WEB & YAZILIM GELİŞTİRME SERİSİ · MODÜL 9

PHP

Sunucu tarafı programlama: sözdizimi, değişkenler, koşullar, döngüler, diziler ve fonksiyonlar; PHP ve HTML, formlar ve doğrulama; PDO ile veritabanı, hazırlanmış ifadeler ve oturumlar; OOP, güvenlik ve dağıtım. Özgün diyagramlar ve gerçek PHP örnekleriyle.

Sözdizimi · Form · PDO · Güvenlik · Eğitim amaçlıdır

Bu Kitap Hakkında

Bu modül, web'in en yaygın sunucu dillerinden biri olan PHP'yi sıfırdan öğretir ve Backend ile Veritabanı modüllerinde öğrenilen kavramları gerçek koda döker. Dört seviye ve yirmi altı bölüm boyunca echo, değişkenler, operatörler, koşullar, döngüler, diziler ve fonksiyonlardan; PHP'yi HTML'e gömmeye, formlara (GET/POST), doğrulamaya ve süper globallere; PDO ile veritabanına, hazırlanmış ifadelere, oturumlara ve çerezlere; nesne yönelimli programlamaya, Composer'a, güvenliğe, API uç noktalarına ve dağıtıma kadar uzanır.

Her bölümde konuyu görselleştiren özgün bir diyagram (kod → tarayıcı çıktısı panelleri, akış şemaları, sınıf kutuları), çalıştırılabilir PHP örnekleri, 'tarayıcıda ne görünür?' kartı ve bir alıştırma yer alır. Güvenlik baştan sona vurgulanır: kullanıcı verisini doğrulama ve kaçışlama (XSS), SQL enjeksiyonuna karşı hazırlanmış ifadeler (Modül 8'in PHP karşılığı), oturum güvenliği, password_hash ile şifre saklama, CSRF ve HTTPS. Bu, on altı modüllük 'Web & Yazılım Geliştirme' serisinin dokuzuncu modülüdür; Python ve C# modülleri aynı kavramları farklı sözdizimiyle ele alır. Bu seri eğitim amaçlıdır.

Web & Yazılım Geliştirme Serisi · Modül 9

İçindekiler

SEVİYE 1

PHP'ye Giriş

Temeller: PHP nedir, echo ile ilk kod, değişkenler ve veri tipleri, operatörler, koşullar, döngüler, diziler ve fonksiyonlar.

BÖLÜM 01

PHP Nedir?

PHP, web için tasarlanmış bir sunucu tarafı programlama dilidir. Tarayıcıda değil, sunucuda çalışır: bir istek geldiğinde PHP kodu işlenir ve sonuç (genelde HTML) tarayıcıya gönderilir. Backend ve Veritabanı modüllerinde öğrendiğin kavramların somut bir dili budur.

PHP nerede çalışır?

- PHP kodu **sunucuda** çalışır; tarayıcı yalnızca sonucu (HTML) görür.
- Dosyalar `.php` uzantılıdır.
- PHP, HTML'in içine gömülebilir — sayfayı dinamik kılar.

Şema 1.1 — PHP sunucuda çalışır; tarayıcıya yalnızca ürettiği HTML gider.

İPUCU

Önemli bir ayrım: tarayıcıda **"sayfa kaynağını görüntüle"** dediğinde PHP kodunu **göremezsin** — yalnızca PHP'nin ürettiği HTML'i görürsün. Çünkü PHP sunucuda çalışıp biter; tarayıcıya kod değil, sonuç gider. Bu, JavaScript'ten (tarayıcıda çalışır, kaynağı görünür) temel farkıdır. PHP web'in en yaygın dillerinden biridir; WordPress dahil sayısız site PHP ile çalışır.

Tarayıcıda ne görünür? ÇIKTI

Kullanıcı bir `.php` sayfasını açtığında, sunucu PHP kodunu çalıştırır ve sonucu HTML olarak döner. Kullanıcı tarayıcıda yalnızca bu HTML çıktısını görür — arkadaki PHP mantığını değil. PHP, "perde arkasındaki" dildir.

Alıştırma 8 dk

PHP'yi konumla:

1. PHP'nin nerede (sunucu/tarayıcı) çalıştığını kendi cümlenle yaz.
2. Kullanıcının neden PHP kodunu göremediğini açıkla.
3. PHP ile JavaScript'in çalışma yeri açısından farkını belirt.

BÖLÜM 02

İlk PHP Kodu: echo

PHP kodu ile biter. En temel komut echo'dur: ekrana (tarayıcıya) bir şey yazdırır. Bu, her dilde olduğu gibi PHP'de de ilk adımdır.

echo ile çıktı

- PHP bloğu: `<?php ... ?>`.
- `echo` — metni tarayıcıya yazdırır.
- Her ifade **noktalı virgülle (;)** biter.

Şema 2.1 — echo, sunucuda çalışır ve tarayıcıya metni yazdırır.

İlk PHP kodun

```
<?php
  echo "Merhaba, dünya!";
  echo "<br>";          // HTML de yazdırabilirsin
  echo "PHP öğreniyorum.";
?>
```

İPUCU

PHP ifadelerinin sonundaki **noktalı virgül (;) zorunludur** — en sık başlangıç hatası onu unutmaktır. `echo` ile düz metin yazabileceğin gibi HTML etiketleri de (`<br>` , `<b>`) yazabilirsin; çünkü PHP'nin çıktısı tarayıcıya HTML olarak gider. Çift tırnak (`"..."`) ile tek tırnak (`'...'`) arasında ince bir fark vardır (çift tırnak değişken çözer) — birazdan göreceğiz.

Tarayıcıda ne görünür? ÇIKTI

`echo "Merhaba, dünya!";` satırı tarayıcıda "Merhaba, dünya!" yazısını gösterir. Birden çok echo art arda çalışır; `echo "<br>"` ile arada satır atlatabilirsin. Tarayıcı, PHP'nin ürettiği bu metni/HTML'i görür.

Alıştırma

10 dk

echo yaz:

1. Kendini tanıtan üç satırlık bir PHP kodu yaz (echo ile).
2. Satırlar arasına `<br>` ekle.
3. Noktalı virgülü unutursan ne olur, araştır/yaz.

BÖLÜM 03

Değişkenler ve Veri Tipleri

Değişken, bir değeri saklayan adlandırılmış bir kutudur. PHP'de değişkenler $ işaretiyle başlar. PHP "gevşek tipli"dir: bir değişkenin tipini sen belirtmezsin, değer belirler.

Değişken ve tipler

- Değişken adı `$` ile başlar: `$ad`, `$yas`.
- Atama `=` ile yapılır: `$yas = 30;`.
- Tipler: metin (string), sayı (int/float), mantıksal (bool), dizi, null.

Şema 3.1 — Değişkenler değer saklar; çift tırnakta doğrudan kullanılabilir.

Değişkenler ve tipler

```
<?php
  $ad     = "Ayşe";      // string (metin)
  $yas    = 30;           // integer (tam sayı)
  $boy    = 1.68;         // float (ondalık)
  $aktif  = true;         // boolean
  echo "Ad: $ad, Yaş: $yas";
?>
```

İPUCU

PHP'de **çift tırnaklı** metinlerde değişkenler doğrudan çözülür: `"Merhaba $ad"` → "Merhaba Ayşe". **Tek tırnakta** ise çözülmez: `'Merhaba $ad'` → "Merhaba $ad" (harfi harfine). Değişken adları büyük/küçük harfe duyarlıdır (`$ad` ≠ `$Ad`) ve anlamlı isimler seçmek (Modül 5'teki gibi) kodu okunur kılar. PHP gevşek tipli olduğundan tip belirtmen gerekmez ama tipleri bilmek önemlidir.

Tarayıcıda ne görünür? ÇIKTI

`$ad = "Ayşe"; $yas = 30;` tanımlandıktan sonra `echo "$ad, $yas yaşında"` tarayıcıda "Ayşe, 30 yaşında" yazdırır — çift tırnak değişkenleri değerleriyle değiştirir. Değişkenler, verini programın akışında taşımanın yoludur.

Alıştırma 10 dk

Değişken kullan:

1. Ad, şehir ve yaş için üç değişken tanımla ve echo ile yazdır.
2. Çift tırnak ile tek tırnağın değişken çözmedeki farkını dene.
3. Bir sayısal değişkenle basit bir işlem yap (örn. yaş + 1).

BÖLÜM 04

Operatörler ve İfadeler

Operatörler, değerler üzerinde işlem yapmanı sağlar: toplama, karşılaştırma, metin birleştirme. PHP'de metin birleştirme için özel bir operatör vardır: nokta (.).

Temel operatörler

- Aritmetik: `+ - * / %`.
- Metin birleştirme: **nokta (.)** — `$ad . " " . $soyad`.
- Karşılaştırma: `==` (eşit), `===` (tip dahil eşit), `!=`, `>`, `<`.
- Atama kısayolları: `+=`, `.=`.

Şema 4.1 — Aritmetik (*) ve metin birleştirme (.) bir arada.

Operatörler

```
<?php
  $a = 10;  $b = 3;
  echo $a + $b;          // 13
  echo $a % $b;          // 1  (kalan)
  echo "Ad: " . "Ayşe";  // metin birleştir (.)
  $sayac = 0;  $sayac += 5;   // $sayac artık 5
?>
```

İPUCU

PHP'de en sık karıştırılan nokta: metin birleştirme **+ ile değil, nokta (.) ile** yapılır (birçok dilden farklı olarak). `"Ad: " . $ad` doğru; `"Ad: " + $ad` beklenmedik sonuç verir. Eşitlikte de iki seçenek vardır: `==` değerleri karşılaştırır (tip dönüştürerek), `===` ise tip dahil tam eşitlik arar — kafa karışıklığını önlemek için genelde `===` tercih edilir.

Tarayıcıda ne görünür? ÇIKTI

`$fiyat * $adet` önce 300'ü hesaplar; `"Toplam: " . 300` ile birleşip "Toplam: 300" çıktısını verir. Nokta operatörü metinleri uç uca ekler; aritmetik operatörler sayılarla çalışır. İkisini birlikte kullanarak dinamik metinler üretirsin.

Alıştırma 10 dk

Operatör kullan:

1. İki sayının toplamını ve kalanını (%) yazdır.
2. Ad ve soyadı nokta (.) ile birleştirip yazdır.
3. == ile === arasındaki farkı bir örnekle göster.

BÖLÜM 05

Koşullar: if / else

Programların karar vermesini sağlayan yapı koşullardır. PHP'de if, bir koşul doğruysa bir bloğu çalıştırır; else ise yanlış olduğunda devreye girer. Bu, Algoritma modülündeki koşullu mantığın PHP karşılığıdır.

if / elseif / else

- `if (koşul) { ... }` — koşul doğruysa çalışır.
- `elseif (...) { ... }` — başka bir koşul.
- `else { ... }` — hiçbiri değilse.

Şema 5.1 — if/else: koşulun doğruluğuna göre farklı yol izlenir.

if / elseif / else

```php
<?php
  $not = 75;
  if ($not >= 85) {
    echo "Pekiyi";
  } elseif ($not >= 60) {
    echo "Geçer";
  } else {
    echo "Kaldı";
  }
?>
```

İPUCU

Koşulların sırası önemlidir: PHP ilk doğru olan dalı çalıştırıp gerisini atlar. Yukarıdaki örnekte `$not = 75` için önce `>= 85` (yanlış), sonra `>= 60` (doğru) → "Geçer" yazdırılır ve `else`'e bakılmaz. Mantıksal operatörlerle (`&&` ve, `||` veya) birden çok koşulu birleştirebilirsin. Süslü parantezleri `{ }` hep kullan — tek satırlık bloklar bile ileride hata kaynağıdır.

Tarayıcıda ne görünür? ÇIKTI

`$not = 75` için kod sırasıyla koşulları dener: 85'ten küçük (atla), 60'tan büyük (eşleşti) → tarayıcıda "Geçer" görünür. Koşulu değiştirirsen (örn. 90) farklı bir dal çalışır ve farklı çıktı üretilir.

Alıştırma 12 dk

Koşul yaz:

1. Bir sayının pozitif, negatif veya sıfır olduğunu yazdıran if/elseif/else yaz.
2. Bir kullanıcının yaşına göre "Yetişkin / Reşit değil" yazdır.
3. && (ve) kullanarak iki koşulu birleştir.

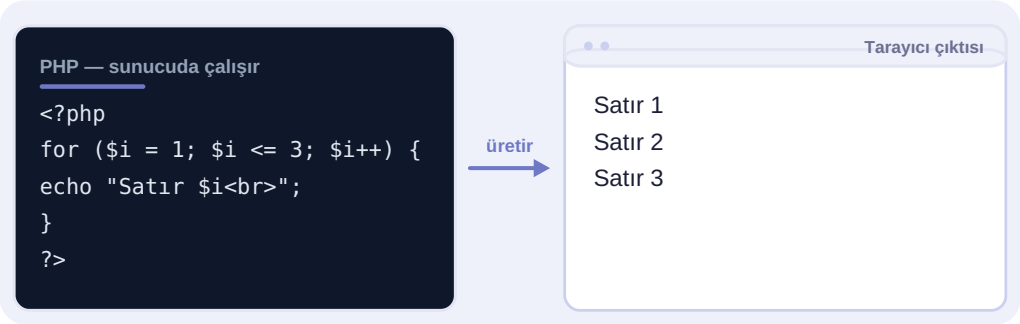
BÖLÜM 06

Döngüler: for / while / foreach

Döngüler, bir işi tekrar tekrar yapmanı sağlar. PHP'de üç temel döngü vardır: belirli sayıda tekrar için for, koşul sürdükçe çalışan while ve diziler üzerinde gezinen foreach.

Üç temel döngü

- `for` — sayaçla belirli tekrar (1'den 5'e kadar).
- `while` — koşul doğru olduğu sürece.
- `foreach` — bir dizinin her elemanı için.



Şema 6.1 — for döngüsü, gövdeyi belirli sayıda tekrar eder.

for, while ve foreach

```
<?php
  // for: 1'den 3'e
  for ($i = 1; $i <= 3; $i++) { echo $i; }

  // foreach: dizinin her elemanı
  $renkler = ["kırmızı", "yeşil", "mavi"];
  foreach ($renkler as $renk) { echo $renk . " "; }
?>
```

İPUCU

`foreach`, PHP'de dizilerle çalışmanın en doğal yoludur — Modül 5'teki forEach'in PHP karşılığıdır. Anahtarı da almak istersen: `foreach ($dizi as $anahtar => $deger)`. `for` döngüsünde en sık hata, koşulu yanlış kurup **sonsuz döngüye** girmektir (örn. sayacı artırmayı unutmak). while kullanırken koşulun bir gün yanlış olacağından emin ol — yoksa sayfa asla yüklenmez.

Tarayıcıda ne görünür? ÇIKTI

`for ($i = 1; $i <= 3; $i++)` döngüsü gövdesini üç kez çalıştırır; her turda `$i` bir artar ve "Satır 1", "Satır 2", "Satır 3" art arda yazdırılır. Döngü, tekrar eden işi tek bir blokla ifade etmenin yoludur.

Alıştırma 12 dk

Döngü yaz:

1. 1'den 10'a kadar sayıları yazdıran bir for döngüsü yaz.
2. Bir renk dizisini foreach ile dolaşıp her rengi yazdır.
3. Bir for döngüsünün neden sonsuz döngüye girebileceğini açıkla.

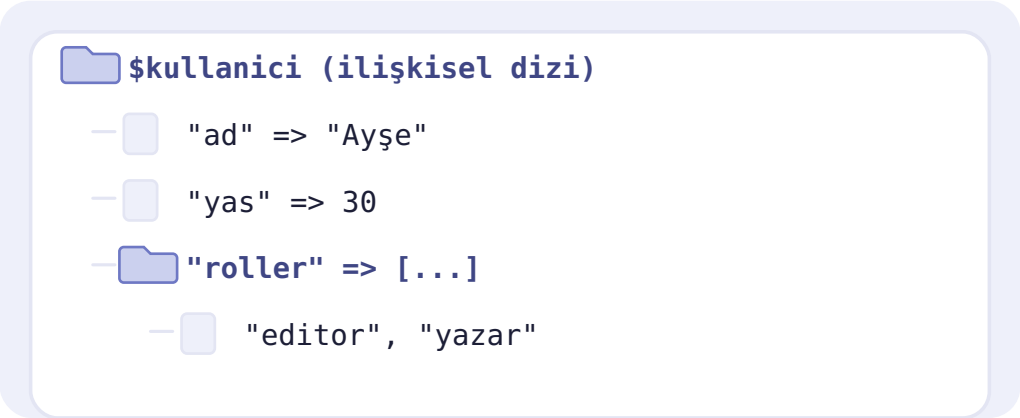
BÖLÜM 07

Diziler (Arrays)

Dizi, birden çok değeri tek bir değişkende tutmanı sağlar. PHP'de iki tür dizi vardır: sıralı (indeksli) diziler ve anahtar–değer çiftlerinden oluşan ilişkisel (associative) diziler.

İki tür dizi

- **İndeksli:** `["a", "b", "c"]` — 0'dan başlayan sayısal indeks.
- **İlişkisel:** `["ad" => "Ayşe", "yas" => 30]` — anahtarla erişim.
- Erişim: `$dizi[0]` veya `$dizi["ad"]`.



Şema 7.1 — İlişkisel dizi: anahtar–değer çiftleri, iç içe olabilir.

İndeksli ve ilişkisel diziler

```
<?php
  // İndeksli dizi
  $renkler = ["kırmızı", "yeşil", "mavi"];
  echo $renkler[0];          // kırmızı

  // İlişkisel dizi
  $kullanici = ["ad" => "Ayşe", "yas" => 30];
  echo $kullanici["ad"];     // Ayşe
?>
```

İPUCU

İlişkisel diziler PHP'nin en güçlü ve en çok kullanılan yapılarından biridir: bir kaydı (örn. bir kullanıcı) anlamlı anahtarlarla ("ad", "eposta") tutarsın — Modül 8'deki veritabanı satırları PHP'ye genelde böyle ilişkisel dizi olarak gelir. `foreach` ile bir dizinin tüm elemanlarını kolayca dolaşabilirsin. Var olmayan bir anahtara erişmek uyarı üretir; `isset($dizi["x"])` ile önce var olup olmadığını kontrol edebilirsin.

Tarayıcıda ne görünür? ÇIKTI

`$renkler[0]` ilk rengi ("kırmızı") döner; `$kullanici["ad"]` ise "ad" anahtarının değerini ("Ayşe") döner. İndeksli dizilerde konuma, ilişkisel dizilerde anlamlı bir anahtara göre erişirsin — ikisi de tek bir değişkende çok sayıda değer taşır.

Alıştırma 12 dk

Dizi kullan:

1. Beş şehirden oluşan indeksli bir dizi oluştur ve üçüncüsünü yazdır.
2. Bir kişiyi tanımlayan ilişkisel bir dizi oluştur (ad, yas, sehir).
3. foreach ile ilişkisel dizinin anahtar ve değerlerini yazdır.

BÖLÜM 08

Fonksiyonlar

Fonksiyon, bir işi yapan ve isimlendirilmiş, tekrar kullanılabilir bir kod bloğudur. Bir kez yazarsın, defalarca çağırırsın. Fonksiyonlar, kodu düzenli, okunur ve bakımı kolay kılar.

Fonksiyon tanımı ve çağrısı

- `function ad($parametreler) { ... }` ile tanımlanır.
- `return` ile bir değer döndürür.
- Çağrı: `ad($argümanlar)`.

Şema 8.1 — Fonksiyon: parametre alır, iş yapar, değer döndürür.

Fonksiyon tanımı

```
<?php
  function topla($a, $b) {
    return $a + $b;
  }

  echo topla(3, 5);        // 8
  echo topla(10, 20);      // 30  (tekrar kullan)
?>
```

İPUCU

Fonksiyonlar Modül 6'daki "problemi parçalara bölme" ve "soyutlama" ilkelerinin somut hâlidir: karmaşık bir işi küçük, isimli parçalara ayırırsın. İyi bir fonksiyon **tek bir işi** iyi yapar ve adı ne yaptığını söyler (`topla`, `selamla`). Parametrelere varsayılan değer verebilirsin: `function selamla($isim = "misafir")`. `return` bir değer geri verir; `echo` ise doğrudan yazdırır — ikisini karıştırma.

Tarayıcıda ne görünür? ÇIKTI

`selamla("Ayşe")` çağrısı fonksiyonu çalıştırır, "Merhaba, Ayşe!" metnini `return` ile döndürür ve `echo` bunu tarayıcıda gösterir. Aynı fonksiyonu farklı argümanlarla tekrar çağırarak (örn. `selamla("Veli")`) kodu bir kez yazıp defalarca kullanırsın.

Alıştırma 12 dk

Fonksiyon yaz:

1. İki sayının çarpımını döndüren bir fonksiyon yaz ve çağır.
2. Bir ismi alıp selamlama metni döndüren fonksiyon yaz.
3. Bir parametreye varsayılan değer ver ve argümansız çağır.

SEVİYE 2

Web ile Çalışmak

PHP'yi web'e bağlamak: PHP ve HTML'i birlikte kullanmak, formlar (GET/POST), form doğrulama, süper globaller, dosya ekleme (include/require) ve hata yönetimi.

BÖLÜM 09

PHP ve HTML'i Birlikte Kullanmak

PHP'nin gerçek gücü, HTML'in içine gömülebilmesidir. Sabit HTML iskeleti yazar, değişken kısımları (kullanıcı adı, ürün listesi) PHP ile doldurursun. Böylece statik sayfalar dinamik hâle gelir.

HTML içinde PHP

- PHP blokları `<?php ... ?>` HTML'in herhangi bir yerine girebilir.
- Kısa yazım: `<?= $deger ?>` (echo'nun kısası).
- Sabit kısım HTML, değişken kısım PHP olur.

Şema 9.1 — HTML iskeleti sabit, PHP değişkenleri dinamik doldurur.

HTML içinde PHP (şablon)

```
<!doctype html>
<body>
  <h1>Merhaba, <?= $ad ?>!</h1>
  <ul>
  <?php foreach ($urunler as $u) { ?>
    <li><?= $u ?></li>
  <?php } ?>
  </ul>
</body>
```

İPUCU

`<?= ... ?>` (kısa echo), şablonlarda en çok kullanılan yazımdır — `<?php echo ... ?>`'nun kısasıdır. HTML ile PHP'yi iç içe yazarken kodu okunur tutmak için mantığı (veri hazırlama) yukarıda, sunumu (HTML) aşağıda tutmak iyi bir alışkanlıktır; büyük projelerde bu ayrım "şablon motorları" ile daha da netleşir. **Önemli güvenlik notu:** kullanıcıdan gelen veriyi HTML'e basarken birazdan göreceğin gibi kaçışlamalısın (XSS önlemi).

Tarayıcıda ne görünür? ÇIKTI

`<?= $ad ?>`, HTML'in tam o noktasına değişkenin değerini (Ayşe) yazar; `foreach` ise her ürün için bir `<li>` üretir. Tarayıcıya giden şey saf HTML'dir — PHP blokları çıktıda görünmez, yalnızca ürettikleri değerler görünür.

Alıştırma 12 dk

Şablon yaz:

1. Bir başlıkta kullanıcı adını `<?= ?>` ile gösteren HTML yaz.
2. Bir diziyi foreach ile `<li>` listesine dök.
3. Sabit (HTML) ve değişken (PHP) kısımları ayırt et.

BÖLÜM 10

Formlar: GET ve POST

Web uygulamaları kullanıcıdan veri alır: arama kutusu, giriş formu, yorum alanı. Bu veri sunucuya HTTP formuyla gönderilir ve PHP'de $_GET veya $_POST süper globalleriyle okunur — Backend modülündeki GET/POST metotlarının somut hâli.

Form verisini almak

- **GET:** veri URL'de görünür; arama, filtre gibi (`$_GET`).
- **POST:** veri gövdede gizli; giriş, kayıt gibi (`$_POST`).
- Form, `method` ve `action` ile hangi metot ve nereye gideceğini belirtir.

Şema 10.1 — Form gönderimi: kullanıcı → POST → PHP $_POST → işlem.

Form ve PHP ile okuma

```
 <!-- form.html -->
<form method="post" action="kaydet.php">
  <input name="ad">
  <button>Gönder</button>
</form>

<?php // kaydet.php
  $ad = $_POST["ad"];
  echo "Merhaba, $ad";
?>
```

İPUCU

Genel kural: **veri okumak/aramak** için GET (URL'de görünür, paylaşılabilir, yer imine eklenebilir); **veri değiştirmek/göndermek** (giriş, kayıt, ödeme) için POST kullan. Şifre gibi hassas bilgiler **asla** GET ile (URL'de) gönderilmemeli. Ama en kritik nokta bir sonraki bölümde: **kullanıcıdan gelen hiçbir veriye doğrudan güvenme** — her zaman doğrula ve temizle.

Tarayıcıda ne görünür?

ÇIKTI

Kullanıcı formu doldurup gönderince, `kaydet.php` çalışır ve `$_POST["ad"]` ile girilen değeri okur; "Merhaba, [ad]" çıktısını üretir. Form, kullanıcıyla sunucu arasındaki en temel köprüdür — ama bu köprüden gelen veri her zaman denetlenmelidir.

Alıştırma

12 dk

Form işle:

1. Bir ad ve e-posta alan basit bir HTML form yaz (method="post").
2. Formu işleyip değerleri ekrana yazan PHP kodunu yaz.
3. Hangi durumda GET, hangi durumda POST kullanırdın, örnekle.

BÖLÜM 11

Form Verisini Doğrulamak

Kullanıcıdan gelen veri asla güvenilir değildir: boş olabilir, hatalı biçimde olabilir veya kötü niyetli olabilir. Doğrulama (geçerli mi?) ve temizleme (zararsız hâle getirme), her güvenli PHP uygulamasının olmazsa olmazıdır.

Doğrula ve temizle

- **Boş mu?** Zorunlu alanlar dolu mu kontrol et.
- **Biçim doğru mu?** E-posta, sayı, uzunluk denetle.
- **Temizle:** HTML'e basmadan önce kaçışla (XSS önlemi).

Şema 11.1 — Gelen veri önce doğrulanır ve temizlenir, sonra kullanılır.

Doğrulama ve temizleme

```
<?php
  $ad = trim($_POST["ad"] ?? "");
  if ($ad === "") {
    echo "Ad zorunludur.";
  } else {
    // HTML'e basmadan önce kaçışla (XSS önlemi)
    echo "Merhaba, " . htmlspecialchars($ad);
  }
?>
```

İPUCU

İki ayrı kavramı ayırt et: **doğrulama** "bu veri kurallara uyuyor mu?" (örn. geçerli e-posta), **temizleme/kaçışlama** ise "bu veri zarar veremez hâle geldi mi?" sorusudur. Kullanıcı verisini HTML'e basarken `htmlspecialchars()` kullanmak, **XSS** (siteye zararlı script enjekte etme) saldırısını önler. `$_POST["x"] ?? ""` yazımı, alan hiç gönderilmemişse hata yerine boş değer verir. Kural değişmez: **asla gelen veriye güvenme**.

Tarayıcıda ne görünür? ÇIKTI

Kod önce `trim` ile boşlukları kırpıp alanın boş olup olmadığını kontrol eder; boşsa uyarı verir. Doluysa `htmlspecialchars` ile kaçışlayıp güvenle gösterir — böylece kullanıcı ada zararlı bir script yazsa bile tarayıcıda kod olarak çalışmaz, düz metin olarak görünür.

Alıştırma 12 dk

Doğrula:

1. Bir formda "ad" alanının boş olup olmadığını kontrol eden kod yaz.
2. Kullanıcı verisini ekrana basarken neden htmlspecialchars kullanılır, açıkla.
3. Doğrulama ile temizleme arasındaki farkı bir örnekle yaz.

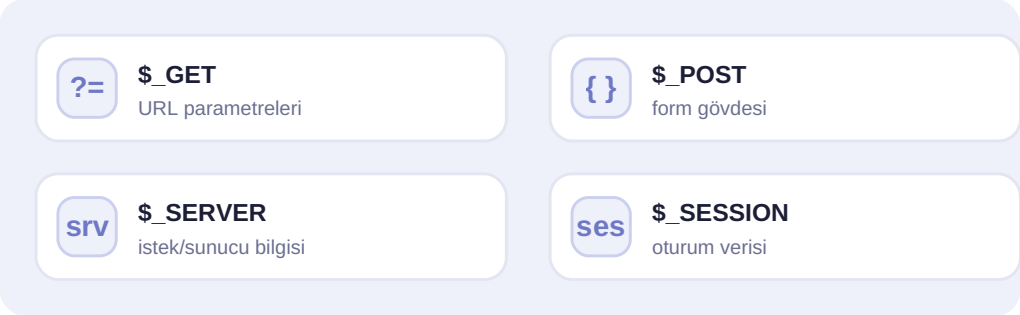
BÖLÜM 12

Süper Globaller

PHP'de bazı özel diziler her yerden erişilebilir; bunlara süper globaller denir. Form verisi, sunucu bilgisi ve oturum verisi bu dizilerde tutulur. En sık kullandıkların $_GET, $_POST, $_SERVER ve $_SESSION'dır.

Sık kullanılan süper globaller

- `$_GET` — URL'deki sorgu parametreleri.
- `$_POST` — form gövdesindeki veri.
- `$_SERVER` — istek/sunucu bilgisi (yol, metot, IP).
- `$_SESSION` — oturum verisi (kullanıcıyı hatırlar).



Şema 12.1 — PHP süper globalleri: veriye her yerden erişim.

Süper globaller

```
<?php
  $arama  = $_GET["q"] ?? "";          // URL: ?q=...
  $eposta = $_POST["eposta"] ?? "";    // form alanı
  $metot  = $_SERVER["REQUEST_METHOD"]; // GET / POST
  echo "Metot: $metot";
?>
```

İPUCU

`$_SERVER["REQUEST_METHOD"]` ile isteğin GET mi POST mu olduğunu anlayıp aynı dosyada hem formu gösterip hem işleyebilirsin (yaygın bir kalıp). Tüm süper globaller, özellikle `$_GET` ve `$_POST`, **kullanıcı kontrolündedir** — yani güvenilmez. Bu yüzden her zaman önce doğrula. `$_SESSION` ise sunucuda tutulur ve kullanıcı doğrudan değiştiremez; bu yüzden oturum/kimlik bilgisi için uygundur (Seviye 3).

Tarayıcıda ne görünür? ÇIKTI

`$_GET["q"]` URL'deki arama terimini, `$_POST["eposta"]` form alanını, `$_SERVER["REQUEST_METHOD"]` ise isteğin türünü verir. Bu diziler sayesinde PHP, tarayıcıdan gelen her bilgiye erişir — ama bu bilgilerin denetlenmesi gerektiğini unutma.

Alıştırma 10 dk

Süper global kullan:

1. URL'den bir arama terimini ($_GET) okuyup yazdır.
2. İsteğin GET mi POST mu olduğunu $_SERVER ile belirle.
3. Hangi süper globalin kullanıcı tarafından değiştirilemez olduğunu yaz.

BÖLÜM 13

Birden Çok Dosya: include / require

Gerçek projeler tek dosyaya sığmaz. Ortak parçaları (başlık, alt bilgi, ayarlar) ayrı dosyalara koyup her sayfaya dahil edersin. include ve require, bir dosyanın içeriğini başka bir dosyaya ekler.

include ve require

- `include "dosya.php";` — içeriği ekler (yoksa uyarı).
- `require "dosya.php";` — ekler (yoksa **durdurur**).
- Ortak parçalar (header, footer, config) tek yerde tutulur.

Şema 13.1 — Ortak parçalar dahil edilerek tutarlı sayfalar oluşur.

Dosya dahil etme

```
 <?php require "config.php"; ?>   <!-- ayarlar, zorunlu -->
<?php include "baslik.php"; ?>   <!-- üst menü -->

<main>Sayfanın kendine özel içeriği</main>

<?php include "altbilgi.php"; ?>  <!-- alt bilgi -->
```

İPUCU

`require` ile `include` arasındaki fark kritiktir: dosya bulunamazsa `include` sadece uyarı verip **devam eder**, `require` ise hatayla **durdurur**. Bu yüzden olmazsa olmaz dosyalar (veritabanı ayarları, güvenlik) için `require` kullan; isteğe bağlı parçalar için `include`. Ortak başlık/alt bilgiyi tek dosyada tutmak, bir değişikliği (örn. menüye yeni bağlantı) tüm sayfalara tek seferde yansıtmanı sağlar — tekrarı önleme ilkesi.

Tarayıcıda ne görünür? ÇIKTI

`require "baslik.php"` üst menüyü, `include "altbilgi.php"` alt bilgiyi her sayfaya ekler; tarayıcı bunları tek, tutarlı bir sayfa olarak görür. Menüyü değiştirmek istediğinde yalnızca `baslik.php`'yi düzenlersin — tüm sayfalar otomatik güncellenir.

Alıştırma 10 dk

Dosya böl:

1. Bir projede hangi parçaları ayrı dosyalara koyardın, listele.
2. require ile include arasındaki farkı bir örnekle açıkla.
3. Ortak başlığı tek dosyada tutmanın faydasını yaz.

BÖLÜM 14

Hata Yönetimi

Hatalar kaçınılmazdır: dosya bulunamaz, veritabanı yanıt vermez, beklenmedik veri gelir. İyi bir PHP uygulaması hataları yakalar, kullanıcıya nazik bir mesaj gösterir ve ayrıntıyı (saldırganlara değil) loglara yazar.

try / catch ile yakalamak

- `try { ... }` — riskli kodu dene.
- `catch (Exception $e) { ... }` — hata olursa yakala.
- Kullanıcıya genel mesaj, loga ayrıntı.

Şema 14.1 — try/catch: hatayı yakala, kullanıcıyı koru, ayrıntıyı logla.

Hata yakalama

```
<?php
  try {
    $db = new PDO($dsn, $kullanici, $sifre);
    // ... sorgular
  } catch (PDOException $e) {
    error_log($e->getMessage());   // ayrıntı loga
    echo "Bir hata oluştu, lütfen tekrar deneyin."; // kullanıcıya genel
  }
?>
```

İPUCU

Backend modülündeki ilkeyle aynı: **hata ayrıntısını kullanıcıya gösterme**. Bir veritabanı hatasının tam metnini ekrana basmak, saldırgana sistemin iç yapısı hakkında ipucu verir. Bunun yerine ayrıntıyı `error_log()` ile loga yaz, kullanıcıya genel bir mesaj göster. Üretimde PHP'nin ekrana hata basmasını kapatıp (`display_errors = Off`) hataları yalnızca loglara yönlendirmek standart bir güvenlik uygulamasıdır.

Tarayıcıda ne görünür? ÇIKTI

Veritabanı bağlantısı başarısız olursa `catch` bloğu devreye girer: hatanın ayrıntısını loga yazar, kullanıcıya yalnızca "Bir hata oluştu" gibi nazik bir mesaj gösterir. Uygulama çökmez, hassas bilgi sızmaz ve kullanıcı ne yapacağını bilir.

Alıştırma 10 dk

Hata yönet:

1. Riskli bir işlemi try/catch ile saran bir PHP kodu yaz.
2. Hata ayrıntısının neden kullanıcıya gösterilmemesi gerektiğini açıkla.
3. Üretimde display_errors neden kapatılır, yaz.

SEVİYE 3

Veritabanı ve Oturumlar

Kalıcı veri ve kullanıcılar: PDO ile veritabanına bağlanmak, sorgu çalıştırmak, hazırlanmış ifadelerle güvenli sorgular, veri eklemek/güncellemek, oturumlar ve çerezler.

BÖLÜM 15

Veritabanına Bağlanmak: PDO

PHP, veritabanıyla konuşmak için PDO (PHP Data Objects) kullanır. PDO, farklı veritabanlarına (MySQL, PostgreSQL, SQLite) aynı arayüzle bağlanmanı sağlar ve güvenli sorgular için hazırlanmış ifadeleri destekler.

PDO ile bağlantı

- **DSN:** hangi veritabanı, nerede (sürücü + sunucu + ad).
- Kullanıcı adı ve şifre ile bağlanılır.
- Bağlantı bir `try/catch` içinde kurulur (hata yönetimi).

Şema 15.1 — PDO, PHP ile veritabanı arasındaki standart köprüdür.

PDO bağlantısı

```
<?php
  $dsn = "mysql:host=localhost;dbname=magaza";
  try {
    $db = new PDO($dsn, $kullanici, $sifre);
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  } catch (PDOException $e) {
    error_log($e->getMessage());
    exit("Veritabanına bağlanılamadı.");
  }
?>
```

İPUCU

Bağlantı bilgilerini (sunucu, kullanıcı, şifre) **asla kaynak koda gömme** — Backend modülündeki gibi ortam değişkenlerinde veya ayar dosyasında (Git dışında) tut. `PDO::ERRMODE_EXCEPTION` ayarı, veritabanı hatalarının sessizce yutulmak yerine yakalanabilir istisnalar olarak fırlatılmasını sağlar — bu, hataları erken görmen için önemlidir. Bağlantıyı genelde tek bir `config.php` içinde kurup her sayfaya `require` ile dahil edersin.

Tarayıcıda ne görünür? ÇIKTI

`new PDO(...)` başarılı olursa `$db` üzerinden sorgu çalıştırabileceğin bir bağlantı elde edersin. Başarısız olursa `catch` devreye girer, hata loglanır ve kullanıcıya nazik bir mesaj gösterilir. PDO, bir sonraki bölümlerde güvenli sorguların da temelidir.

Alıştırma 10 dk

Bağlan:

1. Bir PDO bağlantısının hangi üç bilgiye ihtiyaç duyduğunu yaz.
2. Bağlantı bilgilerinin neden koda gömülmemesi gerektiğini açıkla.
3. Bağlantıyı neden try/catch içinde kurarız, belirt.

BÖLÜM 16

Sorgu Çalıştırmak

Bağlantı kurulduktan sonra PDO ile SQL sorguları çalıştırırsın. Bir SELECT sorgusu çalıştırıp dönen satırları PHP dizilerine alır, sonra foreach ile ekrana dökersin — Modül 8'deki SELECT'in PHP'deki karşılığı.

Sorgu ve sonuç

- `query()` veya `prepare()/execute()` ile çalıştır.
- `fetchAll()` — tüm satırları dizi olarak al.
- Her satır genelde ilişkisel bir dizidir (sütun => değer).

Şema 16.1 — Sorgu sonucu satırlar PHP'de dolaşılır ve yazdırılır.

SELECT çalıştırmak

```
<?php
  $stmt = $db->query("SELECT ad, fiyat FROM urunler");
  $urunler = $stmt->fetchAll(PDO::FETCH_ASSOC);

  foreach ($urunler as $u) {
    echo $u["ad"] . ": " . $u["fiyat"] . " TL<br>";
  }
?>
```

İPUCU

`query()` yalnızca **kullanıcı verisi içermeyen** sabit sorgular için güvenlidir (örn. "tüm ürünleri getir"). İçinde kullanıcıdan gelen herhangi bir değer varsa (örn. arama terimi, id) **asla query() ile birleştirme** — bir sonraki bölümdeki hazırlanmış ifadeleri kullan. `PDO::FETCH_ASSOC`, her satırı sütun adlarıyla bir ilişkisel dizi olarak verir (örn. `$u["ad"]`); bu, Modül 8'deki tablo satırlarının PHP'deki doğal karşılığıdır.

Tarayıcıda ne görünür? ÇIKTI

`query("SELECT ad FROM urunler")` ürün adlarını döndürür; `foreach` her satırı dolaşır ve `$satir["ad"]` ile ürün adını yazdırır. Sonuç, veritabanındaki satırların tarayıcıda bir liste olarak görünmesidir — verinin saklandığı yerden kullanıcının ekranına yolculuğu tamamlanır.

Alıştırma 12 dk

Sorgu çalıştır:

1. Bir tablodan tüm satırları çekip foreach ile yazdıran kod yaz.
2. fetchAll'ın döndürdüğü yapının ne olduğunu açıkla.
3. query()'nin ne zaman güvenli olmadığını belirt.

BÖLÜM 17

Güvenli Sorgular: Hazırlanmış İfadeler

Kullanıcı verisini bir SQL sorgusuna doğrudan gömmek, Modül 8'de gördüğün SQL enjeksiyonu açığını yaratır. PHP'de çözüm hazırlanmış ifadelerdir (prepared statements): sorgu yapısı ile veri ayrılır, böylece kullanıcı verisi asla "kod" olarak çalışmaz.

Tehlikeli ve güvenli yol

Şema 17.1 — Hazırlanmış ifade: sorgu yapısı ile kullanıcı verisini ayırır.

Hazırlanmış ifade (prepared statement)

```
<?php
  // ? yer tutucu; kullanıcı verisi ayrı gönderilir
  $stmt = $db->prepare(
    "SELECT * FROM kullanicilar WHERE eposta = ?"
  );
  $stmt->execute([$_POST["eposta"]]);
  $kullanici = $stmt->fetch(PDO::FETCH_ASSOC);
?>
```

İPUCU

Bu, tüm PHP+veritabanı çalışmanın **en önemli güvenlik kuralıdır**: kullanıcıdan gelen herhangi bir değer içeren her sorgu, hazırlanmış ifadeyle yazılmalıdır. `?` (veya isimli `:eposta`) yer tutucuları kullan, gerçek değerleri `execute()`'a ayrı bir dizi olarak ver. Böylece veritabanı önce sorgu yapısını derler, sonra veriyi yalnızca bir değer olarak yerleştirir — saldırgan girdiye SQL yazsa bile çalışmaz. Modül 8'deki parametreli sorgu ilkesinin PHP karşılığı tam olarak budur.

Tarayıcıda ne görünür? ÇIKTI

`prepare("...WHERE eposta = ?")` sorgunun iskeletini tanımlar; `execute([$_POST["eposta"]])` kullanıcı verisini ayrı olarak yerleştirir. Kullanıcı e-posta alanına zararlı bir SQL parçası yazsa bile, o değer yalnızca aranan bir metin olarak ele alınır — sorgunun yapısını değiştiremez. Sonuç hem doğru hem güvenlidir.

Alıştırma 14 dk

Güvenli sorgula:

1. Kullanıcı verisini birleştiren tehlikeli bir sorguyu hazırlanmış ifadeye çevir.
2. `?` yer tutucusunun enjeksiyonu nasıl önlediğini açıkla.
3. Hangi sorguların hazırlanmış ifade gerektirdiğini belirt.

BÖLÜM 18

Veri Eklemek ve Güncellemek

Veriyi okumak kadar yazmak da işin parçasıdır: yeni kayıt eklemek (INSERT), var olanı güncellemek (UPDATE). PDO ile bunları da hazırlanmış ifadelerle, güvenli biçimde yaparsın.

Yazma işlemleri

- **INSERT:** yeni satır ekle (hazırlanmış ifadeyle).
- **UPDATE:** var olanı güncelle (WHERE ile sınırla!).
- Değerler her zaman yer tutucularla (`?`) verilir.

Şema 18.1 — Yazma işlemleri de hazırlanmış ifadelerle güvenle yapılır.

INSERT ve UPDATE (PDO)

```
<?php
  // Ekle
  $stmt = $db->prepare(
    "INSERT INTO urunler (ad, fiyat) VALUES (?, ?)"
  );
  $stmt->execute([$_POST["ad"], $_POST["fiyat"]]);

  // Güncelle (WHERE şart!)
  $g = $db->prepare("UPDATE urunler SET fiyat = ? WHERE id = ?");
  $g->execute([$_POST["fiyat"], $_POST["id"]]);
?>
```

İPUCU

Modül 8'deki uyarı PHP'de de geçerli: **UPDATE ve DELETE'i her zaman WHERE ile sınırla** — yer tutucularla bile olsa WHERE'i unutursan tüm tabloyu güncellersin. Yazma işlemlerinde girdiyi önce doğrulamayı (boş mu, biçim doğru mu) ve hazırlanmış ifade kullanmayı asla atlama. Birden çok yazma işlemini "ya hep ya hiç" yapmak istersen (Modül 8'deki işlemler/transactions), PDO bunu da `beginTransaction()` / `commit()` ile destekler.

Tarayıcıda ne görünür? ÇIKTI

`prepare` + `execute` ikilisi, formdan gelen ürün bilgisini güvenle veritabanına ekler; ikinci örnek belirli bir ürünün fiyatını `WHERE id = ?` ile yalnızca o satırda günceller. Kullanıcı verisi her zaman yer tutucularla geçtiği için işlem hem güvenli hem nettir.

Alıştırma 12 dk

Veri yaz:

1. Formdan gelen veriyi hazırlanmış ifadeyle ekleyen INSERT kodu yaz.
2. Tek bir kaydı güncelleyen, WHERE'li bir UPDATE yaz.
3. WHERE'siz UPDATE'in PHP'de de neden tehlikeli olduğunu açıkla.

BÖLÜM 19

Oturumlar (Sessions)

HTTP durumsuzdur: sunucu, istekler arasında seni hatırlamaz. Oturumlar (sessions), kullanıcıyı istekler boyunca hatırlamanın yoludur — özellikle giriş yapmış kullanıcıyı takip etmek için. Backend modülündeki "durumsuzluk" sorununun çözümüdür.

Oturum nasıl çalışır?

- `session_start()` ile oturum başlatılır.
- `$_SESSION` dizisine veri yazılır (sunucuda saklanır).
- Tarayıcıya bir oturum kimliği (çerez) verilir; her istekte gelir.

Şema 19.1 — Oturum: kullanıcı bir kez giriş yapar, sonraki isteklerde hatırlanır.

Oturum kullanımı

```
<?php
  session_start();

  // Giriş başarılıysa:
  $_SESSION["kullanici_id"] = $kullanici["id"];

  // Başka bir sayfada: giriş yapmış mı?
  if (isset($_SESSION["kullanici_id"])) {
    echo "Hoş geldin!";
  }
?>
```

İPUCU

Oturum verisi **sunucuda** tutulur; tarayıcıya yalnızca rastgele bir oturum kimliği (çerez) gider. Bu yüzden oturum, kimlik gibi hassas durumlar için çerezden daha güvenlidir — kullanıcı içeriği göremez/değiştiremez. Güvenlik için: girişten sonra `session_regenerate_id()` çağırmak oturum çalınmasını zorlaştırır; çıkışta `session_destroy()` ile oturumu temizle. `session_start()` her sayfada, çıktıdan önce çağrılmalıdır.

Tarayıcıda ne görünür? ÇIKTI

Kullanıcı giriş yapınca `$_SESSION["kullanici_id"]` sunucuda saklanır; sonraki her sayfada `session_start()` + `isset(...)` ile "bu kullanıcı giriş yapmış mı?" kontrol edilir. Böylece kullanıcı her sayfada yeniden giriş yapmak zorunda kalmaz — sistem onu hatırlar.

Alıştırma 12 dk

Oturum kur:

1. Giriş sonrası kullanıcıyı $_SESSION ile "hatırlayan" kod yaz.
2. Başka bir sayfada kullanıcının giriş yapıp yapmadığını kontrol et.
3. Oturum verisinin neden çerezden daha güvenli olduğunu açıkla.

BÖLÜM 20

Çerezler ve Kimlik

Çerezler (cookies), tarayıcıda saklanan küçük verilerdir ve her istekte sunucuya geri gönderilir. Oturumlarla birlikte, kullanıcıyı hatırlamanın ve kimlik doğrulamanın temelini oluştururlar. İkisinin farkını ve doğru kullanımını bilmek önemlidir.

Çerez ve oturum

Şema 20.1 — Oturum verisi sunucuda, çerez verisi tarayıcıda saklanır.

Çerez kullanımı

```
<?php
  // Çerez ayarla (30 gün geçerli)
  setcookie("tema", "koyu", time() + 30*24*60*60);

  // Sonraki isteklerde oku
  $tema = $_COOKIE["tema"] ?? "acik";
?>
```

İPUCU

Temel kural: **hassas veriyi çereze koyma** (kullanıcı görebilir ve değiştirebilir). Kimlik gibi şeyler oturumda (sunucuda) tutulmalı; çerezler tercih/ayar (tema, dil) veya oturum kimliğini taşımak için uygundur. Güvenlik için çerezleri `HttpOnly` (JavaScript erişemez, XSS'e karşı) ve `Secure` (yalnızca HTTPS) bayraklarıyla ayarla. Şifreleri çerezde **asla** saklama. Kimlik doğrulamanın iyi tasarımı: oturum + güvenli çerez + hash'lenmiş şifreler.

Tarayıcıda ne görünür? ÇIKTI

`setcookie("tema", "koyu", ...)` tarayıcıya bir çerez kaydeder; sonraki isteklerde `$_COOKIE["tema"]` ile okunur ve kullanıcının tema tercihi hatırlanır. Çerez tarayıcıda durduğu için tercih gibi zararsız veriler için uygundur — ama kimlik gibi hassas bilgiler oturumda tutulur.

Alıştırma 10 dk

Çerez kullan:

1. Kullanıcının tema tercihini çerezde saklayan ve okuyan kod yaz.
2. Oturum ile çerez arasındaki temel farkı açıkla.
3. Neden hassas verinin çereze konmaması gerektiğini yaz.

SEVİYE 4

Yapı ve Üretim

Gerçek uygulamalar: fonksiyonlardan sınıflara (OOP), kod düzeni ve Composer, güvenlik ilkeleri, basit bir API uç noktası, dağıtım ve küçük bir PHP uygulaması.

BÖLÜM 21

Fonksiyonlardan Sınıflara: OOP

Uygulamalar büyüdükçe, ilişkili veri ve davranışı bir arada paketlemek işe yarar. Nesne yönelimli programlama (OOP), bunu sınıflarla yapar: bir sınıf, özellikleri (veri) ve davranışları (fonksiyonlar) bir kalıpta birleştirir.

Sınıf ve nesne

- **Sınıf (class):** bir kalıp; özellikler + davranışlar.
- **Nesne (object):** sınıftan üretilen somut örnek.
- `$k = new Kullanici();` ile nesne oluşturulur.

Şema 21.1 — Bir sınıf: özellikler (veri) ve davranışlar (metotlar) bir arada.

Sınıf tanımı ve kullanımı

```
<?php
  class Kullanici {
    public $ad;
    function __construct($ad) { $this->ad = $ad; }
    function selamla() { return "Merhaba, " . $this->ad; }
  }

  $k = new Kullanici("Ayşe");
  echo $k->selamla();      // Merhaba, Ayşe
?>
```

İPUCU

OOP, Modül 6'daki **soyutlama** ve "ilgili şeyleri bir arada tutma" fikirlerinin güçlü bir biçimidir. Bir `Kullanici` sınıfı, kullanıcıyla ilgili tüm veri ve işlemleri tek bir yerde toplar; her yeni kullanıcı bu kalıptan üretilen bir nesnedir. `$this`, nesnenin kendi özelliklerine erişmesini sağlar. OOP büyük projelerde kodu düzenli ve yeniden kullanılabilir kılar; küçük betikler için fonksiyonlar yeterli olabilir — aracı probleme göre seç.

Tarayıcıda ne görünür? ÇIKTI

`new Kullanici("Ayşe")`, sınıf kalıbından somut bir nesne üretir; `$k->selamla()` o nesnenin davranışını çağırır ve "Merhaba, Ayşe" döner. Sınıf bir kez tanımlanır, ondan istediğin kadar nesne (Ayşe, Veli, Can) üretebilirsin — her biri kendi verisini taşır.

Alıştırma 14 dk

Sınıf yaz:

1. Bir "Urun" sınıfı tasarla: özellikleri (ad, fiyat) ve bir davranışı.
2. Sınıftan iki nesne oluştur ve bir metodunu çağır.
3. Sınıf ile nesne arasındaki farkı kendi cümlenle açıkla.

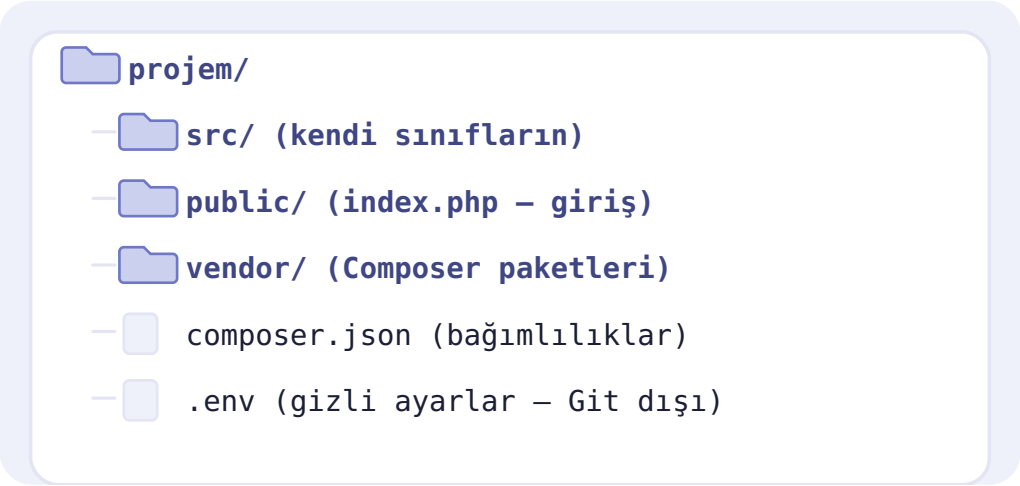
BÖLÜM 22

Kod Düzeni ve Composer

Profesyonel PHP projeleri düzenli bir klasör yapısına ve dış kütüphaneleri yöneten bir araca sahiptir: Composer. Composer, hazır paketleri projene ekler ve sınıfları otomatik yükler — tekerleği yeniden icat etmemeni sağlar.

Düzen ve bağımlılıklar

- **Klasör yapısı:** kod, genel dosyalar ve bağımlılıklar ayrılır.
- **composer.json:** projenin bağımlılıklarını listeler.
- **Otomatik yükleme:** sınıfları elle include etmeden kullan.



Şema 22.1 — Tipik bir PHP proje yapısı: kod, genel dosyalar ve bağımlılıklar ayrı.

İPUCU

Composer, PHP dünyasının paket yöneticisidir (Node'daki npm gibi). `composer require` ile bir kütüphane eklersin; o ve bağımlılıkları `vendor/` klasörüne iner ve `vendor/autoload.php` ile hepsini tek satırda kullanılabilir kılarsın. **vendor/ klasörünü Git'e koyma** (composer.json yeterlidir; başkaları `composer install` ile aynı paketleri kurar). Yalnızca `public/` klasörünü web'e açmak (kodun gerisini gizlemek) önemli bir güvenlik uygulamasıdır.

Tarayıcıda ne görünür? ÇIKTI

İyi düzenlenmiş bir projede senin kodun `src/` 'de, dışa açık giriş noktası `public/index.php` 'de, dış paketler `vendor/` 'da durur. `composer.json` hangi paketlere ihtiyaç olduğunu söyler; böylece proje başka bir makinede tek komutla (`composer install`) kurulabilir. Düzen, projeyi büyümeye ve ekip çalışmasına hazır kılar.

Alıştırma

10 dk

Projeyi düzenle:

1. Bir PHP projesinin temel klasörlerini ve görevlerini yaz.
2. Composer'ın ne işe yaradığını kendi cümlenle açıkla.
3. Neden yalnızca public/ klasörünün web'e açıldığını belirt.

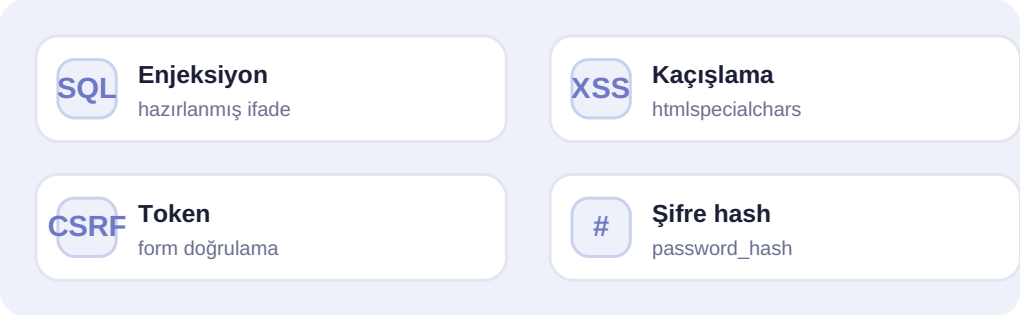
BÖLÜM 23

Güvenlik İlkeleri

PHP, web'in en çok saldırıya uğrayan katmanlarından birinde çalışır. Birkaç temel ilkeyi alışkanlık hâline getirmek, uygulamanı en yaygın saldırıların çoğundan korur. Güvenlik baştan tasarlanır, sonradan eklenmez.

Olmazsa olmaz önlemler

- **SQL enjeksiyonu:** hazırlanmış ifadeler (Bölüm 17).
- **XSS:** çıktıyı kaçışla (`htmlspecialchars`).
- **CSRF:** formlara gizli token ekle.
- **Şifreler:** `password_hash` ile sakla, asla düz metin.



Şema 23.1 — PHP güvenliğinin dört temel taşı.

Şifre hash'leme

```
<?php
  // Kayıtta: şifreyi hash'le (asla düz metin sakla!)
  $hash = password_hash($_POST["sifre"], PASSWORD_DEFAULT);

  // Girişte: girilen şifreyi hash ile doğrula
  if (password_verify($_POST["sifre"], $hash)) {
    echo "Giriş başarılı";
  }
?>
```

İPUCU

Dört kuralı alışkanlık yap: **(1)** her veritabanı sorgusunda hazırlanmış ifade, **(2)** ekrana bastığın her kullanıcı verisinde `htmlspecialchars` , **(3)** her veri değiştiren formda CSRF token'ı, **(4)** şifreleri `password_hash` ile sakla, `password_verify` ile doğrula. Şifreyi asla düz metin veya basit hash'le saklama — `password_hash` güçlü ve güncel algoritmaları otomatik kullanır. Ve her zaman **HTTPS**. Bu önlemler, gerçek dünya saldırılarının büyük çoğunluğunu durdurur.

Tarayıcıda ne görünür? ÇIKTI

`password_hash` şifreyi geri döndürülemez biçimde saklar; girişte `password_verify` girilen şifreyi bu hash ile karşılaştırır — şifrenin kendisi hiçbir zaman saklanmaz. Veritabanı çalınsa bile şifreler okunamaz. Bu dört önlem birlikte, uygulamanı en yaygın saldırı türlerine karşı sağlamlaştırır.

Alıştırma 12 dk

Güvenliği uygula:

1. Dört temel PHP güvenlik önlemini ve neye karşı koruduğunu yaz.
2. Şifre kaydetme ve doğrulama akışını password_hash ile yaz.
3. Şifrelerin neden hiçbir zaman düz metin saklanmadığını açıkla.

BÖLÜM 24

Basit Bir API Uç Noktası

PHP yalnızca HTML üretmez; JSON da döndürebilir. Böylece Backend modülünde öğrendiğin türden bir API uç noktası yazabilir, verini başka uygulamalara (örneğin bir JavaScript önyüzüne) sunabilirsin.

JSON yanıtı döndürmek

- `header("Content-Type: application/json")` ile tür belirt.
- `json_encode()` ile PHP dizisini JSON'a çevir.
- Sonuç: başka uygulamaların tüketebileceği bir API.

Şema 24.1 — PHP, JSON döndürerek bir API uç noktası olur.

JSON API uç noktası

```
<?php
  header("Content-Type: application/json");

  $stmt = $db->query("SELECT ad, fiyat FROM urunler");
  $urunler = $stmt->fetchAll(PDO::FETCH_ASSOC);

  echo json_encode($urunler);   // dizi -> JSON
?>
```

İPUCU

Bu, Backend (Modül 7) ve Veritabanı (Modül 8) ile PHP'nin birleştiği noktadır: PHP veritabanından veri çeker, `json_encode` ile JSON'a çevirir ve `Content-Type: application/json` başlığıyla döner — tam da REST API'lerin yaptığı gibi. Bir JavaScript önyüzü (Modül 5'teki `fetch`) bu uç noktayı çağırıp veriyi ekrana basabilir. Gerçek API'lerde ayrıca doğru durum kodlarını (200, 404...) döndürmeyi ve girdileri doğrulamayı unutma.

Tarayıcıda ne görünür? ÇIKTI

Bu uç nokta veritabanından ürünleri çeker ve `json_encode` ile JSON dizisine çevirip döner. Tarayıcı yerine başka bir program (örneğin bir JavaScript uygulaması) bu uç noktayı çağırır ve dönen JSON'u kendi ekranında kullanır — PHP, görünür bir sayfa değil, bir veri servisi olur.

Alıştırma 12 dk

API yaz:

1. Bir diziyi JSON olarak döndüren basit bir PHP uç noktası yaz.
2. Content-Type başlığının neden gerekli olduğunu açıkla.
3. Bu uç noktayı hangi tür uygulamaların tüketebileceğini yaz.

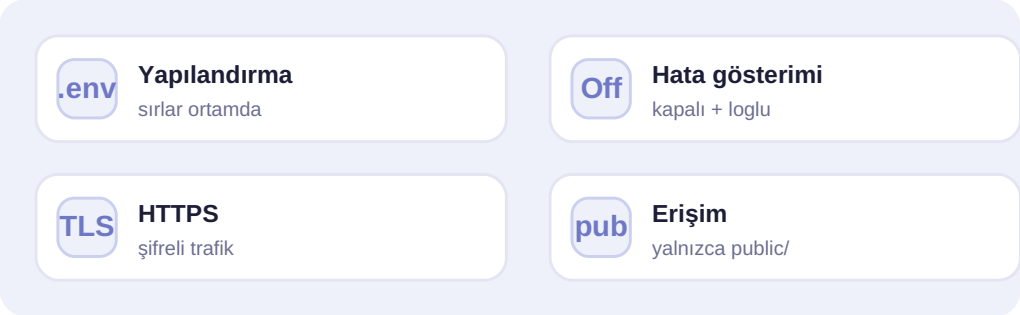
BÖLÜM 25

Dağıtım ve Yapılandırma

Uygulaman çalışır hâle geldiğinde, onu bir sunucuya taşıyıp dünyaya açarsın: dağıtım (deployment). Geliştirme ortamından üretime geçerken yapılandırma, güvenlik ve performans ayarları önem kazanır.

Üretime hazırlık

- **Ortam değişkenleri:** sırlar koda değil, ortama (Backend Bölüm 21).
- **Hata gösterimi kapalı:** `display_errors = Off`, loglar açık.
- **HTTPS:** tüm trafiği şifrele.
- **Yalnızca public/ açık:** kodun gerisini gizle.



Şema 25.1 — Üretim ortamı: güvenli yapılandırma ilkeleri.

İPUCU

Geliştirme ile üretim ortamı farklı davranmalı: geliştirmede hataları ekranda görmek istersin (hızlı düzeltme için), üretimde ise **asla** — hatalar yalnızca loglara gitmeli. Bu farkı ortam değişkenleriyle yönetirsin (aynı kod, farklı ayar). Üretime almadan önce kontrol listesi: sırlar koda gömülü değil, HTTPS aktif, hata gösterimi kapalı, veritabanı yedekleniyor (Modül 8), bağımlılıklar güncel. Dağıtım, "çalışıyor" ile "güvenle yayında" arasındaki köprüdür.

Tarayıcıda ne görünür? ÇIKTI

Üretime geçerken aynı kod, ortam değişkenleri sayesinde gerçek veritabanına bağlanır, hataları ekrana basmak yerine loglar ve HTTPS üzerinden çalışır. Kullanıcı yalnızca `public/` klasörüne erişir; kodun, ayarların ve sırların gerisi gizli kalır. Doğru yapılandırma, uygulamanı gerçek dünyaya güvenle açar.

Alıştırma

10 dk

Dağıtıma hazırlan:

1. Üretime almadan önce kontrol edeceğin beş maddeyi listele.
2. Geliştirme ve üretimde hata gösteriminin neden farklı olduğunu açıkla.
3. Sırların üretimde nasıl yönetilmesi gerektiğini yaz.

BÖLÜM 26

Bitirme: Küçük Bir PHP Uygulaması

Tüm öğrendiklerini birleştirip baştan sona küçük bir uygulama tasarlıyorsun: form alır, doğrular, veritabanına güvenle yazar, listeler ve kullanıcıyı oturumla hatırlar. Bu, gerçek bir PHP uygulamasının çekirdeğidir.

Bir "not" uygulamasının akışı

Şema 26.1 — Küçük bir uygulama: form → güvenli yazma → listeleme → oturum.

Uygulama kontrol listesi

```
// 1. Form verisini al ve DOĞRULA (boş mu, biçim?)
// 2. PDO + prepare/execute ile GÜVENLE yaz
// 3. SELECT ile listele, htmlspecialchars ile KAÇIŞLA
// 4. session_start + $_SESSION ile kullanıcıyı hatırla
// 5. password_hash ile şifreleri sakla
// 6. Ortam değişkenleri + HTTPS + hata loglama
```

İPUCU

Gerçek bir uygulama yazarken bu kontrol listesini izle: girdiyi **doğrula**, veritabanına **hazırlanmış ifadeyle** yaz, çıktıyı **kaçışla**, kullanıcıyı **oturumla** hatırla, şifreleri **hash'le**, sırları **ortam değişkeninde** tut. Bu modülü tamamladıysan, artık sunucu tarafını gerçek kodla yazabiliyorsun — Backend'in kavramları, SQL'in sorguları ve PHP'nin sözdizimi birleşti. Sıradaki diller (Python, C#) aynı kavramları farklı sözdizimiyle uygular; temel mantık değişmez.

Tarayıcıda ne görünür? ÇIKTI

Tasarladığın not uygulaması: kullanıcı bir not girer (doğrulanır), hazırlanmış ifadeyle veritabanına güvenle yazılır, mevcut notlar kaçışlanarak listelenir ve kullanıcı oturumla hatırlanır. Form, doğrulama, PDO, güvenlik ve oturum bir araya gelince — çalışan, güvenli, gerçek bir PHP uygulaman olur.

Alıştırma 20 dk

Uygulamanı tasarla:

1. Küçük bir uygulama seç (yapılacaklar listesi, ziyaretçi defteri).
2. Form → doğrulama → güvenli yazma → listeleme akışını tasarla.
3. Hangi güvenlik önlemlerini (hazırlanmış ifade, kaçışlama, hash) ekleyeceğini yaz.
4. Kullanıcıyı oturumla nasıl hatırlayacağını belirt.

EK

PHP Terimleri ve Komutları Sözlüğü

En sık kullanılan PHP yapıları ve komutları. Bir başvuru kaynağı olarak saklayabilirsin.

<?php ... ?>	PHP bloğu
echo	Tarayıcıya yazdır
$degisken	Değişken
. (nokta)	Metin birleştir
if / else	Koşul
for / foreach	Döngü
[ ] dizi	Çoklu değer
function	Fonksiyon
$_POST / $_GET	Form verisi
PDO	Veritabanı erişimi
prepare / execute	Güvenli sorgu
session	Oturum
password_hash	Şifre hash'leme
include / require	Dosya ekleme

PHP'nin özeti ÇIKTI

PHP sunucuda çalışır ve tarayıcıya HTML üretir. **echo** ile yazdırır, **$değişkenlerle** veri taşır, **if** ve **döngülerle** akışı yönetir, **dizilerle** çoklu veriyi, **fonksiyonlarla** tekrar kullanılabilir mantığı kurar. Formlardan (**$_POST**) veri alır, **PDO** ve **hazırlanmış ifadelerle** veritabanına güvenle bağlanır, **oturumlarla** kullanıcıyı hatırlar. Güvenliğin altın kuralı Modül 8'le aynıdır: kullanıcı verisine güvenme — her zaman doğrula ve parametreli sorgu kullan.