

PHP de la zero

Caiet de laboratoare și verificare

Ediția 16 octombrie 2026. Exerciții locale cu date fictive. Lectura este estimată separat de practica individuală. Exemple verificate cu PHP 8.3; sunt necesare mbstring și pdo_sqlite pentru laboratoarele respective.

Cuprins

1. Cum rulează PHP
2. Variabile, tipuri, string-uri
3. Funcții și array-uri
4. Structuri de control
5. Formulare și date de la utilizator
6. Sesiuni și autentificare simplă
7. PDO: prima interogare

Cum folosești caietul

Parcurge mai întâi lecția din curs sau manual. Caietul folosește funcțiile și fișierele definite acolo; nu este un pachet independent de cod. Pentru fiecare test notează intrarea, predicția, rezultatul efectiv și corectura.

Cum rulează PHP

Laborator suplimentar: lista participanților, de la date la HTML

Construiește pagina `public/participanti.php` într-un director nou de laborator. Cerința este să afișeze numărul participanților și numele lor, iar pentru o listă goală să afișeze un mesaj explicit. Numărul trebuie calculat din aceeași listă folosită la afișare. În acest fel, dacă introduci sau elimini un participant, nu trebuie să corectezi manual două zone ale paginii.

Înainte de implementare, scrie rezultatul pentru trei intrări: lista cu Ana și Mihai, lista goală și lista care conține numele Atelier <A&B>. Prima intrare trebuie să genereze două elemente li. A doua nu trebuie să genereze un participant fictiv cu numele „nimeni”. A treia trebuie să păstreze semnele speciale ca parte a numelui, fără să construiască un element HTML din ele.

```
<?php
declare(strict_types=1);
$participanti = ['Ana', 'Mihai', 'Atelier <A&B>'];
function h(string $text): string {
    return htmlspecialchars($text, ENT_QUOTES | ENT_SUBSTITUTE, 'UTF-8');
}
?>
<!doctype html>
<html lang="ro"><meta charset="utf-8"><title>Participanți</title>
<h1>Participanți: <?= count($participanti) ?></h1>
<?php if ($participanti === []): ?>
    <p>Nu există participanți încă.</p>
<?php else: ?>
    <ul>
        <?php foreach ($participanti as $nume): ?>
            <li><?= h($nume) ?></li>
        <?php endforeach; ?>
    </ul>
<?php endif; ?>
</html>
```

Nu memora toate instrucțiunile din prima încercare. Urmărește traseul: array-ul furnizează datele, count calculează numărul, condiția decide dacă există elemente, iar foreach repetă marcajul pentru fiecare nume. Funcția `h` pregătește numele pentru contextul HTML. Niciunul dintre acești pași nu schimbă lista originală. În lecțiile următoare vei putea verifica separat fiecare operație.

Compară trei feluri de a privi aceeași pagină

Fișierul din editor este sursa programului. Răspunsul HTTP este rezultatul execuției. DOM-ul din browser este structura construită de browser din acel răspuns. Aceste trei lucruri sunt legate, dar nu sunt identice. Dacă inspectorul arată un element adăugat automat de browser, nu înseamnă că PHP l-a scris explicit în fișier. Pentru a verifica ce a trimis serverul, examinează răspunsul din fila de rețea sau sursa paginii.

În sursa răspunsului caută textul Atelier și observă entitățile folosite pentru semnele speciale. În pagina vizuală trebuie să vezi numele original. Acesta nu este un defect de codificare: entitățile sunt reprezentarea potrivită pentru ca browserul să trateze semnele ca text. Dacă vezi entitățile literal în pagină, este posibil să fi aplicat escaparea de două ori.

Rulează același fișier și cu `php public/participanti.php` din terminal. Vei vedea HTML-ul ca text, deoarece terminalul nu îl interpretează ca browserul. Acest experiment demonstrează că PHP poate produce același tip de ieșire fără o interfață grafică. Totuși, execuția din terminal nu testează antete, cookie-uri sau comportamentul unui server HTTP, deci nu înlocuiește verificarea în contextul paginii.

Găsește cauza înainte să modifichi programul

Dacă primești 404, verifică adresa și rădăcina publică a serverului. Dacă primești o eroare de sintaxă, verifică fișierul indicat și linia din jurul erorii. Dacă pagina apare, dar lista este goală, inspectează datele și condiția. Aceste simptome apar în etape diferite: rezolvarea adresei, interpretarea PHP și logica programului. Nu încerca să repara o adresă greșită schimbând funcția de escapare.

Pentru un experiment controlat, introdu deliberat un nume de fișier greșit în URL și notează răspunsul. Revino la adresa corectă, apoi schimbă lista în array gol. Compară cele două rezultate: în primul caz resursa nu a fost găsită, în al doilea pagina funcționează și descrie o colecție fără elemente. Diferența te ajută să scrii mesaje corecte și în aplicațiile cu baze de date.

La final, păstrează trei dovezi simple: adresa testată, numărul elementelor li și numele special afișat literal. Repetă testul după adăugarea unui al patrulea nume. Dacă titlul arată patru participanți, dar lista are trei, ai două surse de adevăr sau o condiție care exclude un element. Corectează cauza, apoi repetă cazul gol pentru a verifica și ramura care nu produce listă.

Variabile, tipuri, string-uri

Laborator suplimentar: trei etape pentru aceeași cantitate

Păstrează funcția `cantitateValida` din lecție într-un fișier separat, fără apelul demonstrativ de la final. În scriptul de lucru, include acel fișier și construiește un rezumat pentru un produs. Scopul este să urmărești separat intrarea brută, cantitatea acceptată și mesajul afișat. Dacă ai o eroare de validare, nu construiești un total folosind o valoare implicită aleasă pentru a evita oprirea programului.

Pentru primul caz, produsul costă 1250 bani, iar cantitatea primită este textul „3”. După validare ai întregul 3, după calcul ai întregul 3750, iar pentru afișare ai textul „37,50 lei”. Aceste rezultate seamănă între ele, dar fiecare aparține unei etape diferite. Schimbarea separatorului zecimal din afișare nu trebuie să afecteze primele două etape.

```
<?php
declare(strict_types=1);
require __DIR__ . '/cantitate.php';
$nume = 'Agendă <A5>';
$pretBani = 1250;
$cantitateBruta = '3';
try {
    $cantitate = cantitateValida($cantitateBruta);
    $totalBani = $pretBani * $cantitate;
    $mesaj = $nume . ': ' . number_format($totalBani / 100, 2, ',', '.') . ' lei';
    echo htmlspecialchars($mesaj, ENT_QUOTES | ENT_SUBSTITUTE, 'UTF-8');
} catch (InvalidArgumentException $e) {
    echo htmlspecialchars($e->getMessage(), ENT_QUOTES | ENT_SUBSTITUTE, 'UTF-8');
}
```

Acest script folosește un preț intern fix și mic. Dacă prețul ar proveni dintr-o sursă externă, ar avea nevoie de propriul contract și de verificarea domeniului înainte de înmulțire. Nu interpreta validarea cantității ca pe o validare a întregii comenzi. Fiecare intrare trebuie evaluată potrivit rolului ei, iar faptul că o valoare este înt nu dovedește că aparține domeniului permis.

De ce testul cu zero trebuie scris de două ori

În scenariul nostru, cantitatea zero este interzisă, dar prețul zero poate reprezenta un produs gratuit. Prin urmare, scriem două teste distincte. Primul trimite cantitatea „0” și așteaptă eroare. Al doilea folosește prețul 0 și cantitatea „3” și așteaptă totalul 0. Dacă aceeași verificare `empty` respinge ambele situații, implementarea a confundat două reguli diferite.

Această observație este utilă dincolo de bani. Vârsta zero poate fi validă pentru un copil, în timp ce numărul de participanți zero poate împiedica începerea unui atelier. Un text „0” poate fi un cod acceptat, iar șirul gol poate indica lipsa codului. Valoarea nu are sens comercial izolată de câmpul în care apare. Numele și regulile câmpurilor trebuie să păstreze acest context.

Experimentează cu spații și cu formatul întregului

Funcția normalizează marginile prin `trim`, deci „ 3 ” este acceptat ca 3. Nu elimină spații din interior și nu ghicește ce înseamnă „3 cutii”. Filtrul ales are propriile reguli pentru reprezentarea unui întreg. Dacă produsul trebuie să accepte un format diferit, precum zerouri inițiale în anumite coduri, nu reutiliza automat validatorul de cantități pentru acel câmp.

Încearcă intrările „1”, „100”, „101”, „3.5”, „3cutii”, șirul gol și un array. Pentru fiecare, notează mai întâi predicția, apoi rezultatul. Când predicția diferă de rezultat, verifică dacă problema este în cerință, în înțelegerea filtrului sau în cod. Nu schimba testul doar pentru a transforma rezultatul curent într-un succes. Un test este util tocmai când îți contrazice presupunerea.

Observă diferența dintre afișare și diagnostic

Echo poate afișa true ca 1 și poate face unele valori să pară identice. Var_dump arată și tipul, deci este mai potrivit pentru experimentele din terminal. Nu lăsa însă diagnosticarea completă a intrărilor într-o pagină publică: un array primit de la utilizator poate conține informații pe care nu vrei să le expui. Pentru laborator folosește date fictive și scoate diagnosticul din versiunea finală a paginii.

Compară strlen și mb_strlen pentru textul „țară”, specificând UTF-8 în a doua funcție. Prima numără octeții, a doua numără punctele de cod. Dacă editorul salvează în altă codificare decât cea presupusă, rezultatul nu mai verifică experimentul dorit. Notează codificarea fișierului și cerința extensiei mbstring, ca altcineva să poată reproduce observația în propriul mediu.

Corectura unei soluții aparent funcționale

O soluție greșită transformă imediat orice intrare în int, calculează totalul, apoi afișează rezultatul. Pentru „3” pare corectă. Pentru „3cutii” poate produce tot trei și ascunde faptul că formatul nu era acceptat. Corectura nu constă în adăugarea unui mesaj după calcul, ci în mutarea validării înaintea conversiei folosite de logica internă. Datele respinse nu trebuie să ajungă la calcul.

Predarea laboratorului include un caz valid, două limite, două forme invalide și cazul produsului gratuit. Pentru fiecare caz valid, arată tipul cantității și totalul înainte de formatare. Pentru fiecare caz invalid, arată că nu a fost construit un total. Astfel demonstrezi un flux de date controlat, nu doar o propoziție care arată bine pentru un singur exemplu favorabil.

Funcții și array-uri

Laborator suplimentar: raportul complet al coșului

În această variantă, raportul nu acceptă poziții cu structură incompletă și nu modifică array-ul primit. Id-ul identifică poziția, nu produsul. Pentru claritate, acceptăm id-uri textuale nevide și tipuri interne deja corecte. Conversia din datele formularului ar trebui făcută înaintea apelului, într-o funcție dedicată. Nu amestecăm în această funcție validarea HTTP cu agregarea raportului.

Păstrează funcția subtotalBani din lecție într-un fișier separat, fără apelul demonstrativ. Apoi implementează funcția de mai jos. Verificarea prin array_key_exists distinge absența câmpului de o valoare prezentă, iar is_int exprimă contractul intern al prețului și cantității. Un șir „3” nu este acceptat aici doar fiindcă ar putea fi interpretat numeric în alt strat al aplicației.

```
<?php
declare(strict_types=1);
require __DIR__ . '/subtotal.php';
function raportCos(array $pozitii): array {
    $raport = [];
    $vazute = [];
    $total = 0;
    foreach ($pozitii as $index => $p) {
        if (!is_array($p)) {
            throw new InvalidArgumentException('Poziție invalidă: ' . $index);
        }
        foreach (['id', 'pret_bani', 'cantitate'] as $camp) {
            if (!array_key_exists($camp, $p)) {
                throw new InvalidArgumentException('Lipsește câmpul ' . $camp);
            }
        }
        if (!is_string($p['id']) || trim($p['id']) === '' ||
            !is_int($p['pret_bani']) || !is_int($p['cantitate'])) {
            throw new InvalidArgumentException('Tipuri interne invalide');
        }
        // Prefixul păstrează id-urile numerice ca parte dintr-o cheie textuală.
        $scheie = 'id:' . $p['id'];
        if (isset($vazute[$scheie])) {
            throw new InvalidArgumentException('Id duplicat');
        }
        $vazute[$scheie] = true;
        $subtotal = subtotalBani($p['pret_bani'], $p['cantitate']);
        if ($total > PHP_INT_MAX - $subtotal) {
            throw new OverflowException('Total prea mare');
        }
        $total += $subtotal;
        $raport[] = ['id' => $p['id'], 'subtotal_bani' => $subtotal];
    }
    return ['pozitii' => $raport, 'total_bani' => $total];
}
```

Suma este verificată înainte de adunare deoarece două subtotaluri valide separat pot depăși împreună domeniul întregului. Nu este suficient să protejăm numai înmulțirea. Condiția folosește faptul că subtotalurile sunt nenegative; dacă am accepta și ajustări negative, contractul și verificarea depășirii ar necesita alte cazuri. O presupunere importantă trebuie legată de validarea care o garantează.

Urmărește contribuțiile, nu doar totalul

Pentru poziția A cu preț 1250 și cantitate 2, subtotalul este 2500. Pentru B cu preț 300 și cantitate 3, subtotalul este 900. Raportul trebuie să conțină ambele id-uri în ordinea intrării și totalul 3400. Dacă verifici numai 3400, ai putea rata o implementare care schimbă id-urile între rânduri. Raportul explicabil cere atât suma, cât și asocierea fiecărei contribuții.

Pentru lista goală, funcția întoarce poziții goale și total zero. Acesta este un rezultat valid în contractul raportului, chiar dacă o etapă ulterioară, precum finalizarea comenzii, poate cere cel puțin o poziție. Nu adăuga această regulă ulterioară în funcția de agregare fără să decizi explicit că îi aparține. Funcțiile ușor de refolosit au limite pe care le poți descrie într-o propoziție.

Testează ce nu trebuie să se modifice

Salvează o copie a intrării înainte de apel și compar-o strict cu intrarea după apel. Raportul trebuie să fie nou, iar sursa să rămână identică. Această verificare detectează o transformare accidentală prin referință sau o suprascriere a câmpurilor originale. Dacă prețul din sursă este înlocuit cu subtotalul, o a doua execuție poate produce alte valori chiar fără schimbarea cerinței.

Rulează funcția de două ori pe aceeași intrare și compară rezultatele. Pentru acest calcul fără efecte externe trebuie să obții același raport. Dacă nu, caută variabile globale, acumulate păstrate între apeluri sau modificări ascunse ale sursei. Proprietatea de a produce același rezultat pentru aceeași intrare face diagnosticarea mult mai simplă decât urmărirea unei stări globale modificate din mai multe locuri.

Duplicații, identitate și normalizare

În contractul de aici, A și a sunt identificatori diferiți. La fel, spațiile din interiorul unui id nu sunt eliminate automat. Dacă domeniul produsului cere alte reguli, normalizează identitatea înainte de verificarea unicității și documentează transformarea. Nu adăuga `strtolower` doar pentru că pare o curățare utilă: poate uni identificatori care trebuiau păstrați separați.

Testul cu id duplicat trebuie să aibă două poziții cu același id, dar valori diferite. Astfel demonstrezi că funcția respinge ambiguitatea și nu păstrează arbitrar prima sau ultima valoare. Rezultatul așteptat este eroare, fără un raport parțial acceptat. În aplicație, apelantul poate prezenta eroarea utilizatorului sau poate opri importul pentru corectarea sursei.

Extensie: filtrare după agregare sau înainte de agregare

Adaugă un câmp activ și decide dacă pozițiile inactive trebuie validate. Pentru politica discutată în lecție, toate pozițiile sunt validate înainte ca o selecție să le excludă din raportul final. Dacă filtrezi întâi și validezi după aceea, un rând inactiv defect rămâne ascuns până când cineva îl activează. Aceasta poate fi o alegere deliberată în alt sistem, dar nu este echivalentă cu validarea întregului lot.

La predare, descrie un caz care diferențiază cele două politici: o poziție inactivă cu preț textual invalid. Scrie apoi comportamentul ales și motivul lui. Exercițiul te învață că ordinea operațiilor exprimă reguli, nu doar preferințe de sintaxă. Un raport de încredere poate explica de ce a inclus sau a respins fiecare rând, precum și cum a obținut suma finală.

Structuri de control

Laborator suplimentar: toate rezultatele și primul rezultat

Pornim de la un catalog validat intern. Fiecare produs are id textual, activ boolean și pret_bani întreg nenegativ. Funcția de selecție nu este granița de validare a unor formulare arbitrare; presupune acest contract al catalogului și verifică separat plafonul primit. Separarea ne permite să concentrăm exercițiul pe ramuri și pe oprirea buclei.

```
<?php
declare(strict_types=1);
function selecteazaProduce(array $produse, int $plafonBani): array {
    if ($plafonBani < 0) {
        throw new InvalidArgumentException('Plafon negativ');
    }
    $rezultat = [];
    foreach ($produse as $produs) {
        if (!$produs['activ']) { continue; }
        if ($produs['pret_bani'] <= $plafonBani) {
            $rezultat[] = $produs;
        }
    }
    return $rezultat;
}

function primulProduce(array $produse, int $plafonBani): ?array {
    if ($plafonBani < 0) {
        throw new InvalidArgumentException('Plafon negativ');
    }
    foreach ($produse as $produs) {
        if ($produs['activ'] && $produs['pret_bani'] <= $plafonBani) {
            return $produs;
        }
    }
    return null;
}
```

Prima funcție termină parcurgerea întregului catalog și întoarce o listă. A doua se oprește la primul rezultat și întoarce un singur produs sau null. Aceste tipuri de rezultat sunt diferite și îi spun apelantului cum să continue. Nu întoarce când listă, când produs, când text de eroare din aceeași funcție fără un contract clar; interfața ar trebui atunci să ghicească ce a primit.

Return din interiorul buclei oprește întreaga funcție. Break ar opri bucla și ar permite executarea instrucțiunilor de după ea. În prima funcție niciunul nu este potrivit după prima potrivire, deoarece cerința cere toate produsele. Această distincție poate fi verificată cu două produse eligibile: un singur rezultat nu este suficient pentru a detecta o oprire prematură.

Construiește cazurile în jurul condițiilor

Folosește A activ la 9000 bani, B inactiv la 4000 bani și C activ la 12000 bani. La plafonul 10000, lista conține numai A. La plafonul 13000, conține A și C. La plafonul 9000, A trebuie inclus, deoarece egalitatea este permisă. La plafonul 0, lista este goală pentru acest set de date, dar nu înseamnă că funcția respinge toate produsele gratuite.

Adaugă D activ la zero bani și repetă plafonul zero. Acum D trebuie selectat. Testul diferențiază regula corectă de o condiție vagă care verifică prețul ca boolean. Adaugă apoi E inactiv la zero bani: el trebuie exclus. Cele două exemple arată că gratuitatea și disponibilitatea sunt axe distincte ale deciziei, chiar dacă ambele apar în aceeași expresie.

Pentru primulProduce, schimbă ordinea produselor eligibile și observă schimbarea rezultatului. Nu este o eroare: funcția promite primul din ordine, nu cel mai ieftin. Dacă produsul cere cel mai ieftin, trebuie schimbate numele, regula și testele. O denumire precisă te împiedică să atribui unei funcții o proprietate pe

care nu a implementat-o.

Rezolvă o condiție de acces cu tabelul adevărului

Regula demonstrativă spune că un document trebuie să fie nearhivat și persoana să fie proprietar sau membru. Sunt trei valori booleene, deci poți verifica toate cele opt combinații. Începe cu documentul arhivat: indiferent de roluri, rezultatul trebuie să fie false. Pentru documentul nearhivat, lipsa ambelor roluri produce false, iar prezența oricăruia produce true.

```
<?php
declare(strict_types=1);
function poateVedeo(bool $arhivat, bool $proprietar, bool $membru): bool {
    return !$arhivat && ($proprietar || $membru);
}
```

Elimină temporar parantezele și scrie expresia cu && lângă proprietar, urmată de || membru. Testul arhivat true, proprietar false, membru true va detecta diferența. Scopul nu este memorarea mecanică a precedentei, ci demonstrarea unei cerințe printr-un caz care ar eșua dacă gruparea ar fi greșită. Păstrează parantezele în forma finală pentru ca regula să fie vizibilă.

Urmărește o buclă fără să o lași să ruleze la nesfârșit

Pentru o buclă while, scrie pe hârtie starea înainte de condiție, ramura aleasă și starea înainte de iterația următoare. Dacă două iterații consecutive au aceeași stare și nu există o sursă externă care o schimbă, bucla poate repeta același traseu. Într-un exemplu cu contor, identifică exact instrucțiunea care produce progresul și verifică dacă un continue o poate sări.

O corectură utilă pentru parcurgerea unei liste este folosirea foreach, dacă nu ai nevoie de control manual asupra indexului. Nu toate buclele trebuie convertite în foreach: citirea unei resurse până la terminare poate avea alt model. Alegerea structurii trebuie să reflecte sursa pașilor. Dacă bucla este mai greu de explicat decât problema, simplifică reprezentarea înainte să adaugi alte condiții.

Criterii de predare și interpretarea eșecurilor

Predă rezultatele pentru limita exactă, produsul inactiv, produsul gratuit, lista goală și plafonul negativ. Primele patru sunt selecții valide, iar ultimul este eroare de intrare. Dacă ai un rezultat gol, verifică dacă acesta era așteptat înainte să tratezi situația drept defect. O funcție de selecție corectă poate întoarce zero elemente fără ca programul să fi eșuat.

Adaugă cele opt combinații ale regulii de acces și explică un caz care detectează parantezele greșite. În final, compară rezultatul funcției care selectează toate produsele cu cel al funcției care alege primul. Această comparație încheie laboratorul cu două idei verificabile: condițiile definesc cine intră în rezultat, iar instrucțiunile de oprire definesc cât de mult din sursă este examinat.

Formulare și date de la utilizator

Laborator suplimentar: testează funcția fără formular

Păstrează valideazaOferta în validate.php exact ca în lecție. Construiește un script separat care o apelează cu array-uri fictive. Astfel poți verifica limitele chiar dacă browserul ar împiedica introducerea datelor respective. Un input cu maxlength nu este un mod suficient de a testa ce se întâmplă când serverul primește un mesaj mai lung decât limita.

Pentru cazul de bază folosește emailul ana@example.test și mesajul „Vreau o ofertă”. Rezultatul are erori goale și valori normalizate. Introdu apoi spații la marginile ambelor câmpuri. Valorile acceptate trebuie să fie aceleași după trim. Acest comportament este o decizie a formularului nostru; nu înseamnă că toate câmpurile din orice aplicație trebuie normalizate identic.

```
<?php
declare(strict_types=1);
require __DIR__ . '/validate.php';
$cazuri = [
    ['ana@example.test', 'Vreau o ofertă'],
    ['ana', 'Vreau o ofertă'],
    [['ana@example.test'], 'Vreau o ofertă'],
    ['ana@example.test', str_repeat('ț', 1000)],
    ['ana@example.test', str_repeat('ț', 1001)],
    ['ana@example.test', ' '],
];
foreach ($cazuri as $index => [$email, $mesaj]) {
    $r = valideazaOferta(['email' => $email, 'mesaj' => $mesaj]);
    echo $index . ': ' . implode(',', array_keys($r['erori'])) . "\n";
}
```

Liniile pentru cazurile zero și trei nu au nume de câmp după separator: acestea sunt acceptate. Cazurile unu și doi indică email, iar cazurile patru și cinci indică mesaj. Testele cu o mie și o mie unu de litere Ț verifică atât limita, cât și folosirea unei funcții potrivite pentru UTF-8. Dacă ai folosi strlen pentru limită, ai număra octeți și ai respinge prea devreme acest text.

Testează două erori în aceeași cerere

Trimite simultan email invalid și mesaj gol. Funcția trebuie să returneze erori pentru ambele câmpuri. Dacă se oprește după prima eroare, poate respecta o altă politică de validare, dar nu comportamentul promis aici. Avantajul colectării este că utilizatorul poate corecta formularul într-o singură încercare, fără să descopere câte un câmp greșit după fiecare trimitere.

Verifică și absența completă a unui câmp, valoarea null și un array în locul textului. Nu toate aceste cazuri trebuie să aibă mesaje diferite pentru utilizator, însă niciunul nu trebuie să ajungă necontrolat în trim sau mb_strlen. Forma datelor este verificată înaintea conținutului. O eroare de structură nu devine adresă validă prin conversia forțată a unui array la string.

Verifică escaparea în două contexte HTML

Mesajul este afișat atât în textarea, cât și în previzualizarea din pre. Folosește ca text de laborator o închidere de textarea urmată de un element simplu, de exemplu </textarea>test. În răspuns, aceste caractere trebuie reprezentate prin entități; în interfață trebuie să apară literal. Nu trebuie să apară un element b nou creat din mesajul utilizatorului.

Nu schimba validatorul ca să elimine toate semnele unghiulare doar pentru a face testul să pară sigur. Contractul formularului acceptă text liber. Responsabilitatea afișării este să păstreze textul ca text în contextul HTML. Dacă produsul ar accepta HTML formatat, ar fi necesară o politică separată de sanitizare, cu elemente și attribute permise, nu simpla eliminare a câtorva șiruri cunoscute.

În câmpul email, valoarea apare într-un atribut între ghilimele. Funcția `h` folosește `ENT_QUOTES` și codificarea explicită. Nu elimina ghilimelele atributului pentru a scurta șablonul și nu muta aceeași valoare direct într-un script JavaScript presupunând că escaparea HTML rămâne suficientă. Fiecare context are propriile reguli, iar acest exemplu testează numai contextele HTML folosite efectiv de formular.

Leagă statusul HTTP de rezultatul vizibil

Deschide fila de rețea a browserului înainte să trimiți formularul. Pentru GET, răspunsul este reușit și nu conține confirmarea unei previzualizări. Pentru POST valid, răspunsul este reușit și conține textul care precizează că nimic nu a fost salvat. Pentru POST invalid, răspunsul are 422 și păstrează formularul cu mesajele pe câmpuri. Codul de status și conținutul trebuie să descrie aceeași situație.

Trimite o metodă neacceptată cu un client HTTP de test și verifică 405 împreună cu Allow. Dacă endpointul răspunde cu formularul obișnuit la orice metodă, contractul nu este aplicat. Testarea aceasta nu necesită date reale și nu trebuie făcută împotriva unor servicii externe: serverul local al laboratorului este suficient pentru a observa răspunsul controlat.

De ce previzualizarea nu este încă o trimitere de ofertă

Un formular poate arăta complet și totuși să nu aibă stocare sau livrare. Textul butonului și mesajul de succes trebuie să corespundă operației efective. Dacă utilizatorul vede „Ofertă trimisă”, dar programul doar afișează datele primite, interfața promite ceva ce nu s-a întâmplat. Laboratorul numește explicit previzualizarea tocmai pentru a păstra această diferență observabilă.

Pentru etapa următoare, scrie ce ar trebui adăugat înainte de salvare: protecția cererii, identitatea și drepturile când sunt necesare, stocarea parametrizată și un răspuns care confirmă rezultatul real. Nu pune pur și simplu un INSERT după afișarea formularului. Operația trebuie să fie într-o ramură executată doar după toate verificările și trebuie să trateze și eventualul eșec al bazei.

Rezolvarea unei regresii introduse la refactorizare

Presupune că muți funcția de validare într-un fișier nou și schimbi accidental cheia mesaj în text. Formularul continuă să trimită mesaj, dar validatorul nu îl mai găsește. Testul de bază va eșua înainte să investighezi HTML-ul sau serverul. Aceasta este valoarea unui contract verificat separat: eroarea este localizată la legătura dintre numele câmpurilor și funcția care le interpretează.

La predare păstrează rezultatele testelor de funcție și ale cererilor HTTP în două grupuri. Primele verifică regulile datelor; celelalte verifică integrarea cu metoda, statusul și marcajul. Dacă funcția trece testele, dar pagina nu, caută includerea fișierului, numele câmpurilor și ramura POST. Dacă ambele eșuează pentru aceeași intrare, corectează regula înainte să ajustezi prezentarea.

Sesiuni și autentificare simplă

Laborator suplimentar: documentează viața unei sesiuni

Pregătește proiectul local exact în structura descrisă în lecție și generează contul fictiv o singură dată. Deschide instrumentele de rețea înainte de prima cerere. Scopul este să legi fiecare schimbare vizibilă de o cerere și un răspuns, nu să deduci starea doar din faptul că pagina afișează numele Anei. O interfață poate afișa acel nume din greșeală chiar fără verificarea identității.

La primul GET notează existența unui cookie, fără să copiezi valoarea în documente partajate. Trimite apoi o parolă greșită cu tokenul primit în formular. Răspunsul trebuie să păstreze starea anonimă. O nouă cerere către cont.php trebuie să fie respinsă. Dacă pagina principală arată eroarea, dar pagina protejată este accesibilă, verificarea autentificării a produs totuși un efect înainte de a valida complet datele.

Separă cele patru intrări ale verificării

Un test de autentificare implică emailul, parola, cookie-ul și tokenul formularului. Schimbă numai una dintre ele la un moment dat. Pentru parola greșită păstrezi emailul, cookie-ul și tokenul corecte. Pentru token greșit păstrezi datele contului corecte. Pentru sesiune independentă folosești alt depozit de cookie-uri și formularul obținut de acel client, nu un token copiat din prima sesiune.

Această disciplină împiedică rezultate ambigue. Dacă trimiți simultan o parolă greșită și un token greșit, răspunsul 403 demonstrează verificarea tokenului, dar nu spune nimic despre verificarea parolei. Codul s-a oprit înainte să ajungă la ea. Un caz de test trebuie să precizeze ce condiție izolează și ce rezultat ar demonstra că acea condiție nu este aplicată.

Verifică schimbarea identității sesiunii

Salvează temporar identificatorul anonim într-un client local de test, apoi autentifică-te corect. Răspunsul de succes trebuie să fie o redirectionare 303 și să trimită cookie-ul noii sesiuni. Compară identificatorii doar local: trebuie să fie diferiți. Cererea GET care urmează folosește noul cookie și afișează contul. Aceste observații verifică tranziția, nu doar destinația finală.

Încearcă pagina protejată cu vechiul identificator anonim, într-un client separat. Nu trebuie să obții identitatea autentificată. În laborator, sesiunea veche este eliminată la regenerare; un identificator necunoscut este tratat potrivit modului strict configurat. Într-o implementare de producție, verificarea tranziției trebuie adaptată strategiei documentate pentru sesiuni vechi și cereri concurente, fără să lași sesiunea anterioară să dobândească privilegiile noi.

Un cookie nu este un rol trimis de utilizator

Adaugă într-o cerere un câmp `user_id=1` sau `rol=admin`, fără autentificare validă. Pagina protejată trebuie să refuze accesul. Codul nostru nu copiază aceste câmpuri în sesiune. Identitatea este stabilită numai după verificarea contului. Acest experiment arată de ce o atribuire precum copierea întregului POST în SESSION ar fi o greșeală de proiectare.

Nu confunda cookie-ul de sesiune cu un cookie creat de tine care conține textul admin. Primul este o cheie către starea controlată de server; al doilea este o afirmație a clientului, dacă nu există un mecanism valid de integritate și verificare. Pentru acest curs păstrăm modelul simplu cu sesiune pe server, fără să inventăm un protocol de autentificare în cookie-uri proprii.

Expirarea: exercițiu de logică înaintea integrării

Implementează separat o funcție care decide dacă un interval de inactivitate a expirat. Primește momentul ultimei activități, momentul curent și limita în secunde. Pentru contractul nostru, egalitatea cu limita înseamnă expirare, iar un moment curent anterior ultimei activități este invalid. Alegerea explicită a graniței evită diferențele între pagini care folosesc când mai mare, când mai mare sau egal.

```
<?php
declare(strict_types=1);
function sesiuneExpirata(int $ultima, int $acum, int $limita): bool {
    if ($ultima < 0 || $acum < $ultima || $limita < 1) {
        throw new InvalidArgumentException('Interval invalid');
    }
    return $acum - $ultima >= $limita;
}
```

Testează ultima 1000, limita 60 și momentele 1059, 1060 și 1061. Rezultatele sunt false, true și true. Funcția nu citește direct ceasul sistemului, astfel încât testele nu depind de așteptarea unui minut real. La integrare, serverul furnizează timpul curent și păstrează ultima activitate în sesiune; clientul nu poate trimite o dată proprie care să prelungească drepturile.

Funcția singură nu implementează expirarea în pagina principală a laboratorului. Pentru integrarea ca exercițiu, verifică termenul înainte de acceptarea identității pe fiecare pagină protejată, apoi actualizează activitatea numai dacă sesiunea este încă validă. Dacă actualizezi mai întâi timpul, vei compara aproape mereu momentul curent cu el însuși și sesiunea nu va expira niciodată din inactivitate.

Logout repetat și încercarea cu un cookie vechi

După o autentificare reușită, obține tokenul curent și trimite logout. Cererea este POST; deschiderea unui link GET nu trebuie să execute ieșirea din cont. Verifică apoi pagina protejată atât cu clientul actual, cât și cu o copie locală a cookie-ului anterior logoutului. Niciunul nu trebuie să primească acces. Testul verifică invalidarea serverului, nu numai dispariția cookie-ului dintr-un browser.

Repetarea exactă a POST-ului vechi poate fi respinsă deoarece sesiunea și tokenul nu mai sunt valabile. Acest rezultat nu înseamnă că utilizatorul a rămas autentificat; verifică separat pagina protejată. Nu interpreta fiecare răspuns de eroare ca eșec al logoutului. Contractul relevant este că identitatea veche nu mai acordă acces după operația reușită.

Predarea și limitele demonstrației

Raportul final conține starea inițială, parola greșită, tokenul greșit, autentificarea corectă, accesul din altă sesiune, logoutul și reutilizarea cookie-ului vechi. Pentru fiecare notează metoda, statusul și rezultatul vizibil, fără valori secrete. Include și rezultatele pure ale funcției de expirare, etichetate separat de fluxul HTTP. Astfel nu pretinzi că o funcție testată izolat este deja integrată în toate paginile.

Încheie prin identificarea funcțiilor pe care laboratorul nu le acoperă: recuperarea parolei, blocarea sau limitarea încercărilor și gestionarea sesiunilor de pe mai multe dispozitive. Acestea sunt cerințe distincte ale unui sistem real, fiecare cu propriile teste. Înțelegerea lor te ajută să folosești corect autentificarea unui framework și să nu consideri un formular care acceptă o parolă drept întregul sistem de conturi.

PDO: prima interogare

Laborator suplimentar: funcția de citire autorizată

Refactorizează interogarea de detaliu într-o funcție care returnează array sau null. Conexiunea este un parametru, astfel încât testul poate folosi o bază izolată, iar pagina poate folosi conexiunea aplicației. Funcția nu citește direct GET sau SESSION și nu afișează HTML. Ea primește identificatori interni validați și aplică regula că rândul trebuie să aparțină utilizatorului indicat.

```
<?php
declare(strict_types=1);
function citesteCerere(PDO $db, int $id, int $utilizator): ?array {
    if ($id < 1 || $utilizator < 1) {
        throw new InvalidArgumentException('Identificatori invalizi');
    }
    $stmt = $db->prepare(
        'SELECT id, titlu, stare FROM cereri
        WHERE id = :id AND proprietar_id = :utilizator'
    );
    $stmt->execute(['id' => $id, 'utilizator' => $utilizator]);
    $rand = $stmt->fetch(PDO::FETCH_ASSOC);
    return $rand === false ? null : $rand;
}
```

În aplicație, valoarea utilizatorului trebuie să provină din identitatea verificată. Numele parametrului nu poate garanta această proveniență, deci verifică și apelantul. O funcție cu filtrul corect poate fi folosită greșit dacă pagina îi transmite proprietarul din URL. Testul de integrare trebuie să demonstreze că schimbarea unui parametru al cererii nu schimbă identitatea cu care este apelată funcția.

Setul minim de date care detectează accesul greșit

Folosește cele trei cereri din inițializare. Pentru utilizatorul 1, identificatorii 1 și 2 produc rânduri, iar 3 și 999 produc null. Pentru utilizatorul 2, identificatorul 3 produce rândul lui Mihai, iar 1 produce null. Acest set include atât succes, cât și refuz în ambele sensuri. Dacă ai testa numai utilizatorul 1 cu cererea 1, o funcție fără filtrul proprietarului ar trece.

Compară și titlul, nu doar faptul că rezultatul este array. O funcție care returnează primul rând disponibil indiferent de id ar putea trece un test prea vag. Verifică identificatorul, titlul și starea așteptate. Testele bune separă erori posibile: ignorarea id-ului, ignorarea proprietarului și confundarea rezultatului absent cu un rând gol.

Demonstrează că valorile nu schimbă structura SQL

Într-un test separat al parametrizării, caută un titlu egal cu textul fictiv x' OR 1=1 -- folosind un marcator. Nu inserezi acest text în SQL; îl transmiți la execute ca valoare a titlului. Cum niciun rând din set nu are acel titlu, rezultatul trebuie să fie gol. Testul arată că semnele sunt tratate ca date, fără să transforme condiția într-una care returnează toate rândurile.

Nu folosi acest test ca dovadă universală că orice cod din proiect este protejat. El verifică interogarea construită în acel mod. Dacă altă parte concatenează o coloană sau un fragment ORDER BY din intrare, trebuie analizată separat. Pentru sortare, definește o mapare internă între opțiunile publice și fragmentele SQL permise și respinge opțiunile necunoscute.

Exercițiu de rollback cu rezultat observabil

Pornește de la o bază de laborator proaspătă. Începe tranzacția, inserează id 10 cu stare noua, apoi încearcă id 11 cu starea necunoscuta. Constrângerea CHECK va respinge a doua inserare. În blocul catch verifică dacă tranzacția este activă, execută rollBack și apoi numără rândurile cu id 10 și 11 într-o interogare nouă. Rezultatul așteptat este zero pentru ambele.

```
<?php
declare(strict_types=1);
$db = require __DIR__ . '/conexiune.php';
$stmt = $db->prepare(
    'INSERT INTO cereri (id, proprietar_id, titlu, stare)
    VALUES (:id, 1, :titlu, :stare)'
);
$anulat = false;
$db->beginTransaction();
try {
    $stmt->execute(['id' => 10, 'titlu' => 'Temporar', 'stare' => 'noua']);
    $stmt->execute(['id' => 11, 'titlu' => 'Invalid', 'stare' => 'necunoscuta']);
    $db->commit();
} catch (PDOException $e) {
    if ($db->inTransaction()) { $db->rollBack(); }
    $anulat = true;
}
$numar = (int) $db->query('SELECT COUNT(*) FROM cereri WHERE id IN (10, 11)')->fetchColumn();
var_dump($anulat, $numar); // true, 0
```

Citirea de control este executată după ieșirea din tranzacție. Dacă ai verifica doar că a fost prinsă o excepție, nu ai demonstra anularea primei inserări. Dacă ai verifica doar totalul final al tabelului, ai putea ascunde o modificare compensată de alta. Căutarea identificatorilor temporari leagă rezultatul exact de operațiile pe care ai intenționat să le anulezi.

Constrângerea unică și cererile repetate

Cheia primară împiedică două rânduri cu același id. Încearcă într-un test separat inserarea unei alte cereri cu id 1. Baza trebuie să respingă duplicatul, iar rândul original trebuie să rămână neschimbat. Această verificare arată rolul constrângerii, însă nu este încă o implementare completă de idempotență pentru un formular care generează identificatori noi la fiecare POST.

Pentru deduplicarea unei operații comerciale ai nevoie de un identificator al operației și de o regulă explicită de unicitate potrivită produsului. Două cereri cu titluri identice pot fi legitime, deci titlul nu este automat o cheie de deduplicare. Nu adăuga UNIQUE pe un câmp doar pentru a opri un test care dublează datele; mai întâi stabilește ce reprezintă identitatea reală a operației.

Integrarea cu HTML și rezultatul final

După ce funcția de citire trece testele, pagina verifică sesiunea, validează id-ul, apelează funcția și decide răspunsul. Pentru un rând valid, titlul este escapezat la afișare. În baza de date rămâne textul original Catalog <primăvară>, iar în sursa HTML apar entități. Nu aplica escaparea HTML parametrilor SQL și nu considera parametrizarea SQL suficientă pentru ieșirea în pagină.

Predă rezultatele citirilor pentru ambii proprietari, cazul absent, testul valorii cu semne SQL, rollback-ul și duplicatul respins. Include un exemplu de titlu afișat literal în HTML. Acest set verifică responsabilități diferite și îți permite să explici unde ar trebui investigată o eroare: la validarea intrării, la filtrul de acces, la instrucțiunea SQL, la tranzacție sau la reprezentarea de ieșire.