulamayı mağazaya almak için onu paketler, bir geliştirici hesabıyla gönderir, ikon/ekran görüntüsü/açıklama eklersin; mağaza uygulamayı kurallarına göre inceler ve onaylarsa yayınlar. Kullanıcılar artık onu mağazadan arayıp indirebilir. Bu süreç görünürlük ve güven kazandırır; karşılığında ücret, inceleme ve kesinti getirir — PWA ise bu adımların hiçbirini gerektirmez.

Alıştırma

10 dk

Mağazayı kavra:

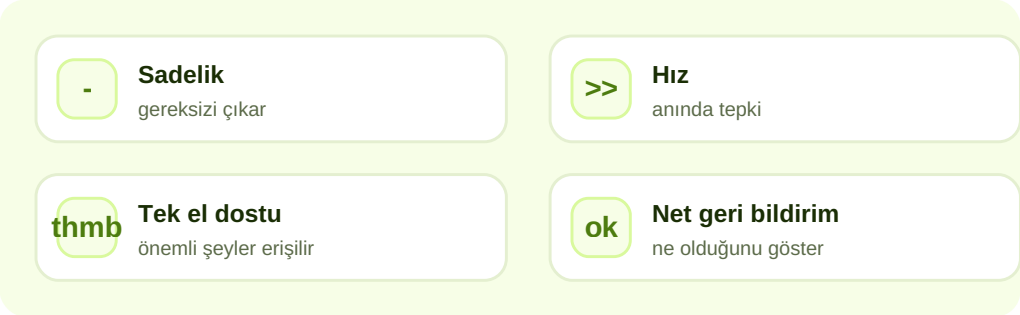
1. Bir uygulamanın mağazaya alınma sürecini adım adım yaz.
2. Mağazaların sağladığı iki fayda ve getirdiği iki maliyet nedir, yaz.
3. Hangi durumda mağaza yerine PWA tercih edilebilir, açıkla.

BÖLÜM 22

Mobil UX İlkeleri

İyi bir mobil deneyim (UX), küçük ekranın ve hareket hâlindeki kullanıcının gerçeğine saygı duyar: sade, hızlı, parmakla kolay ve dikkat dağıtmayan. Mobilde kullanıcı genelde aceleci, tek elli ve bölünmüş dikkatlidir; tasarım buna göre olmalı.

İyi mobil UX'in temelleri



Şema 22.1 — İyi mobil UX'in dört temel ilkesi.

İPUCU

Mobilde UX, "az ama öz"dür. **Sadelik:** her ekranda tek bir ana eyleme odaklan; gereksiz seçenek ve dağınıklık küçük ekranda boğucudur. **Hız:** kullanıcı dokununca anında tepki ver, ağ beklerken bir gösterge göster (boş ekran "dondu mu?" hissi verir). **Tek el dostu:** önemli butonları başparmağın kolayca eriştiği yere (genelde alt kısma) koy; ekranın en üst köşeleri tek elle zor erişilir. **Net geri bildirim:** bir işlem başarılı/başarısız olduğunda kullanıcıya açıkça söyle. **Yerleşik kalıplara sadık kal:** kullanıcılar alışık oldukları davranışları bekler; "yaratıcı" ama beklenmedik arayüzler kafa karıştırır. İyi mobil UX, kullanıcının düşünmeden, akıcı biçimde hedefe ulaşmasını sağlar — bu da Modül 1'den beri vurgulanan "kullanıcı için tasarla" ilkesinin mobil hâlidir.

Telefonda nasıl görünür? MOBİL

İyi tasarlanmış bir mobil arayüzde kullanıcı ne yapacağını anında anlar: ekran sade, ana eylem belirgin, butonlar başparmakla rahat erişilir ve her dokunuş net bir tepki verir. Aceleci ve dikkati bölünmüş bir kullanıcı bile hedefine zorlanmadan ulaşır. Sonuç: insanların kullanmaktan keyif aldığı, terk etmediği bir uygulama.

Alıştırma 10 dk

UX'i uygula:

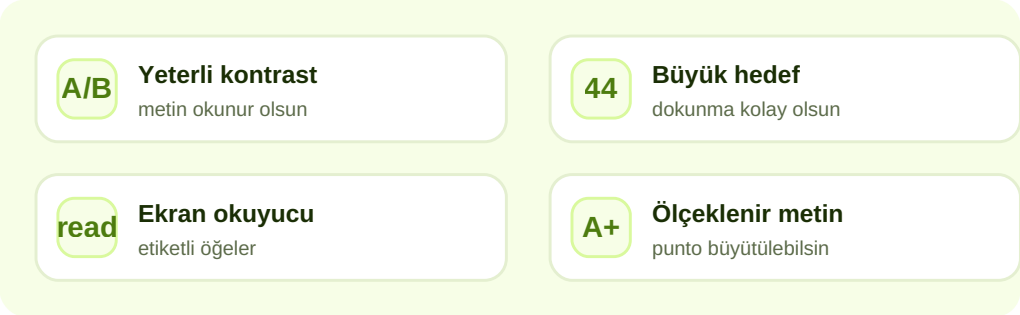
1. İyi mobil UX'in dört ilkesini ve birer örneğini yaz.
2. Önemli butonları neden ekranın altına koymak iyi olur, açıkla.
3. Yerleşik kalıplara sadık kalmanın önemini kendi cümlenle belirt.

BÖLÜM 23

Erişilebilirlik (Mobilde)

Erişilebilirlik, uygulamanın herkes tarafından — görme, işitme, hareket veya başka bir zorluğu olanlar dahil — kullanılabilmesidir. Mobilde bu daha da önemlidir, çünkü insanlar uygulamalara her ortamda ve her koşulda erişir. Erişilebilir tasarım, herkes için daha iyi tasarımdır.

Herkes için kullanılabilir



Şema 23.1 — Mobil erişilebilirliğin dört temel ölçütü.

İPUCU

Erişilebilirlik bir "ek özellik" değil, **temel bir kalite ölçütüdür** — ve çoğu önlem herkesin deneyimini iyileştirir. **Kontrast:** metin ile arka plan arasında yeterli fark, hem görme zorluğu olanlar hem de güneşte telefon kullanan herkes için okumayı sağlar. **Dokunma hedefleri:** büyük butonlar, motor güçlüğü olanlar dahil herkese yardımcı olur. **Ekran okuyucu desteği:** görme engelli kullanıcılar arayüzü sesli olarak kullanır — bunun için butonların ve görsellerin anlamlı **etiketleri** (alt metin, erişilebilir ad) olmalı. **Ölçeklenebilir metin:** kullanıcı sistem yazı boyutunu büyüttüğünde uygulamanın bozulmaması gerekir. Bu önlemler hem etik bir sorumluluktur hem de birçok yerde **yasal bir gerekliliktir**; ayrıca uygulamanın kitlesini genişletir. "Herkes için tasarla" — erişilebilirlik, iyi tasarımın ayrılmaz parçasıdır.

Telefonda nasıl görünür? MOBİL

Erişilebilir bir mobil uygulamada, metinler yeterli kontrastla okunur, butonlar herkesin rahatça dokunabileceği büyüklüktedir, ekran okuyucu kullanan biri öğeleri sesli olarak anlayabilir ve yazı boyutu büyütüldüğünde arayüz bozulmaz. Bu sayede görme, hareket veya başka bir zorluğu olan kullanıcılar da uygulamayı tam olarak kullanabilir. Erişilebilir tasarım, kullanıcı kitlesini genişletir ve deneyimi herkes için iyileştirir.

Alıştırma

10 dk

Erişilebilir yap:

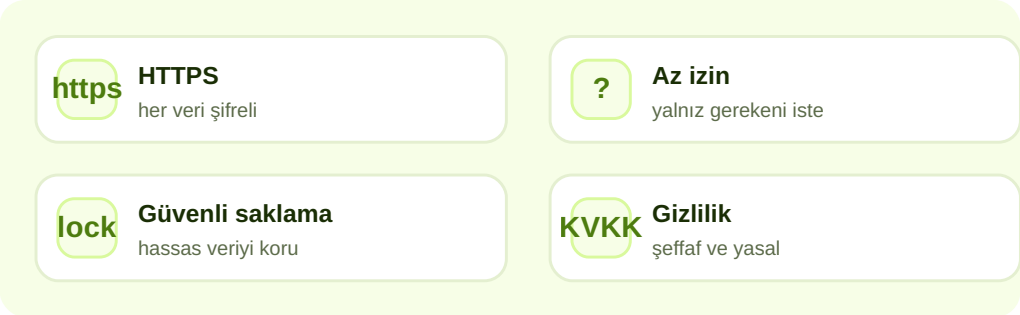
1. Mobil erişilebilirliğin dört ölçütünü ve kime yardımcı olduğunu yaz.
2. Ekran okuyucu için öğelerin neden etiketli olması gerektiğini açıkla.
3. Erişilebilirliğin "herkes için daha iyi tasarım" olduğunu bir örnekle göster.

BÖLÜM 24

Mobil Güvenlik ve Gizlilik

Mobil cihazlar çok kişisel veri taşır: konum, kişiler, fotoğraflar, sağlık. Bu yüzden mobilde güvenlik ve gizlilik özellikle hassastır. Tüm seri boyunca öğrendiğin ilkeler burada da geçerlidir — artı izinler ve KVKK gibi mobil-özel konular.

Güvenli ve saygılı uygulama



Şema 24.1 — Mobil güvenlik ve gizliliğin dört temel taşı.

- **HTTPS:** tüm ağ trafiği şifreli olmalı (Modül 12).
- **En az izin:** yalnızca gerçekten gereken izinleri iste.
- **Güvenli saklama:** belirteç/şifre gibi hassas veriyi koru.
- **Şeffaflık:** hangi veriyi neden topladığını açıkla (KVKK).

İPUCU

Mobil güvenlik, tüm serinin güvenlik ilkelerinin kişisel veri açısından en hassas hâlidir. **HTTPS** şarttır (Modül 12) — uygulama ile sunucu arasındaki her şey şifreli olmalı. **İzinlerde en az ilkesi** kritiktir: konum, kişiler, kamera gibi izinleri yalnızca **gerçekten gerektiğinde** ve **nedenini açıklayarak** iste; gereksiz izin hem kullanıcıyı ürkütür hem gizlilik riski yaratır. **Hassas verileri** (giriş belirteçleri, şifreler) cihazın güvenli depolama alanlarında tut, asla düz metin saklama; API anahtarlarını uygulamaya gömme (kolayca çıkarılabilir). **KVKK ve gizlilik:** hangi veriyi neden topladığını şeffaf biçimde açıkla, gereğinden fazla veri toplama (veri minimizasyonu), kullanıcıya kontrol ver. Bu hem yasal bir zorunluluk hem de kullanıcı güveninin temelidir. Mobilde gizliliği ciddiye almak, artık bir rekabet avantajıdır — kullanıcılar buna değer veriyor.

Telefonda nasıl görünür? MOBİL

Güvenli ve saygılı bir mobil uygulama: tüm trafiği HTTPS ile şifreler, yalnızca gerçekten gereken izinleri (nedenini açıklayarak) ister, hassas verileri güvenli biçimde saklar ve hangi veriyi neden topladığını şeffafça (KVKK'ya uygun) bildirir. Böylece kullanıcının en kişisel cihazındaki verisi korunur ve güveni kazanılır. Güvenlik ve gizlilik, mobilde lüks değil, temel sorumluluktur.

Alıştırma 12 dk

Güvenli tasarla:

1. Mobil güvenlik ve gizliliğin dört temel önlemini yaz.
2. "En az izin" ilkesinin neden önemli olduğunu açıkla.
3. KVKK açısından bir uygulamanın kullanıcıya karşı sorumluluğunu kendi cümlenle belirt.

BÖLÜM 25

Test ve Cihaz Uyumluluğu

Mobil dünyada sayısız cihaz, ekran boyutu ve işletim sistemi sürümü vardır. Senin telefonunda mükemmel çalışan bir uygulama, başka bir cihazda bozuk görünebilir. Bu yüzden farklı koşullarda test etmek, kaliteli bir mobil deneyimin vazgeçilmez parçasıdır.

Farklı koşullarda dene

Şema 25.1 — Mobil test: farklı ekran, OS, cihaz ve ağ koşulları.

İPUCU

Mobil **parçalanma** (fragmentation) gerçektir: yüzlerce farklı ekran boyutu, çözünürlük ve işletim sistemi sürümü vardır. "Benim telefonumda çalışıyor" yeterli değildir. **Tarayıcı geliştirici araçları**, farklı ekran boyutlarını masaüstünde taklit etmenin (responsive mod) en hızlı yoludur — duyarlı tasarımını anında denersin. Ama **gerçek cihazlarda** da test et: gerçek dokunma hassasiyeti, performans ve "his" ancak gerçek bir telefonda anlaşılır. **Emülatörler/simülatörler** elinde olmayan cihazları taklit etmeye yarar. **Kötü koşulları** unutma: yavaş ağda, düşük pilde, çevrimdışında uygulaman ne yapıyor? **Erişilebilirlik testini** de ekle (ekran okuyucuyla dene). Kapsamlı test imkânsız olsa da, en yaygın cihaz/boyut/OS kombinasyonlarını ve kötü senaryoları denemek, kullanıcıların karşılaşacağı sorunların çoğunu önceden yakalar.

Telefonda nasıl görünür?

MOBİL

Bir uygulamayı test ederken onu farklı ekran boyutlarında (küçük telefondan tablete), farklı işletim sistemlerinde (iOS ve Android, eski ve yeni sürümler), gerçek cihazlarda ve emülatörlerde, ayrıca yavaş ağ ve çevrimdışı gibi kötü koşullarda denersin. Böylece yalnızca kendi cihazında değil, kullanıcıların elindeki çeşitli cihazlarda da düzgün çalıştığından emin olursun. Test, sürprizleri kullanıcı yaşamadan önce sana yaşatır.

Alıştırma

10 dk

Test et:

1. Bir mobil uygulamayı hangi farklı koşullarda test etmek gerekir, listele.
2. Gerçek cihazda test etmenin emülatöre göre avantajını yaz.
3. "Mobil parçalanma" ne demek, kendi cümlenle açıkla.

BÖLÜM 26

Bitirme: Mobil-Hazır Bir Proje

Öğrendiklerini birleştirip mobil-hazır bir proje tasarlıyorsun: duyarlı, hızlı, dokunma dostu, erişilebilir ve (istersen) kurulabilir. Bu kontrol listesi, bir web projesini "telefonda da çalışan"dan "mobilde mükemmel"e taşıyan adımların özetidir.

Mobil-hazır kontrol listesi

Şema 26.1 — Mobil-hazır bir uygulamanın bileşenleri tek ekranda.

Mobil-hazır kontrol listesi

```
 [ ] Viewport meta etiketi var
[ ] Mobil-öncelikli, duyarlı düzen (medya sorguları)
[ ] Görseller esnek ve optimize (max-width, srcset)
[ ] Gövde metni >= 16px, yeterli kontrast
[ ] Dokunma hedefleri >= ~44px, yeterli boşluk
[ ] Mobilde hızlı yükleniyor (hafif, önbellekli)
[ ] Erişilebilir (ekran okuyucu, ölçeklenir metin)
[ ] HTTPS aktif; izinler az ve açıklamalı
[ ] (İsteğe bağlı) PWA: manifest + service worker
```

İPUCU

Bu kontrol listesi, modülün tamamının özüdür; her mobil proje öncesi geç. Bu modülle birlikte **FAZ 3'ün ilk adımını** attın: artık yalnızca web yapmıyor, onu **mobilin gerçeğine** göre tasarlıyorsun — küçük ekran, dokunmatik, yavaş ağ, çeşitli cihaz. Önemli bir kavrayış: **mobil ayrı bir dünya değil, web becerinin bir uzantısıdır**. HTML/CSS/JS (Modül 3-4-5), duyarlı tasarımla mobile taşınır; PWA ile kurulabilir uygulamaya dönüşür; ve gerektiğinde çapraz platform/native ile gerçek mobil uygulamalara köprü kurulur. Çoğu fikir için, bu seride zaten öğrendiğin web becerisi + duyarlı tasarım + PWA fazlasıyla yeterlidir. Bir sonraki modülde (Yayınlama), projeleri dünyaya açmanın ve sürdürmenin yollarını derinleştireceğiz. Mobil-öncelikli düşünmeyi alışkanlık hâline getir — kullanıcıların çoğu zaten orada.

Telefonda nasıl görünür? MOBİL

Mobil-hazır bir proje: viewport etiketiyle başlar, mobil-öncelikli ve duyarlı düzeniyle her ekrana uyar, esnek/optimize görselleri ve okunur metniyle hızlı ve rahat okunur, dokunma dostu ve erişilebilirdir, HTTPS ile güvenlidir ve istersen PWA olarak kurulabilir. Kontrol listesini geçtiğinde, projen yalnızca telefonda "açılan" değil, mobilde gerçekten iyi çalışan bir deneyim olur — ki kullanıcıların çoğu tam da orada.

Alıştırma 20 dk

Mobil-hazır tasarla:

1. Bir web projesi seç; mobil-hazır kontrol listesini onun için doldur.
2. Her maddenin neden önemli olduğunu bir cümleyle yaz.
3. Projeni PWA yapmak ister miydin, neden? Kararını gerekçelendir.
4. Hangi yolla (PWA/çapraz/native) ilerlerdin ve neden, belirt.

EK

Mobil Terimleri Sözlüğü

En sık kullanılan mobil geliştirme terimleri. Bir başvuru kaynağı olarak saklayabilirsin.

Mobil-önce	`Küçük ekrandan başla`	Viewport	`width=device-width`
Responsive	`Tek site, her ekran`	@media	`Kırılma noktası kuralı`
min-width	`Mobil-öncelikli sorgu`	Flexbox / Grid	`Esnek düzen araçları`
auto-fit / minmax	`Otomatik duyarlı grid`	max-width: 100%	`Esnek görsel`
~44px	`Min. dokunma hedefi`	PWA	`Kurulabilir web uygulaması`
Manifest	`PWA kimlik dosyası`	Service Worker	`Çevrimdışı / önbellek`
Native / Çapraz	`Uygulama yolları`	srcset	`Ekrana göre görsel`

Mobil özet

MOBİL

Mobil-hazır olmak bir zincirdir: **viewport** etiketiyle başlar, **mobil-öncelikli** ve **duyarlı** tasarımla (medya sorguları, Flexbox/Grid) her ekrana uyum sağlar, **esnek görseller** ve **okunur tipografi** ile rahat okunur, **dokunma dostu** hedeflerle parmakla kullanılır ve **hafif/hızlı** olarak yavaş bağlantılarda bile açılır. Bunun ötesinde, web'i **PWA**'ya dönüştürüp kurulabilir ve çevrimdışı yapabilir; ya da **native**, **çapraz platform** veya **hibrit** yollarıyla gerçek mobil uygulamalara geçebilirsin. Hangi yolu seçersen seç, temel aynı: kullanıcının çoğu telefonda; deneyimi önce orada mükemmelleştir.