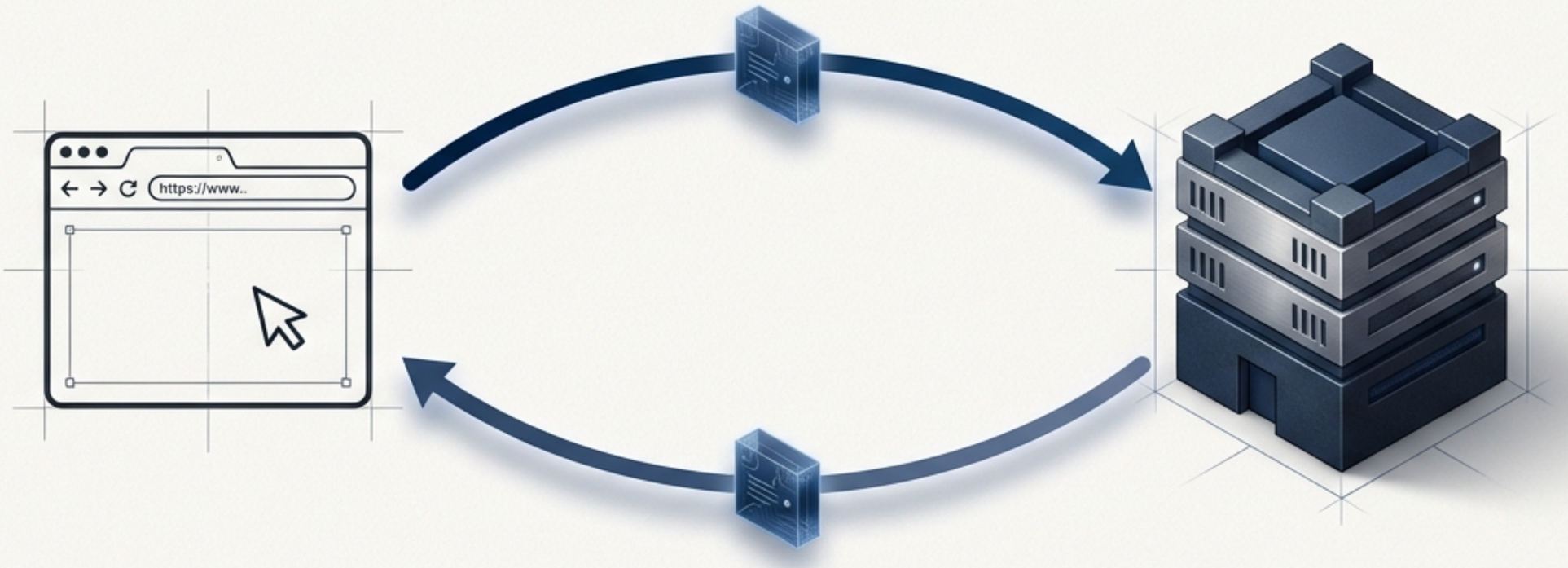


Սերվերի Մտածելակերպը

Մեկ վեբ հարցման ճանապարհորդությունը՝ քլիքից մինչև բովանդակություն

Սերվերը երբեք Չի Վստահում

Սերվերային ծրագրավորումը ոչ թե կոդ գրելն է, այլ պատասխանատվություն, ստուգում և այն ենթադրությունը, որ օգտատիրոջ կողմից ուղարկված ցանկացած տվյալ պոտենցիալ վտանգավոր է:

Client-side ստուգումը (JavaScript)

նախատեսված է օգտատիրոջ հարմարավետության (UX) համար, մինչդեռ մինչդեռ server-side ստուգումը՝ անվտանգության:

Կարևոր մեջբերում

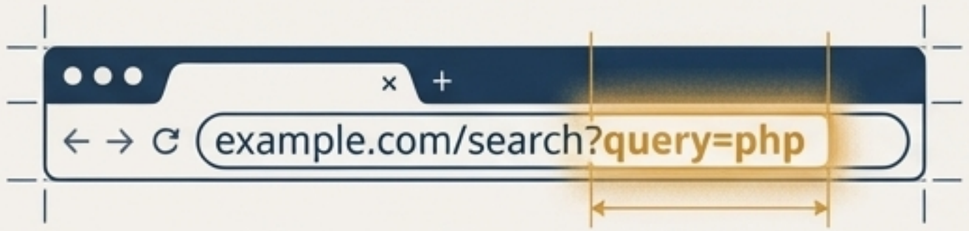
«Սերվերը երբեք չի վստահում օգտատիրոջ ուղարկած տվյալներին»:

Իմաստը

Օգտատերը կարող է ամբողջությամբ շրջանցել ձեր վեբ էջը և ուղարկել հարցումներ ուղիղ սերվերին՝ օգտագործելով այլ գործիքներ, ինչը client-side ստուգումը դարձնում է անօգուտ՝ անվտանգության տեսանկյունից:

Հարցման Երկու Դեմքը՝ GET vs. POST

GET (Տվյալներ ստանալու համար)



Նպատակը: ռեսուրսից տվյալներ պահանջել:

Տվյալների փոխանցում: պարամետրերը երևում են URL-ի մեջ:

Անվտանգություն: քիչ անվտանգ, քանի որ տվյալները տեսանելի են բոլորին:

Քեշավորում: հարցումները կարող են քեշավորվել:

POST (Տվյալներ ուղարկելու համար)

Նպատակը: սերվերին տվյալներ ուղարկել՝ նոր ռեսուրս ստեղծելու կամ փոփոխելու համար:

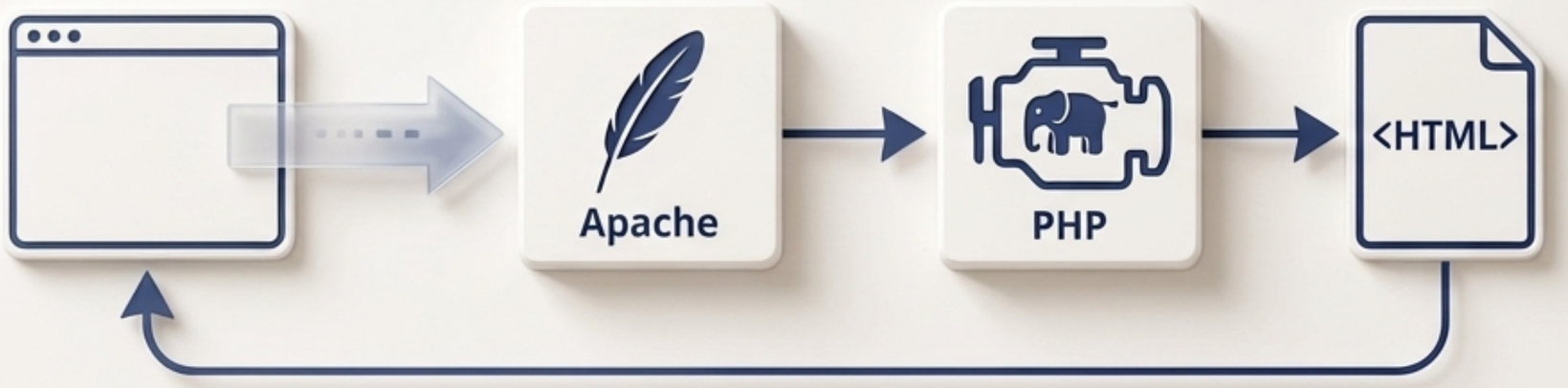
Տվյալների փոխանցում: պարամետրերը թաքցված են հարցման «մարմնում» (request body):

Անվտանգություն: ավելի անվտանգ, քանի որ տվյալները URL-ում չեն երևում:

Քեշավորում: հարցումները չեն քեշավորվում:

📌 **Հիմնական կանոն.** Login form-երը և գաղտնի տվյալները **պարտադիր** պետք է օգտագործեն POST մեթոդը:

Դարպասը՝ ինչպես է Apache-ը «խոսում» PHP-ի հետ

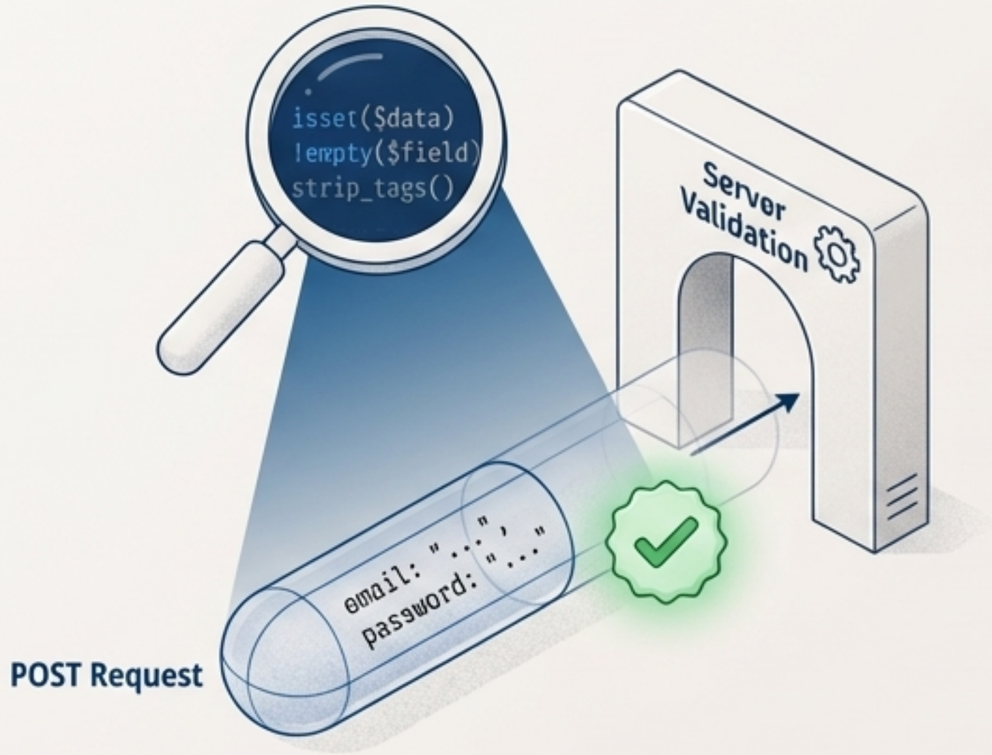


Բրաուզերը երբեք ուղիղ չի դիմում PHP ֆայլին: Web Server-ը (օրինակ՝ Apache) հանդես է գալիս որպես միջնորդ: Գոյություն ունի 3 հիմնական մեթոդ, որով նրանք փոխազդում են:

1. **Որպես մոդուլ** (mod_php): PHP-ն աշխատում է Apache-ի ներսում: Սա ամենաարագ, սերտ ինտեգրված տարբերակն է:
2. CGI (Common Gateway Interface): Ամեն հարցման համար Web Server-ը **ստեղծում է** PHP-ի նոր պրոցես: Պարզ է իրականացման համար, բայց դանդաղ է և լավ չի մասշտաբավորվում:
3. FastCGI: Հավասարակշռություն արագության և մեկուսացման միջև: PHP-ն աշխատում է որպես առանձին դեմոն (daemon), ինչը վերացնում է ամեն անգամ նոր պրոցես ստեղծելու անհրաժեշտությունը:

Օգտատերը տեսնում է միայն PHP-ի կողմից գեներացված վերջնական HTML-ը, ոչ երբեք՝ PHP կոդը:

Առաջին անցակետը. «Իսկ եթե օգտատերը փորձի խաբե՞լ ինձ»



Խնդիրը

Frontend-ի ստուգումները (օրինակ՝ HTML5 `required՝ ատրիբուտը) կարելի է հեշտությամբ շրջանցել: Հարձակվողը կարող է ուղարկել հարցում՝ առանց բրաուզեր օգտագործելու:

Լուծումը

Server-side ստուգում: Միշտ ստուգեք սերվերի վրա, թե արդյոք տվյալները առկա են և դատարկ չեն:

Կոդի օրինակ (PHP)

```php
// Ստուգում ենք, որ հարցումը POST է
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    // Ստուգում ենք՝ արդյոք email-ը և password-ը ուղարկվել են
    // trim() հեռացնում է ավելորդ բացատները
    $email = trim($_POST['email'] ?? '');
    $pass  = $_POST['password'] ?? '';

    if (empty($email) || empty($pass)) {
        die("Սխալ: Բոլոր դաշտերը պարտադիր են:");
    }
    // ... հետագա գործողություններ
}
```

Բացատրություն

Այս կոդը երաշխավորում է, որ նույնիսկ եթե client-side ստուգումը շրջանցվի, սերվերը չի փորձի աշխատել թերի կամ բացակայող տվյալների հետ:

Պահոցը. Գաղտնաբառերը Տեքստով Չեն Պահվում

Խնդիրը

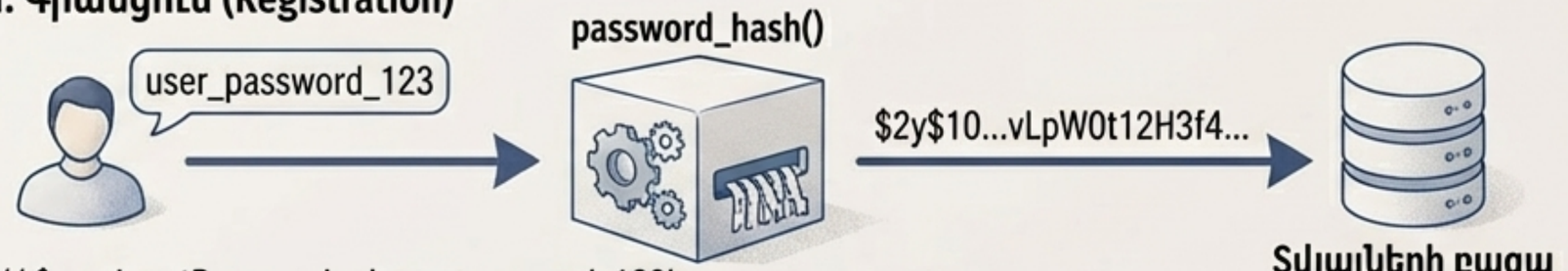
Եթե ձեր տվյալների բազան գողանան, և գաղտնաբառերը պահված լինեն պարզ տեքստով, բոլոր օգտատերերի հաշիվները կվտանգվեն:

Լուծումը

Hashing: Գաղտնաբառը վերածվում է միակողմանի, անվերծանելի տողի (hash):

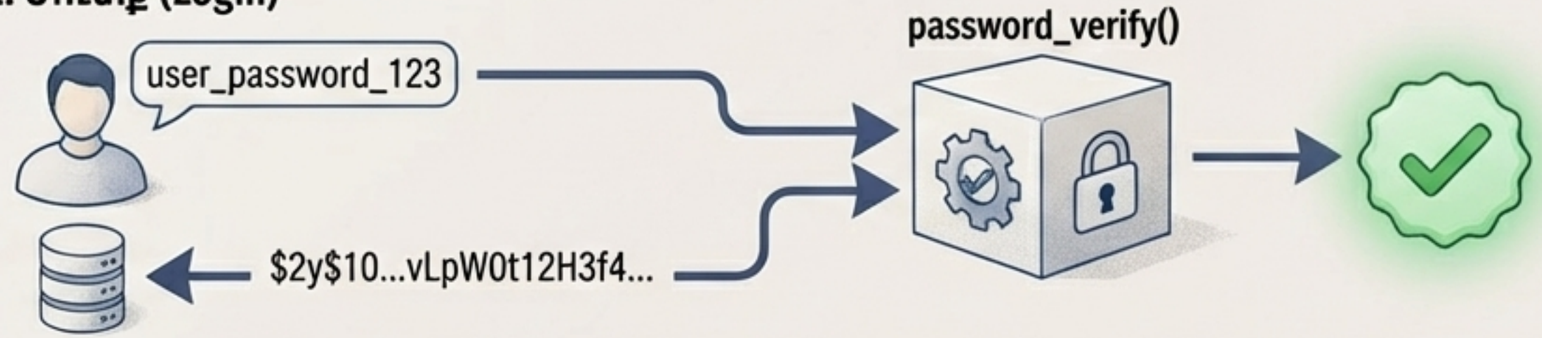
Գործընթացը

1. Գրանցում (Registration)




```
// $userInputPassword = 'user_password_123';
$hashedPassword = password_hash($userInputPassword, PASSWORD_BCRYPT);
// Save $hashedPassword to the database
```

2. Մուտք (Login)




```
// $loginPassword = 'user_password_123';
// $hashFromDb = get_hash_from_database();
if (password_verify($loginPassword, $hashFromDb)) {
    // Password is correct!
}
```

📌 Կարևոր նրբություն

`password_hash()`-ը յուրաքանչյուր անգամ ստեղծում է տարբեր hash նույն գաղտնաբառի համար (շնորհիվ ներդրված "salt"-ի): Այդ պատճառով ուղիղ համեմատություն (`$hash1 === $hash2`) երբեք չի աշխատի: Միշտ օգտագործեք `password_verify()`:

Թարգմանիչը. Պաշտպանություն SQL Injection-ից

Վտանգը

Եթե օգտատիրոջ մուտքագրած տվյալները ուղղակիորեն տեղադրվեն SQL հարցման մեջ, հարձակվողը կարող է փոփոխել հարցման տրամաբանությունը և գողանալ կամ ջնջել տվյալներ:

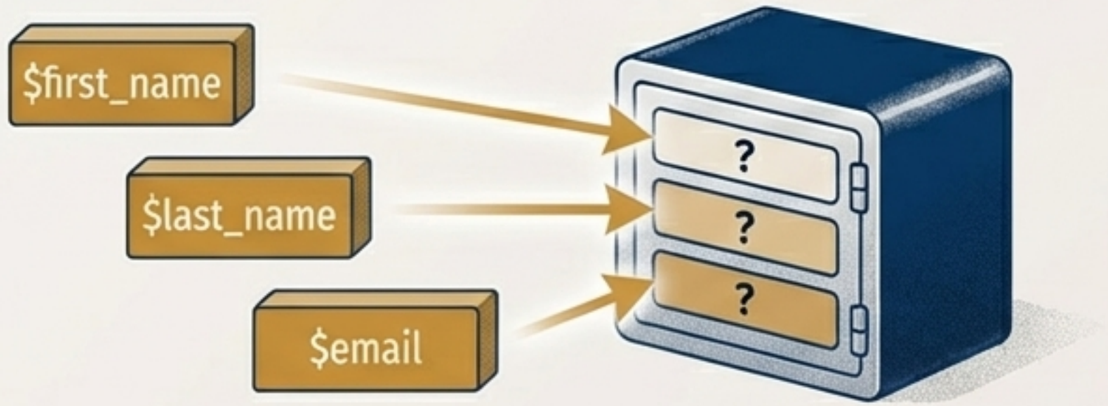
⚠ Վատ օրինակ (ԵՐԲԵՔ ԱՅՍՊԵՍ ՉԱՆԵԼ)

```
$email = $_POST['email'];
$sql = "SELECT * FROM users WHERE email = '$email'"; // ՇԱՏ ՎՏԱՆԳԱՎՈՐ
```

Լուծումը՝ Prepared Statements

Սա երկու քայլից բաղկացած գործընթաց է, որն ամբողջությամբ առանձնացնում է SQL հրամանը օգտատիրոջ տվյալներից:

SQL Հարցման Կաղապար



Տվյալները ապահով տեղադրվում են:

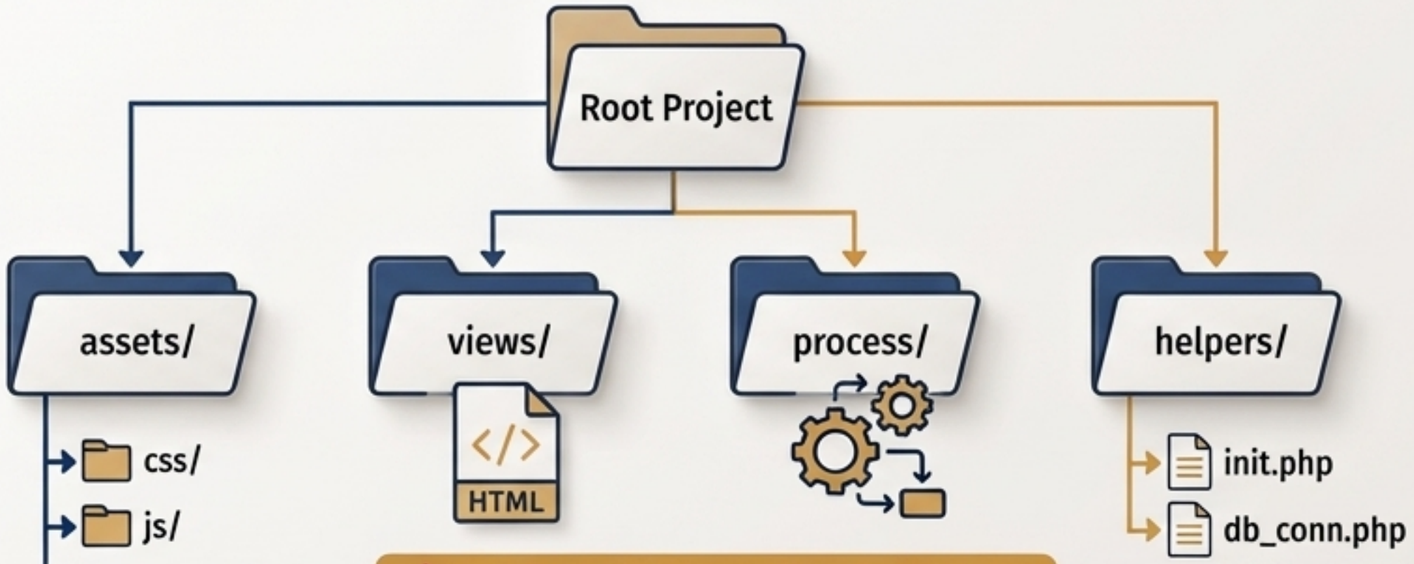
```
$sql = "INSERT INTO persons (first_name, last_name,
       email) VALUES (?, ?, ?)";
$stmt = mysqli_prepare($link, $sql);
mysqli_stmt_bind_param($stmt, "sss", $first_name,
                                     $last_name,
                                     $email);
mysqli_stmt_execute($stmt);
```

Քաոսից դեպի Կառուցվածք

Խնդիրը (Վատ մոտեցում)

Բոլոր ֆայլերը մի թղթապանակում են: Սա դժվարացնում է նախագծի կառավարումը, երբ այն մեծանում է:

Լուծումը (Լավ մոտեցում)



📌 Կյանքը հեշտացնող հնարք

Ստեղծել `**helpers/init.php**` ֆայլ, որը կներառի բոլոր մյուս օգնական ֆայլերը: Այնուհետև, յուրաքանչյուր էջի սկզբում անհրաժեշտ է միայն մեկ `include('helpers/init.php')` տող՝ բոլոր ֆունկցիաներին հասանելիություն ունենալու համար:

«Միշտ ստիպիր քո ներկա «ես»-ին ավելի շատ աշխատել, որպեսզի ապագա «ես»-դ կարողանա հանգստանալ՝ նոր հնարավորություններ ավելացնելիս կամ սխալներ ուղղելիս»:

Սերվերի Դատավճիռը՝ HTTP Status Codes

Սերվերի պատասխանը պարունակում է թվային կոդ, որը նկարագրում է հարցման արդյունքը: Ահա ամենատարածվածները:

2xx (Հաջողություն / Success)

`200 OK`

Ստանդարտ պատասխան հաջող հարցումների համար: Ամեն ինչ նորմալ է:

4xx (Օգտատիրոջ Սխալ / Client Error)

`403 Forbidden`

Դուք չունեք թույլտվություն այս ռեսուրսին հասանելիություն ստանալու համար:

`404 Not Found`

Պահանջված ռեսուրսը չի գտնվել:

5xx (Սերվերի Սխալ / Server Error)

`500 Internal Server Error`

Սերվերի վրա տեղի է ունեցել անսպասելի սխալ: Սա գրեթե միշտ ծրագրավորողի սխալի հետևանք է:

500 Internal Server Error՝ Ծրագրավորողի Սխալը

Սա ընդհանուր «ինչ-որ բան սխալ գնաց» հաղորդագրություն է: Սերվերը գիտի, որ խնդիր կա, բայց չի կարող ավելի կոնկրետ լինել:

Հիմնական պատճառները

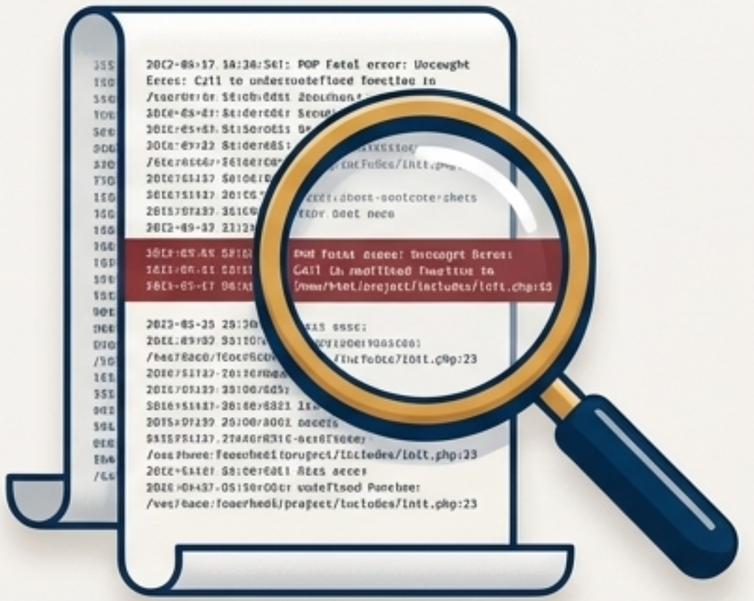
Սինտաքսիսի սխալներ (Syntax Errors): PHP կոդում մոռացված կետ-ստորակետ, փակագիծ և այլն:

Ֆայլերի սխալ թույլտվություններ (Incorrect Permissions): PHP ֆայլերը սովորաբար պետք է ունենան `644`, իսկ թղթապանակները՝ `755`

PHP հիշողության սահմանաչափի սպառում (Memory Limit Exhaustion): Սկրիպտը փորձում է օգտագործել ավելի շատ հիշողություն, քան թույլատրված է սերվերի կողմից:

Սխալներ `.htaccess` ֆայլում: Սխալ կոնֆիգուրացիան կարող է առաջացնել 500 սխալ:

Ի՞նչ անել առաջին հերթին

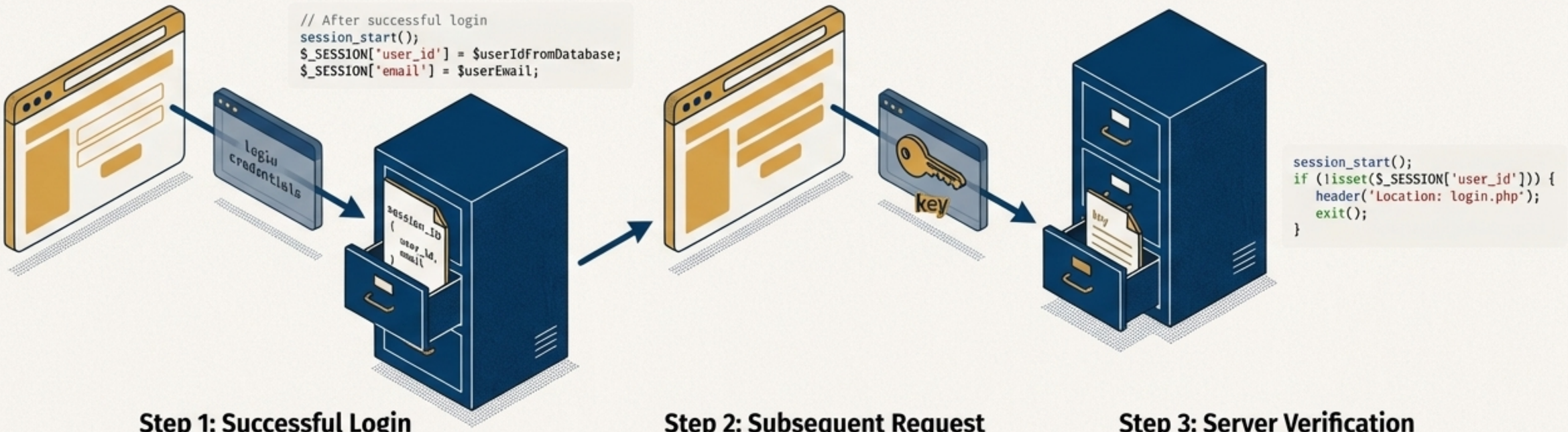


Ստուգեք սերվերի սխալների մատյանը (Error Logs): Սա ամենակարևոր քայլն է: Սխալների մատյանում գրեթե միշտ նշված է, թե որ ֆայլի որ տողում է տեղի ունեցել սխալը:

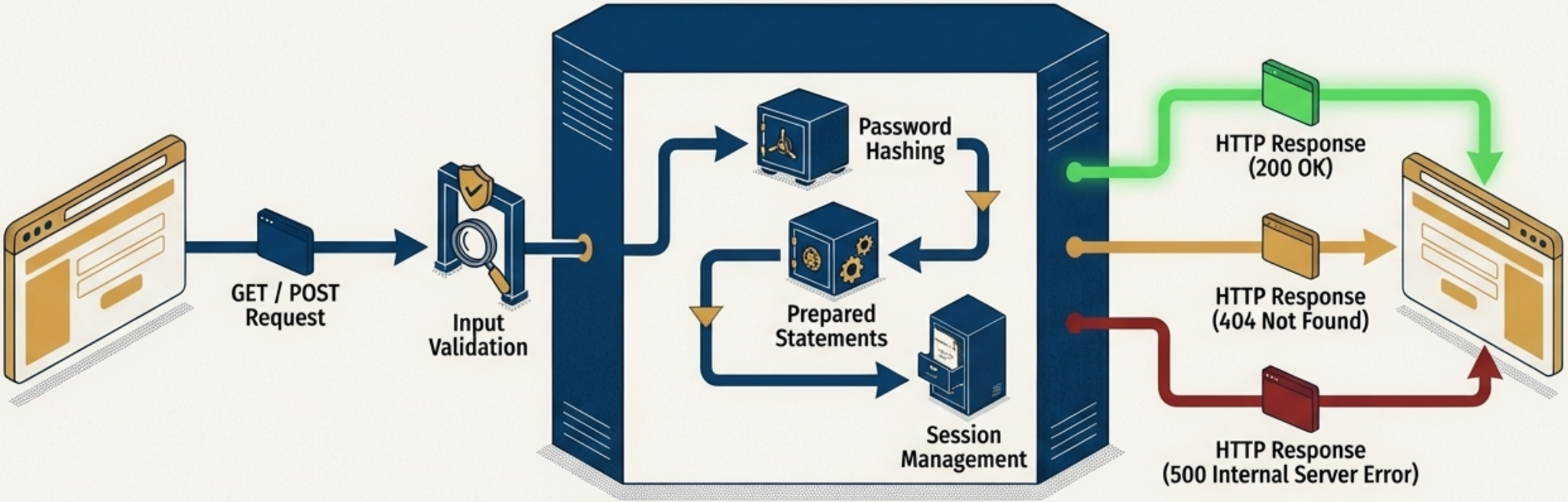
Հիշելով Օգտատիրոջը՝ Session-ներ

Խնդիրը: HTTP պրոտոկոլը «անհիշող» է (stateless): Ամեն հարցում անկախ է նախորդից: Սերվերը լռելյայն չգիտի՝ արդյոք երկու տարբեր հարցումներ եկել են նույն օգտատիրոջից:

Լուծումը: Session-ներ: Սերվերը ստեղծում է ժամանակավոր ֆայլ՝ օգտատիրոջը վերաբերող տվյալները պահելու համար: Բրաուզերին ուղարկվում է յուրահատուկ ID (session cookie), որը բրաուզերն ուղարկում է հետ՝ յուրաքանչյուր հաջորդ հարցման հետ:



Հարցման Կյանքի Ցիկլը. Ամբողջական Պատկերը



Սերվերի հիմնական պարտականություններն են.

- Ընդունել և վերլուծել HTTP հարցումները:
- **ԵՐԲԵՔ** չվստահել մուտքային տվյալներին և միշտ ստուգել դրանք:
- Անվտանգ կերպով աշխատել զգայուն տվյալների հետ (hashing, prepared statements):
- Կառավարել օգտատիրոջ վիճակը (sessions):
- Վերադարձնել հստակ և իմաստալից պատասխան (HTML + status codes):

Տնային Աշխատանք՝ Կառուցեք Ձեր Առաջին Ապահով Login Համակարգը

Նպատակ: Ցույց տալ request-response, validation և session հասկացությունների իմացությունը:

1. Login Form (HTML): email և password դաշտեր, POST մեթոդ:

2. Server-side ստուգում (PHP): Ստուգել, որ դաշտերը դատարկ չեն:

3. Անվտանգ 認証: Օգտագործել password_hash() (գրանցման համար, որը կարող եք նախապես անել) և password_verify() (մուտքի համար):

4. Session-ի կառավարում: Հաջող մուտքի դեպքում ստեղծել session, ստեղծել պաշտպանված էջ (օրինակ՝ dashboard.php), որը հասանելի է միայն մուտք գործելուց հետո, և ավելացնել Logout կոճակ:

⭐ **Բոնուսային մարտահրավեր:**

- Իրականացնել տվյալների բազայի հետ աշխատանք՝ օգտագործելով **Prepared Statements**:
- Իրականացնել վերահղումներ (redirects) **header()** ֆունկցիայի միջոցով:

«Լավ սերվերային
ծրագրավորողը մտածում է.
«Իսկ եթե օգտատերը
փորձի խաբել ինձ»: