

۱. افزودن Database Property به کلاس \$statement

کدی که قبلاً نوشتی:

```
public $statement;
```

✦ با این خط، یک متغیر عضو کلاس (property) تعریف می‌کنیم تا در کل کلاس Database به آبجکت آماده‌شده‌ی PDO (\$this->connection->prepare(...)) دسترسی داشته باشیم.

۲. بازنویسی متد query

```
public function query($query , $params = [])
{
    try {
        $this->statement = $this->connection-
        >prepare($query);
        $this->statement->execute($params);
        return $this;
    }
    catch (Exception $e) {
        echo "خطا در اجرای دستورات query: " . $e-
        >getMessage();
    }
}
```

✦ این متد حالا علاوه‌بر اینکه کوئری رو آماده و اجرا می‌کنه، خودش (آبجکت Database) رو برمی‌گردونه با . return \$this;

یعنی مثلاً وقتی می‌نویسی:

```
$db->query("SELECT * FROM notes")->fetch();
```

اگه query خودش رو برنگردونه، دیگه نمی‌تونی fetch() رو بعدش بزنی.

۳. ساخت متد fetch

```
public function fetch() {  
    return $this->statement->fetch();  
}
```

❖ یعنی حالا می‌تونیم مستقیماً بعد از `query()`، از `fetch()` استفاده کنیم:

```
$note = $db->query("SELECT * FROM notes WHERE id =  
:id", ["id" => $_GET["id"]])->fetch();
```

۴. متد findOrFail

```
public function findOrFail()  
{  
    $result = $this->statement->fetch();  
    if (!$result) {  
        http_response_code(404);  
        header("Location:/laracast-php/view/404.html");  
        exit();  
    }  
    return $result;  
}
```

❖ اگر نتیجه‌ای برنگشت (یعنی هیچ رکوردی در دیتابیس با این id پیدا نشد)، کد ۴۰۴ نشون می‌ده و به صفحه ۴۰۴ هدایت می‌کنه.

۵. تابع Authorize

```
function Authorize($condition, $status = 403)  
{  
    if (!$condition) {  
        http_response_code($status);  
        header("Location:/laracast-php/index.php");  
        exit();  
    }  
}
```

```
}
```

❖ این تابع رو وقتی استفاده می‌کنی که بخوای چک کنی آیا کاربر اجازه داره این اطلاعات رو ببینه یا نه.

مثلاً در: `note.php`

```
Authorize($note['user_id'] == $current_user_id);
```

یعنی اگر صاحب یادداشت نبود، با خطای ۴۰۳ به صفحه اصلی برگردونش.

🔧 ساخت متد `get`

```
public function get() {  
    return $this->statement->fetchAll();  
}
```

❖ برای موقعی که می‌خوای چند رکورد بگیری (مثلاً لیست همه یادداشت‌ها)، به جای `fetch()` از `get()` استفاده کن.

□ مثال نهایی از استفاده:

در فایل: `notes.php`

```
$notes = $db->query("SELECT * FROM notes WHERE user_id  
= :id", ["id" => $current_user_id])->get();
```

در فایل: `note.php`

```
$note = $db->query("SELECT * FROM notes WHERE id =  
:id", ["id" => $_GET["id"]])->findOrFail();  
Authorize($note['user_id'] == $current_user_id);
```

❗ اول یک سوال مهم:

تو قبلاً اینجوری کدت رو نوشته بودی:

```
$statement = $db->prepare("SELECT * FROM notes WHERE id = :id");  
$statement->execute(["id" => $_GET["id"]]);  
$note = $statement->fetch();
```

👉 کار می‌کرد؟ آره. پس چرا عوضش کردیم؟

جواب ساده:

تا مجبور نشی هر بار دوباره بنویسی `prepare, execute, fetch` به‌جاش یه کلاس ساختیم که همه‌ی این مراحل رو نوش قایم کردیم.

□ حالا داستان تغییرات چیه؟

ما یه کلاس `Database` داشتیم. قبلاً فقط وصل می‌شد به دیتابیس. الان گفتیم:

بیایم این کلاس رو باهوش‌تر کنیم که خودش بتونه `query` اجرا کنه، نتیجه رو بگیره، و خطا رو هم مدیریت کنه.

مرحله اول: ساخت متد `query`

```
public function query($query , $params = []) {  
    $this->statement = $this->connection->  
>prepare($query);  
    $this->statement->execute($params);  
    return $this;  
}
```

یعنی:

من) کلاس (Database برات query رو اجرا می‌کنم، و خودم رو برمی‌گردونم تا بعدش هر چی خواستی باهام انجام بدی.

مرحله دوم: متد fetch


```
public function fetch() {  
    return $this->statement->fetch();  
}
```

یعنی:

بیا من همون خط رو که از دیتابیس گرفتم برات می‌دم.

حالا این دو تا رو با هم ترکیب کن:

```
$note = $db->query("SELECT * FROM notes WHERE id =  
:id", ["id" => $_GET["id"]])->fetch();
```

این دقیقاً همونه که قبلاً با ۳ خط می‌نوشتی. 

اما findOrFail چیه؟

```
public function findOrFail()  
{  
    $result = $this->statement->fetch();  
    if (!$result) {  
        http_response_code(404);  
        header("Location:/view/404.html");  
        exit();  
    }  
    return $result;  
}
```

یعنی:

اگه نتیجه‌ای نبود، دیگه نرو ادامه برنامه! برو صفحه ۴۰۴ نشون بده.

مثلاً کاربر اومده id=777 داده که اصلاً وجود نداره. نباید خالی بمونه صفحه، باید بفرستیمش به خطای ۴۰۴.

🔗تابع Authorize برای چیه؟

```
function Authorize($condition, $status = 403) {  
    if (!$condition) {  
        http_response_code($status);  
        header("Location:/index.php");  
        exit();  
    }  
}
```

یعنی:

شرط رو بررسی کن. اگه درست نبود، نذار ادامه بده، بندازش بیرون.

مثلاً کسی داره می‌ره یادداشت تو رو ببینه ولی خودش صاحبش نیست. پس ما می‌گیم:

```
Authorize($note['user_id'] == $current_user_id);
```

□پس جمع‌بندی خیلی ساده:

قبل چی می‌نوشتی؟	حالا چی می‌نویسی؟
prepare + execute + fetch	فقط <code>->query(...)->fetch()</code>
باید شرط عدم وجود رکورد رو دستی می‌نوشتی	حالا <code>->findOrFail()</code> خودش چک می‌کنه
باید دستی بررسی می‌کردی کاربر اجازه داره یا نه	حالا با <code>Authorize(...)</code> فقط یه خط کافیه

✓مثال کامل:

```
$note = $db->query("SELECT * FROM notes WHERE id =  
:id", [  
    "id" => $_GET["id"]  
)->findOrFail();  
  
Authorize($note["user_id"] == $current_user_id);
```

این دو خط تمام کاری رو که قبلاً توی ۷-۶ خط می‌نوشتی، انجام می‌ده:

- هم اطلاعات رو می‌گیره
- هم اگه پیدا نکرد می‌فرسته صفحه ۴۰۴
- هم چک می‌کنه طرف اجازه داره یا نه

اگه هنوز یه بخشیش برات گنگه، فقط بگو کدومش رو باز هم ساده‌تر کنم.
با مثال‌های واقعی و دقیق تا کاملاً برات روشن بشه.

You said:

یعنی اون دو تا if رو که در فایل notes.php گذاشته بودیم به جاش اومدیم دو تا تابع تو
کلاس database تعریف کردیم تا خودش این موارد رو چک کنه

ChatGPT said:

آفرین! دقیقاً فهمیدی چی شد👏
حالا بذار خیلی ساده و روشن‌تر برات خلاصه کنم:

✓سؤال:

اون دو تا if که در فایل notes.php نوشته بودیم چی بودن؟

۱. یکی برای اینکه اگه یادداشتی با اون id پیدا نشد 404 → نشون بدیم

```
if (! $note) {  
    abort(); // به صفحه ۴۰۴ redirect یا  
}
```

۲. یکی برای اینکه اگه کاربر اجازه نداره یادداشت رو ببینه 403 → بدیم

```
if ($note['user_id'] != $current_user_id) {  
    abort(403); // به صفحه اصلی redirect یا  
}
```

✳️ حالا چی کار کردیم؟

اومدیم این دو تا شرط تکراری رو بردیم توی دو تابع جداگانه که دیگه لازم نباشه بارها بنویسیمشون:

۱. تابع `findOrFail()` توی کلاس `Database`

این کار همون `if (!$note)` رو انجام می‌ده:

```
public function findOrFail() {  
    $result = $this->statement->fetch();  
    if (!$result) {  
        http_response_code(404);  
        header("Location:/view/404.html");  
        exit();  
    }  
    return $result;  
}
```

یعنی:

```
if (اگه یادداشتی پیدا نشد، خودم هندل می‌کنم، دیگه لازم نیست تو بنویسی  
    (!$note)).
```

2. تابع () Authorize از کلاس، ولی باز جدا شده)

این جایگزین اون (... != \$note['user_id']) شده:

```
function Authorize($condition, $status = 403) {  
    if (! $condition) {  
        http_response_code($status);  
        header("Location:/index.php");  
        exit();  
    }  
}
```

یعنی:

اگه شرط رد شد (مثلاً کاربر اجازه نداشت)، من خودم خطای ۴۰۳ می‌دم.

🔗 پس نتیجه:

```
// قبلاً  
$note = ...; // اجرا query  
if (! $note) { ...404... }  
if ($note['user_id'] != $current_user_id) { ...403... }  
  
// الان  
$note = $db->query(...)->findOrFail();  
Authorize($note['user_id'] == $current_user_id);
```

✓ دقیقاً همون کاره، فقط تمیزتر، کوتاه‌تر، قابل استفاده در همه جا.

اگه بخوای می‌تونم کد کامل قبل و بعدش رو هم برات کنار هم بذارم تا قشنگ تفاوتش رو ببینی. دوست داری؟

You said:

این وسط چرا `$statement` رو تبدیل به `$this->$statement` کردی اگه مثل قبل بود هم دوتا تبع اجرا میشد

ChatGPT said:

سؤال فوق العاده ای پرسیدی! 🙌
بذار خیلی ساده و با مثال برات توضیح بدم که چرا مجبور شدیم از `$this->statement` استفاده کنیم.

? مشکل چی بود؟

ما یه متد نوشتیم به نام `query()` که توش این خط بود:

```
$this->statement = $this->connection->prepare($query);
```

حالا اگه بخوای تو متدهای دیگه مثل `fetch()` یا `get()` دوباره به این `statement` دسترسی داشته باشی، باید یه جوری اون رو ذخیره کرده باشی.

✓ راه حل؟

باید یه متغیر سراسری (پراپرتی) توی کلاس بسازیم تا همی متدها بهش دسترسی داشته باشن.

```
public $statement;
```

و حالا در متد `query` اون رو مقدار می‌دیم:

```
$this->statement = $this->connection->prepare($query);
```

✗ اگه `$statement` رو نگه نداریم چی میشه؟

ببین این کد:

```
return $this->connection->prepare($query) -  
>execute($params);
```

کارش اینه که query رو اجرا می‌کنه ولی نتیجه‌ی اون رو نگه نمی‌داره. پس توی متد بعدی مثل `fetch()` دیگه دسترسی بهش نداریم.

✓ پس چرا `$this->statement`؟

چون فقط با `$this->statement` می‌تونیم مقدار آماده‌شده (`prepare`) رو در کل کلاس نگه داری و بعداً مثلاً توی `fetch()` یا `get()` ازش استفاده کنی:

```
public function fetch() {  
    return $this->statement->fetch(); // اینجا لازمه که  
    ذخیره شده باشه property قبلاً توی statement  
}
```

✈ خلاصه:

چرا؟	آیا میشه دوباره استفاده کرد؟	حالت
چون فقط داخل همون متد هست	✗	فقط یه متغیر داخل <code>query</code>
چون تو کل کلاس در دسترسه	✓	پراپرتی <code>\$this->statement</code>

📦 مثال ملموس:

فرض کن تو آشپزخونه یه بشقاب غذا درست کردی (`prepare` کردی)، حالا می‌خواهی در اتاق همون غذا رو بخوری (`fetch` کنی).

اگه بشقاب غذا رو تو همون آشپزخونه رها کنی، تو اتاق دیگه دستت بهش نمی‌رسه!
اما اگه غذا رو توی یه ظرف عمومی (پراپرتی) بریزی که همه جا همراهت باشه، هر جا بخوای
می‌تونی بخوریش 😊 !

□ سوال ۱:

چرا در کلاس Database، متغیر \$statement را به صورت public
\$statement تعریف کردیم و در متد query مقدار دادیم؟

□ سوال ۲:

در این کد چه چیزی بازمی‌گردد و چرا؟

```
return $this;
```

در انتهای متد. query.

□ سوال ۳:

چه فرقی بین متد fetch() و متد get() در کلاس Database وجود دارد؟
و هریک در چه موقعیتی استفاده می‌شوند؟

□ سوال ۴:

تابع findOrFail() دقیقاً چه کاری انجام می‌دهد و در چه مواقعی کاربرد دارد؟
(هم قسمت fetch را بگو، هم قسمت 404)

□ سوال ۵:

تابع `authorize($condition)` برای چه هدفی طراحی شده؟
و چرا بهتر است به جای نوشتن مستقیم شرطها از آن استفاده کنیم؟

✓ جواب ۱ (کمی تکمیلش می‌کنم):

درست گفتی. ما `public $statement` رو تعریف کردیم چون می‌خواستیم خروجی `prepare()` که یک شی `PDOStatement` هست رو ذخیره کنیم و بعداً در متدهای دیگه مثل `fetch()` یا `get()` ازش استفاده کنیم.
و چون این متغیر باید در سطح کلاس در دسترس باشه، `public` گذاشتیم.

✓ جواب ۲ (عالی گفتی):

بله، `return $this;` شیء جاری کلاس `Database` رو برمی‌گردونه. این کار رو کردیم تا بتونیم زنجیره‌ای بنویسیم، مثل:

```
$notes = $db->query(...)->get();
```

در اینجا `query()` خودش چیزی بر نمی‌گردونه جز شیء خودش (`$this`) و بعد `get()` روی همون شیء اجرا میشه.

✓ جواب ۳ (دقیق بود فقط با مثال کامل‌ترش کن):

درست گفتی.

- `fetch()` فقط یک رکورد از نتیجه کوئری برمی‌گردونه (مناسب برای `note.php`)
 - `get()` همه رکوردها رو برمی‌گردونه (مناسب برای `notes.php`)
- مثلاً:

```
$note = $db->query("SELECT * FROM notes WHERE id = :id", ['id' => 2])->fetch();  
$notes = $db->query("SELECT * FROM notes WHERE user_id = :id", ['id' => 3])->get();
```

✓جواب ۴ (خیلی خوب گفتی فقط دلیلش رو هم بگو):

آره، `findOrFail()`:

- اول `fetch()` رو اجرا می‌کنه
- اگر نتیجه‌ای نبود (یعنی یادداشت وجود نداشت)، کد ۴۰۴ می‌فرسته و صفحه ۴۰۴ رو نشون میده

چرا این کار خوبه؟ چون دیگه توی کنترلر یا فایل اصلی لازم نیست همیشه دستی `if` بنویسیم. راحت فقط می‌نویسیم:

```
$note = $db->query(...)->findOrFail();
```

و خیال‌مون راحت‌تره که یا رکورد هست یا صفحه ۴۰۴ میاد.

✓جواب ۵ (خیلی خوب گفتی فقط اینم اضافه کن):

تابع `authorize($condition)` بررسی می‌کنه آیا کاربر به چیزی (مثلاً یک یادداشت) دسترسی داره یا نه. اگر نداشت، کد ۴۰۳ و ریدایرکت.

مزیتش چیه؟

- به‌جای نوشتن چندباره 403 ... `if`، یک تابع (DRY یعنی بدون تکرار) ساختیم
 - کد تمیزتر و قابل‌فهم‌تر می‌شه
-

سؤال ۱:

در متد `findOrFail()`، چرا گفتیم:

```
if (!$note) {  
    abort();  
}
```

نه مثلاً:

```
if ($note === false) {  
    abort();  
}
```

سؤال ۲:

فرض کن در `note.php` این کد هست:

```
$note = $db->query('select * from notes where id =  
:id', ['id' => $_GET['id']])->findOrFail();
```

چه اتفاقی می‌افتد اگر `id` ای که در URL وارد کردیم، وجود نداشته باشد؟

سؤال ۳:

در این کد:

```
authorize($note['user_id'] === 1);
```

اگر کاربر لاگین نکرده باشد و `$_SESSION['user']` ست نشده باشد، چه مشکلی
ممکنه پیش بیاد؟ راه حلش چیه؟

سؤال ۴:

چرا `abort()` رو نوشتیم به جای اینکه مستقیماً بنویسیم:

```
http_response_code(404);  
require 'views/404.php';  
die();
```

مزیتش چیه؟

? سؤال ۵:

اگر کسی به صورت مستقیم بخواد آدرس `/note?id=3` رو باز کنه ولی اون یادداشت برای کاربر دیگه‌ای باشه، با وجود `findOrFail()` باز هم امنیت حفظ میشه؟ چرا یا چرا نه؟

✓ جواب ۱:

سؤال:

در متد `findOrFail()` چرا نوشتیم `if (!$note)` نه مثلاً `if ($note === false)`؟

پاسخ ساده:

چون متد `fetch()` اگه هیچ نتیجه‌ای پیدا نکنه، مقدار `false` برمی‌گردونه. ولی ما فقط می‌خوایم بررسی کنیم که «آیا نتیجه‌ای وجود داره یا نه.»
پس `if (!$note)` یعنی "اگه هیچی پیدا نشد"، که هم `false` و هم `null` هر چیز تهی دیگه‌ای رو چک می‌کنه.
اما `if ($note === false)` فقط `false` رو می‌سنجه، ممکنه بعضی حالت‌های دیگه (مثل `null`) رو نادیده بگیره.

✓ جواب ۲:

سؤال:

اگر id وجود نداشته باشد چه می‌شود؟

پاسخ کامل:

دقیقاً همون طور که گفتی، متد `findOrFail()` می‌فهمه که هیچ یادداشتی با این id پیدا نشده، پس تابع `abort()` اجرا میشه و یه صفحه ۴۰۴ نشون داده میشه.
✓ پس درست گفتی: خطای ۴۰۴.

✓ جواب ۳:

سؤال:

اگر کاربر لاگین نکرده باشد و بخوایم `authorize($note['user_id'] === $_SESSION['user']['id'])` اجرا کنیم چه می‌شود؟

پاسخ:

اگه `$_SESSION['user']` ست نشده باشد، وقتی بخوای `$_SESSION['user']['id']` رو بخونی، خطای `undefined index` یا حتی `Trying to access array offset on value of type null` می‌گیری.

✗ پس نتیجه اینه که برنامه می‌پره و اصلاً به تابع `authorize()` نمی‌رسه!

✓ راه حل:

قبل از استفاده از `$_SESSION['user']` باید مطمئن باشیم که کاربر لاگین کرده. مثلاً با یه شرط مثل:

```
if (!isset($_SESSION['user'])) {  
    // کاربر لاگین نکرده، بفرستش به صفحه login یا ارور بده  
}
```

✓ جواب ۴:

سؤال:

چرا `abort()` بهتر از نوشتن مستقیم `http_response_code(404)` و `require '404.php'` هست؟

پاسخ کامل:

آفرین گفتی «برای جلوگیری از» DRY، یعنی **Don't Repeat Yourself** ✓

تابع `abort()` این کدها رو یکجا جمع کرده.

هر بار که بخوایم یه ارور ۴۰۴ بدیم، فقط کافیه بگیم `abort()`، نه اینکه دستی دوباره همه اون کدها رو بنویسیم. این باعث نظم، خوانایی و قابلیت نگهداری بهتر کد میشه.

✓ جواب ۵:

سؤال:

اگر کاربر بخواد یادداشت دیگران رو با URL دستی ببینه، آیا `findOrFail()` جلوی این کارو میگیره؟

پاسخ:

نه! فقط `findOrFail()` جلوی ۴۰۴ رو میگیره، نه ۴۰۳.

اگر id موجود باشه ولی مربوط به کاربر دیگه‌ای باشه، `findOrFail()` اون یادداشت رو پیدا می‌کنه و نشون میده.

برای جلوگیری از این مشکل، باید `authorize()` بنویسیم که بررسی کنه:

```
authorize($note['user_id'] === $currentUserId);
```

این شرط امنیتی باعث میشه فقط کسی که یادداشت مال اونه بتونه ببینه، بقیه ۴۰۳ بگیرن.

✓

حتماً! بیا خیلی ساده، قدم به قدم و با مثال توضیح بدم که چرا در متد `findOrFail()` نوشتیم:

```
if (! $note) {
    abort();
}
```

و نکتیم:

```
if ($note === false) {
    abort();
}
```

✓ **اول: وقتی `fetch()` صدا زده میشه چی برمی‌گردونه؟**

تابع `fetch()` معمولاً یکی از این سه حالت رو برمی‌گردونه:

۱. $\frac{1}{2}$ اگر نتیجه پیدا بشه \rightarrow یه آرایه (مثلاً یه یادداشت)
۲. \times اگر چیزی پیدا نشه `false` \rightarrow
۳. (در بعضی شرایط خیلی خاص) \rightarrow ممکنه `null` یا یه چیز دیگه برگردونه (مثلاً اگه کوئری درست اجرا نشده باشه)

✓ **حالا تفاوت `$note != false` با `$note === false` چیه؟**

👉 بررسی کنیم:

`$note === false` یعنی چی؟ `$note != false` یعنی چی؟ مقدار `$note`

آرایه یادداشت	false	False
false	true ✓	true ✓
null	true ✓	✗ false
"" (رشته خالی)	true ✓	✗ false
0	true ✓	✗ false

✓ **نتیجه‌گیری ساده:**

اگر فقط بنویسی:

```
if ($note === false)
```

یعنی فقط چک می‌کنی که `$note` دقیقاً `false` باشد.
ولی اگر یه حالت خاص دیگه اتفاق بیفته، مثل `null`، اون وقت برنامه متوجه نمی‌شه که باید `abort()` کنه!

✓ اما اگر بنویسی:

```
if (! $note)
```

یعنی:

اگر `$note` خالی بود، `false` بود، `null` بود، یا هرچی که معنیش «هیچی نیست» بود → `abort()` کن.

این روش امن‌تره، مخصوصاً برای دیتابیس و `fetch` کردن، چون شرایط مختلفی ممکنه پیش بیاد.

✓ یه مثال برای درک بهتر:

فرض کن بری تو بانک و بگی:

"سلام، یادداشت با id شماره ۵ رو بده!"

اگه بانک بگه:

- «پیدا نکردم» → یعنی `false`
- «یادداشت اصلاً وجود نداره» → ممکنه `null` باشه
- «یادداشت خالیه» → مثلاً یه آرایه‌ی خالی

تو فقط می‌خواهی بدونی:

"آیا یه یادداشت واقعی بهم دادی یا نه؟"

پس اگه گفتی `if (! $note)` یعنی:

"اگه هیچی ندادی، من صفحه ۴۰۴ رو نشون میدم".

۱. تابع `findOrFail()` کجا تعریف میشه؟

این تابع توی کلاس `Database` تعریف میشه.

چون داریم سعی می‌کنیم یک یادداشت (`Note`) خاص رو از پایگاه داده بیاریم، پس منطقیه که متدی مثل `findOrFail()` در کلاس `Database` باشه تا این کار رو انجام بده.

```
// کلاس Database.php
class Database
{
    // سایر توابع مثل query() و get() و fetch()

    public function findOrFail($query, $params)
    {
        $statement = $this->query($query, $params);

        $result = $statement->fetch();

        if (! $result) {
            abort(); // نشون دادن صفحه ۴۰۴
        }

        return $result;
    }
}
```

این متد:

- `SELECT * FROM notes WHERE id = :id` (مثلاً) می‌گیره
- اونو اجرا می‌کنه
- نتیجه رو با `fetch()` می‌گیره
- اگه خالی بود \rightarrow `($result != true)` تابع `abort()` رو اجرا می‌کنه
- اگه داده پیدا شد \rightarrow همون داده رو برمی‌گردونه

۲. تابع `authorize()` کجا تعریف میشه؟ ✓

تابع `authorize()` یک تابع کمکی (Helper Function) هست، نه در کلاس `Database`. معمولاً این تابع در یکی از فایل‌های عمومی پروژه تعریف میشه، مثل:

```
// helpers.php یا functions.php
function authorize($condition, $status =
Response::FORBIDDEN)
{
    if (! $condition) {
        abort($status); // مثلاً کد ۴۰۳
    }
}
```

این تابع برای این ساخته شده که بررسی کنه آیا کاربر مجاز هست یا نه.

مثال استفاده:

```
authorize($note['user_id'] === 1); // اگه کاربر لاگین کرده صاحب
یادداشت نیست  $\rightarrow$  ۴۰۳
```

✓ جمع‌بندی سریع:

تابع	در کجا تعریف میشه	وظیفه
<code>findOrFail()</code>	در کلاس <code>Database</code>	گرفتن رکورد خاص از دیتابیس یا اجرای ۴۰۴ اگر نبود

وظیفه	در کجا تعریف میشه	تابع
بررسی مجوزها (مثلاً مالکیت یادداشت) و اجرای ۴۰۳ در صورت نداشتن مجوز	در فایل کمکی مثل helpers.php	authorize()
