**Real Estate feladat**

Adott egy egyszerű 1 táblás adatbázis néhány adattal:

CREATE DATABASE real\_estate CHARACTER SET utf8mb4 COLLATE utf8mb4\_unicode\_ci;

USE real\_estate;

CREATE TABLE apartments (

    id INT AUTO\_INCREMENT PRIMARY KEY,

    address VARCHAR(255) NOT NULL,

    city VARCHAR(100) NOT NULL,

    postal\_code VARCHAR(20) NOT NULL,

    size INT NOT NULL COMMENT 'Méret négyzetméterben',

    rooms INT NOT NULL COMMENT 'Szobák száma',

    price DECIMAL(10,2) NOT NULL COMMENT 'Ár forintban',

    owner\_name VARCHAR(100) NOT NULL,

    owner\_phone VARCHAR(20) NOT NULL,

    listed\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP COMMENT 'Hirdetés feladási dátuma',

    description TEXT COMMENT 'Lakás rövid leírása',

    image\_url VARCHAR(255) DEFAULT NULL COMMENT 'Kép URL a lakásról'

);

INSERT INTO apartments (address, city, postal\_code, size, rooms, price, owner\_name, owner\_phone, description, image\_url)

VALUES

('Kossuth Lajos utca 10.', 'Budapest', '1053', 85, 3, 75000000, 'Kovács János', '+36201234567', 'Kiváló állapotú lakás a belvárosban', 'https://example.com/lakas1.jpg'),

('Petőfi Sándor utca 15.', 'Debrecen', '4025', 65, 2, 45000000, 'Nagy Mária', '+36203334444', 'Felújított panel lakás parkra néző kilátással', 'https://example.com/lakas2.jpg'),

('Táncsics Mihály tér 8.', 'Szeged', '6720', 120, 4, 98000000, 'Szabó Péter', '+36204445555', 'Új építésű családi ház csendes környezetben', 'https://example.com/lakas3.jpg');

A feladat egy realestate.php fájl létrehozása, a REST API követelményeinek megfelelően.

Megoldás lépésről lépésre:

Adatbázis kapcsolat létrehozása a legegyszerűbb módon:

$servername = "localhost";

$username = "root";

$password = "";

$database = "real\_estates";

$dbconn = new mysqli($servername, $username, $password, $database);

if ($dbconn->connect\_error) {

die("Connection failed: " . $conn->connect\_error);

}

$dbconn->set\_charset("utf8mb4");

|  |  |  |
| --- | --- | --- |
| **Sor** | **Mit csinál?** | **Megjegyzés** |
| $servername = "localhost"; | MySQL szerver címe | Lokálisan vagy távoli szerveren lehet. |
| $username = "root"; | MySQL felhasználónév | Root az alapértelmezett helyi szervereken. |
| $password = ""; | MySQL jelszó | XAMPP-ben alapértelmezetten üres. |
| $database = "real\_estates"; | Használt adatbázis neve | A megfelelő adatbázist kell beírni. |
| $dbconn = new mysqli(...) | Kapcsolat létrehozása | A MySQLi objektumot inicializálja. |
| if ($dbconn->connect\_error) { ... } | Hibakezelés | Ha a kapcsolat nem jön létre, leállítja a programot. |
| $dbconn->set\_charset("utf8mb4"); | Karakterkódolás beállítása | Fontos az ékezetes betűk miatt. |

A következő négy **header()** függvény meghatározza, hogy az API hogyan kommunikál a böngészőkkel és más kliensekkel. Ezek a fejlécek különösen fontosak akkor, ha másik doménről (cross-origin) érkező kéréseket is engedélyezni szeretnénk.

**1. header("Access-Control-Allow-Origin: \*");**

**Mit csinál?**  
Ez a fejlécküldés **engedi, hogy bármelyik doménről** (\*) lehessen kéréseket küldeni az API-hoz.

**Miért kell?**

* Ha a frontend egy másik doménen van (pl. http://frontend.com), és az API egy másik szerveren fut (http://backend.com), akkor az alapértelmezett **böngészőbiztonsági szabályok (CORS - Cross-Origin Resource Sharing)** miatt a kérések blokkolva lennének.
* Ez a fejléc **megoldja ezt a problémát**, és engedélyezi az API elérését bármilyen doménről.

**Alternatíva:**  
Ha csak bizonyos doméneket szeretnél engedélyezni, akkor helyette ezt használhatod:

header("Access-Control-Allow-Origin: https://frontend.com");

**2. header("Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS");**

**Mit csinál?**  
Ez határozza meg, hogy az API milyen **HTTP metódusokat** engedélyez.

**Miért kell?**

* Ha egy **böngészőben futó JavaScript alkalmazás** küld egy POST vagy PUT kérést az API-ra, a szervernek meg kell adnia, hogy ezt a műveletet engedélyezi.
* Az **OPTIONS** metódus azért fontos, mert a böngészők **előzetes "preflight" kérést küldenek** (OPTIONS), hogy ellenőrizzék, milyen műveletek engedélyezettek.

**Alternatíva:**  
Ha csak GET és POST kéréseket szeretnél engedélyezni:

header("Access-Control-Allow-Methods: GET, POST");

**3. header("Access-Control-Allow-Headers: Content-Type, Authorization");**

**Mit csinál?**  
Ez meghatározza, hogy a kliens milyen **egyedi HTTP-fejléceket** küldhet az API-nak.

**Miért kell?**

* Ha a frontend Content-Type: application/json fejlécet küld egy POST vagy PUT kérésben, akkor **alapértelmezetten a szerver ezt elutasítaná**, kivéve ha kifejezetten engedélyezzük.
* Az **Authorization** fejléc engedélyezése fontos, ha **hitelesítési tokeneket** (Bearer Token, JWT) küldünk az API számára.

**Alternatíva:**  
Ha több egyedi fejlécet szeretnél engedélyezni:

header("Access-Control-Allow-Headers: Content-Type, Authorization, X-Requested-With");

(Az X-Requested-With például AJAX kérések esetén hasznos lehet.)

**4. header("Content-Type: application/json; charset=UTF-8");**

**Mit csinál?**  
Ez azt mondja meg a böngészőnek vagy más kliensnek, hogy az API **válasza JSON formátumú**, és az adatok **UTF-8 kódolásúak**.

**Miért kell?**

* A modern REST API-k **JSON adatokat használnak**, ezért ezt a szervernek egyértelműen jeleznie kell a kliensek felé.
* Az **UTF-8 karakterkódolás** biztosítja, hogy az ékezetes karakterek (á, é, ő, ü) és egyéb speciális karakterek helyesen jelenjenek meg.

**Alternatíva:**  
Ha például egy XML formátumú API-t készítenél:

header("Content-Type: application/xml; charset=UTF-8");

|  |  |
| --- | --- |
| **Fejléc** | **Miért fontos?** |
| Access-Control-Allow-Origin: \* | Engedélyezi a CORS-t, így más domének is elérhetik az API-t. |
| Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS | Meghatározza, milyen HTTP metódusok engedélyezettek. |
| Access-Control-Allow-Headers: Content-Type, Authorization | Engedélyezi az egyedi fejléceket (pl. Content-Type: application/json, Authorization). |
| Content-Type: application/json; charset=UTF-8 | Beállítja, hogy az API válasza JSON és UTF-8 karakterkódolású legyen. |

**Ha ezeket nem állítanánk be, akkor:**

* A böngésző **blokkolhatná a kéréseket** a CORS miatt.
* Nem tudnánk POST, PUT és DELETE műveleteket végrehajtani.
* JSON helyett a szerver lehet, hogy **plain textként** küldené a választ.

Ezért ezek a fejlécek **elengedhetetlenek** egy jól működő REST API esetén!

$requestMethod = $\_SERVER['REQUEST\_METHOD'];

* A **PHP $\_SERVER szuperglobális változójának** REQUEST\_METHOD eleme a **beérkező HTTP kérés metódusát** tárolja (GET, POST, PUT, DELETE, stb.).
* Ez az érték attól függ, hogy a kliens (pl. böngésző, Postman, JavaScript fetch() API) milyen módszert használ a kérés küldésére.

$id = isset($\_GET['id']) ? intval($\_GET['id']) : null;

* Ellenőrzi, hogy a kliens megadott-e egy id paramétert a kérésben (GET metódus esetén az URL-ben, PUT vagy DELETE metódusnál is szerepelhet az URL-ben).
* Ha igen, akkor integer típusúvá alakítja (intval()) az értéket.
* Ha nincs id paraméter, akkor null értéket ad.
* **Példa bemenetek és kimenetek:**

|  |  |  |
| --- | --- | --- |
| **URL** | **$\_GET['id']** | **$id értéke** |
| /apartments.php?id=5 | "5" (string) | 5 (int) |
| /apartments.php?id=abc | "abc" | 0 (mert intval("abc") → 0) |
| /apartments.php | nincs | null |

 Az intval($\_GET['id']) segítségével **biztosítjuk, hogy az id mindig szám legyen**, és így elkerüljük az **SQL injection támadásokat** vagy hibás lekérdezéseket.

 Ha egy kérést például **DELETE metódussal** küldenek, **ellenőriznünk kell, hogy van-e id**:

**1. case 'GET': – Adatok lekérése**

case 'GET':

if ($id) {

getApartmentById($id, $dbconn);

} else {

getAllApartments($dbconn);

}

break;

* **Mit csinál?**
  + Ha van id az URL-ben (GET /apartments.php?id=5), akkor meghívja a getApartmentById($id, $dbconn); függvényt, amely egy adott lakás adatait kéri le.
  + Ha **nincs id paraméter**, akkor a getAllApartments($dbconn); függvény fut le, amely az **összes lakást** lekéri.
* **Miért fontos?**
  + Ez biztosítja, hogy a GET metódus **egyedi lakást** vagy **az összes lakást** lekérje a kliens kérésének megfelelően.

**2. case 'POST': – Új lakás létrehozása**

case 'POST':

createApartment($dbconn);

break;

* **Mit csinál?**
  + Meghívja a createApartment($dbconn); függvényt, amely új lakás adatokat vesz fel az adatbázisba.
* **Miért fontos?**
  + A POST metódus **új rekordot** hoz létre az adatbázisban.
  + Általában a kliens egy JSON objektumot küld a kérés törzsében az új lakás adataival.

**3. case 'PUT': – Lakás adatainak frissítése**

case 'PUT':

if ($id) {

updateApartment($id, $dbconn);

} else {

http\_response\_code(400);

echo json\_encode(['status' => 'error', 'message' => 'Hiányzik az azonosító.']);

}

break;

* **Mit csinál?**
  + Ha van id paraméter (PUT /apartments.php?id=5), akkor meghívja az updateApartment($id, $dbconn); függvényt.
  + Ha **nincs id**, akkor egy 400 Bad Request hibakódot küld vissza és egy hibaüzenetet ad vissza JSON formátumban.
* **Miért fontos?**
  + A PUT metódus **egy létező lakás adatainak frissítésére** szolgál, ezért szükség van egy id-ra, hogy tudja, melyik lakást kell módosítani.

**4. case 'DELETE': – Lakás törlése**

case 'DELETE':

if ($id) {

deleteApartment($id, $dbconn);

} else {

http\_response\_code(400);

echo json\_encode(['status' => 'error', 'message' => 'Hiányzik az azonosító.']);

}

break;

* **Mit csinál?**
  + Ha van id az URL-ben (DELETE /apartments.php?id=5), akkor meghívja a deleteApartment($id, $dbconn); függvényt, amely törli az adott lakást az adatbázisból.
  + Ha **nincs id**, akkor 400 Bad Request hibát ad vissza JSON üzenettel.
* **Miért fontos?**
  + A DELETE metódus **egy adott rekord törlésére** szolgál, ezért szükséges az id.

**5. default: – Nem támogatott HTTP metódus kezelése**

default:

http\_response\_code(405);

echo json\_encode(['status' => 'error', 'message' => 'Nem támogatott HTTP metódus.']);

break;

* **Mit csinál?**
  + Ha a kliens egy **nem támogatott HTTP metódust** (PATCH, HEAD, stb.) küld, akkor 405 Method Not Allowed hibát küld vissza és egy hibaüzenetet ad JSON formátumban.
* **Miért fontos?**
  + Ez megakadályozza, hogy az API **hibás vagy nem engedélyezett metódusokat fogadjon el**.

|  |  |
| --- | --- |
| **HTTP metódus** | **Mit csinál?** |
| GET | Egy adott lakás vagy az összes lakás lekérése. |
| POST | Új lakás létrehozása. |
| PUT | Egy meglévő lakás adatainak frissítése (id szükséges). |
| DELETE | Egy adott lakás törlése (id szükséges). |
| default | Ha nem támogatott metódust küld a kliens, 405 hibát ad vissza. |

Ez a szerkezet biztosítja, hogy az API **megfelelően kezelje a különböző HTTP kéréseket**, és **hibabiztosan működjön**.

**getAllApartments($dbconn)**

Ez a függvény az **apartments** tábla összes rekordját lekéri az adatbázisból, majd JSON formátumban visszaadja. Vizsgáljuk meg soronként!

**1. Függvény deklarálása**

function getAllApartments($dbconn) {

* **Mit csinál?**
  + Létrehoz egy függvényt getAllApartments néven.
  + **Bemeneti paramétere** $dbconn, ami a **MySQL adatbázis kapcsolatot** jelenti.

**2. SQL lekérdezés létrehozása**

$query = "SELECT \* FROM apartments";

* **Mit csinál?**
  + Az SQL lekérdezést egy változóba ($query) menti.
  + A SELECT \* FROM apartments utasítás azt jelenti, hogy **az összes oszlopot és rekordot** ki akarjuk olvasni az apartments táblából.

**3. Lekérdezés futtatása**

$result = mysqli\_query($dbconn, $query);

* **Mit csinál?**
  + A mysqli\_query($dbconn, $query) függvény futtatja a korábban definiált SQL lekérdezést.
  + Ha sikeres a lekérdezés, a mysqli\_query() visszaad egy **eredményobjektumot** ($result), amely tartalmazza a lekérdezett adatokat.
  + Ha **hibás** az SQL utasítás vagy nincs kapcsolat az adatbázissal, $result értéke false lesz.

**4. Ellenőrzi, hogy sikeres volt-e a lekérdezés**

if ($result) {

* **Mit csinál?**
  + Ellenőrzi, hogy a $result változó nem false, azaz a lekérdezés sikeres volt-e.

**5. Adatok tömbbé alakítása**

$data = mysqli\_fetch\_all($result, MYSQLI\_ASSOC);

* **Mit csinál?**
  + A mysqli\_fetch\_all($result, MYSQLI\_ASSOC) függvény az összes lekérdezett rekordot egy **asszociatív tömbként** (MYSQLI\_ASSOC) formázza meg.
  + **Példa visszatérési értékre:**

[

["id" => 1, "address" => "Fő utca 10.", "city" => "Budapest"],

["id" => 2, "address" => "Kossuth tér 5.", "city" => "Szeged"]

]

* + Ez **JSON-kompatibilis**, így könnyen továbbítható a kliens felé.

**6. HTTP státuszkód beállítása a sikeres válaszhoz**

http\_response\_code(200);

* **Mit csinál?**
  + Beállítja a **HTTP válaszkódot 200-ra**, ami azt jelzi a kliens felé, hogy a kérés **sikeresen** teljesült.
  + A 200 OK státuszkód az egyik leggyakoribb HTTP válaszkód az API-k esetében.

**7. JSON válasz küldése a kliensnek**

echo json\_encode(['status' => 'success', 'data' => $data]);

* **Mit csinál?**
  + JSON formátumban küldi vissza az adatokat a kliens felé.
  + **Visszaküldött formátum (példa)**:

{

"status": "success",

"data": [

{"id": 1, "address": "Fő utca 10.", "city": "Budapest"},

{"id": 2, "address": "Kossuth tér 5.", "city": "Szeged"}

]

}

**8. Ha a lekérdezés nem sikerült, hibakódot ad vissza**

} else {

http\_response\_code(500);

* **Mit csinál?**
  + Ha a $result értéke false, azaz a lekérdezés **sikertelen**, akkor **500-as HTTP válaszkódot** (Internal Server Error) küld vissza.
  + Az 500 státuszkód azt jelzi, hogy **a szerver oldalon valamilyen hiba történt**.

**9. Hibaüzenet küldése a kliensnek**

echo json\_encode(['status' => 'error', 'message' => 'Adatbázis hiba: ' . mysqli\_error($dbconn)]);

* **Mit csinál?**
  + JSON formátumban küldi vissza a hibaüzenetet.
  + **Példa visszakapott hibaüzenetre:**

{

"status": "error",

"message": "Adatbázis hiba: Unknown column 'city' in 'field list'"

}

* + A mysqli\_error($dbconn) függvény segítségével **megmutatja a MySQL hibát**, ami segíthet a hibakeresésben.

|  |  |  |  |
| --- | --- | --- | --- |
| **Sor** | **Kód** | **Mit csinál?** | **Miért fontos?** |
| 1. | function getAllApartments($dbconn) { | Létrehozza a függvényt. | Az API így képes hívni ezt a funkciót. |
| 2. | $query = "SELECT \* FROM apartments"; | SQL lekérdezés létrehozása. | Meghatározza, hogy milyen adatokat kérünk le. |
| 3. | $result = mysqli\_query($dbconn, $query); | SQL futtatása. | Ha hiba történik, $result false lesz. |
| 4. | if ($result) { | Ellenőrzi a lekérdezés sikerességét. | Megakadályozza a hibás adatok visszaadását. |
| 5. | $data = mysqli\_fetch\_all($result, MYSQLI\_ASSOC); | Az eredményt asszociatív tömbbé alakítja. | Könnyen feldolgozhatóvá teszi az adatokat. |
| 6. | http\_response\_code(200); | Beállítja a sikeres válaszkódot. | A kliens tudja, hogy a lekérdezés sikeres volt. |
| 7. | echo json\_encode([...]); | JSON válasz visszaküldése. | Az API szabványos választ küld. |
| 8. | } else { | Hibaellenőrzés. | Ha hiba van, ne adjunk vissza üres JSON-t. |
| 9. | http\_response\_code(500); | Beállítja az 500-as hibakódot. | A kliens tudja, hogy hiba történt. |
| 10. | echo json\_encode([...]); | JSON formátumú hibaüzenetet küld. | Segít a hibakeresésben. |

**Lehetséges továbbfejlesztések**

* **Lapozás (LIMIT, OFFSET) bevezetése**, hogy ne terhelje túl a szervert sok rekord esetén:

$query = "SELECT \* FROM apartments LIMIT 50 OFFSET 0";

* **Hibakezelés pontosítása**, pl. ha nincs találat, 404 Not Found küldése:

if (empty($data)) {

http\_response\_code(404);

echo json\_encode(['status' => 'error', 'message' => 'Nem található adat.']);

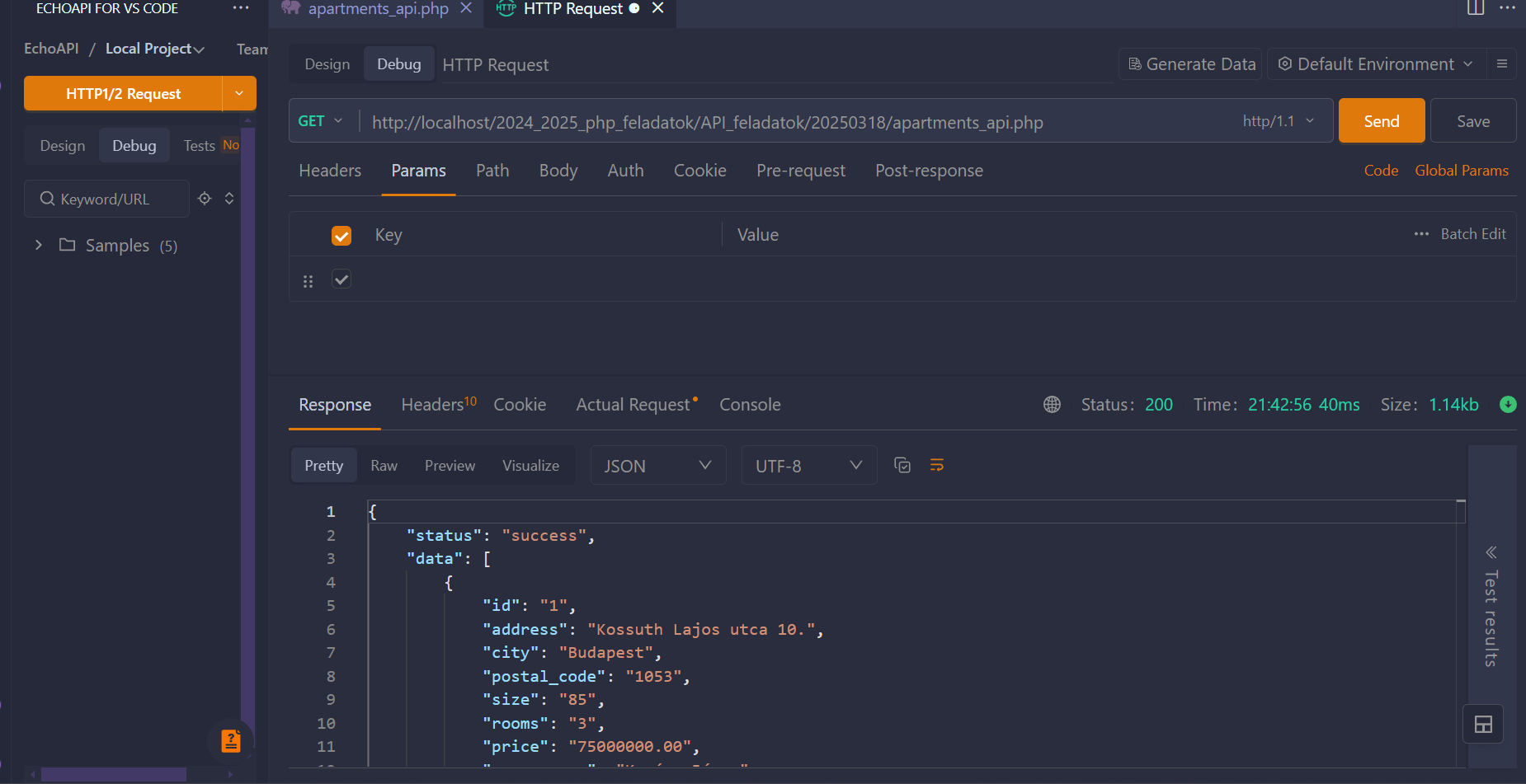
return;

}

* **JSON\_PRETTY\_PRINT hozzáadása** fejlesztői környezetben a könnyebb olvashatóság érdekében:

echo json\_encode(['status' => 'success', 'data' => $data], JSON\_PRETTY\_PRINT);

Ez a függvény **biztonságosan lekérdezi az adatokat**, **JSON formátumban visszaküldi**, és megfelelően kezeli az **adatbázis hibákat**. A helyes HTTP státuszkódokat is beállítja, így jól működik **API-ként** frontend vagy más rendszerek számára.



**getApartmentById($id, $dbconn)**

Ez a függvény egy adott **lakás** adatait kéri le az apartments táblából a **megadott id alapján**. A **lekérdezés biztonságos módon történik** előkészített (prepared) SQL utasítással, és az eredményt **JSON formátumban adja vissza**.

**1. Függvény deklarálása**

function getApartmentById($id, $dbconn) {

* **Mit csinál?**
  + Létrehoz egy függvényt getApartmentById néven.
  + **Bemeneti paraméterek:**
    - $id: A keresett lakás azonosítója.
    - $dbconn: A MySQL adatbázis kapcsolat.

**2. SQL lekérdezés előkészítése**

$query = "SELECT \* FROM apartments WHERE id = ?";

* **Mit csinál?**
  + Létrehoz egy **paraméterezett SQL lekérdezést**, amely az id alapján keresi a lakást.
  + A ? helyettesítő (placeholder) jelzi, hogy az id értékét **biztonságosan, később** fogjuk behelyettesíteni.
  + **Miért fontos?**
    - **SQL injection elleni védelem**: így elkerülhető, hogy a felhasználó **rosszindulatú SQL parancsokat** adjon meg az id helyett.
    - **Hatékonyabb lekérdezés**: az SQL szerver optimalizálni tudja az ilyen lekérdezéseket.

**3. Lekérdezés előkészítése**

$stmt = $dbconn->prepare($query);

* **Mit csinál?**
  + A prepare() függvény **előállít egy SQL utasítást**, amely később végrehajtható **biztonságosan**.
  + **Miért fontos?**
    - Megakadályozza az **SQL injection támadásokat**.
    - Gyorsítja az adatbázis-lekérdezéseket (SQL szerver optimalizáció miatt).

**4. Paraméter értékének beállítása**

$stmt->bind\_param('i', $id);

* **Mit csinál?**
  + A bind\_param('i', $id) függvény **hozzárendeli az id értéket** a korábban létrehozott ? helyettesítőhöz.
  + 'i' azt jelenti, hogy az id **egész szám (integer)** típusú érték.
  + **Miért fontos?**
    - Ha az id **nem szám**, az SQL szerver automatikusan elutasítja, így megelőzhetők a hibák és biztonsági rések.

**5. SQL parancs végrehajtása**

$stmt->execute();

* **Mit csinál?**
  + Végrehajtja az előkészített SQL lekérdezést a **megadott id-val**.
  + **Miért fontos?**
    - A bind\_param() segítségével a **paraméterezett lekérdezések garantáltan biztonságosak**.

**6. Eredmények lekérdezése**

$result = $stmt->get\_result();

* **Mit csinál?**
  + Az SQL szerver **eredményhalmazát (result set)** visszaadja.
  + Az adatok itt még **nem olvashatók**, csak egy objektum tartalmazza őket.

**7. Ellenőrzi, hogy van-e találat**

if ($result->num\_rows > 0) {

* **Mit csinál?**
  + Megnézi, hogy a **lekérdezés visszaadott-e legalább egy eredményt**.
  + Ha num\_rows > 0, az azt jelenti, hogy az id-hoz **létezik egy lakás**.

**8. Ha van találat, JSON formátumban visszaadja az adatokat**

http\_response\_code(200);

echo json\_encode(['status' => 'success', 'data' => $result->fetch\_assoc()]);

* **Mit csinál?**
  + Beállítja a **200 OK** válaszkódot (http\_response\_code(200)), ami azt jelzi, hogy a kérés sikeres volt.
  + A fetch\_assoc() metódus az **első találatot** egy **asszociatív tömbbé alakítja**, pl.:

[

"id" => 5,

"address" => "Kossuth Lajos utca 10.",

"city" => "Budapest",

"price" => 75000000

]

* + **JSON válaszként visszaküldi a klienseknek**:

{

"status": "success",

"data": {

"id": 5,

"address": "Kossuth Lajos utca 10.",

"city": "Budapest",

"price": 75000000

}

}

**9. Ha nincs találat, hibaüzenetet küld**

} else {

http\_response\_code(404);

echo json\_encode(['status' => 'error', 'message' => 'Lakás nem található.']);

* **Mit csinál?**
  + Ha **nincs találat** (num\_rows == 0), beállítja a **404 Not Found** státuszkódot.
  + JSON válaszként hibaüzenetet küld a kliensnek:

{

"status": "error",

"message": "Lakás nem található."

}

* **Miért fontos?**
  + A kliens (frontend, mobilalkalmazás) így tudja, hogy a megadott azonosító **nem található az adatbázisban**.

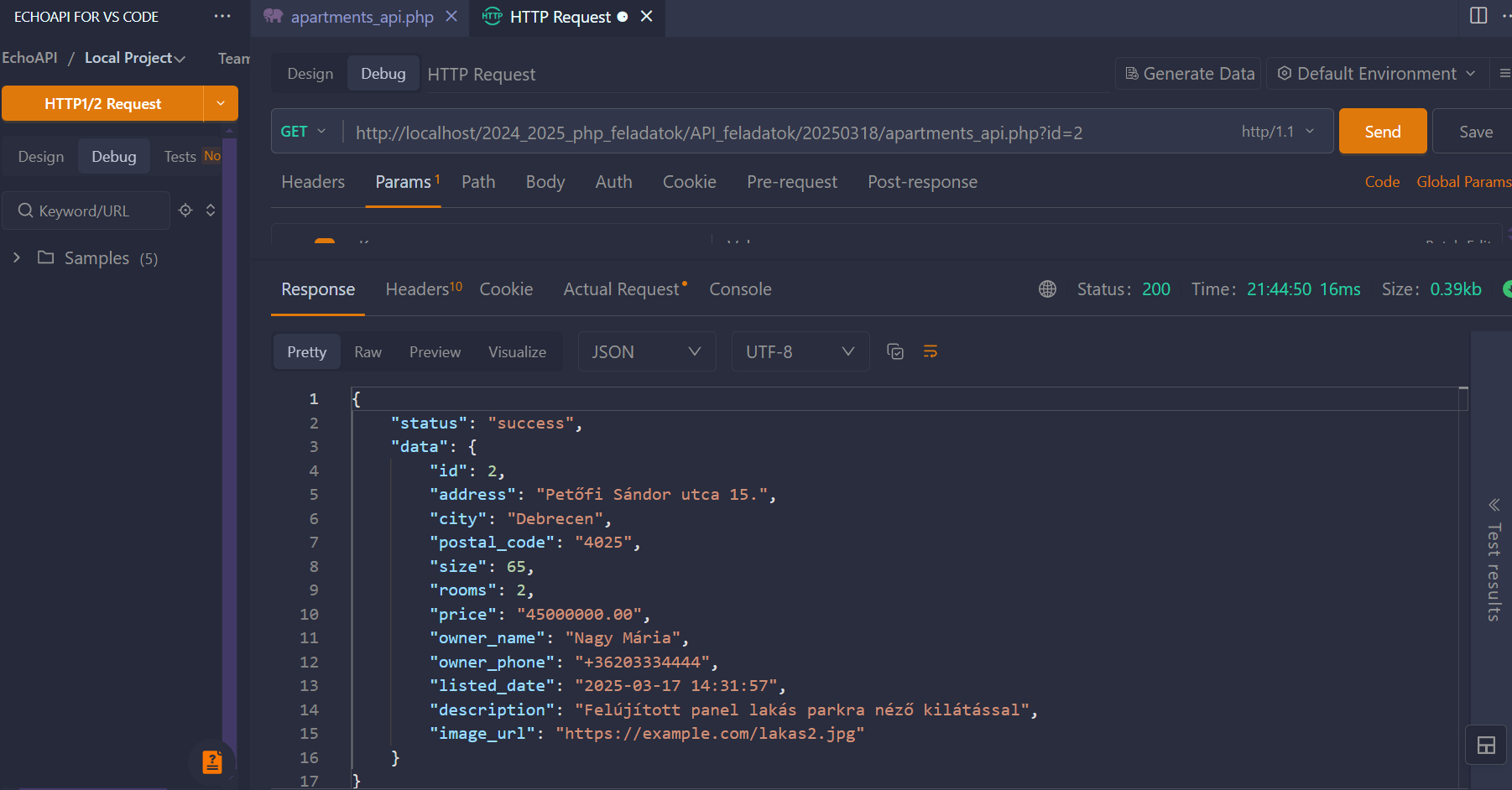
|  |  |  |  |
| --- | --- | --- | --- |
| **Sor** | **Kód** | **Mit csinál?** | **Miért fontos?** |
| 1. | function getApartmentById($id, $dbconn) { | Függvény létrehozása. | Az API ezen keresztül hívható. |
| 2. | $query = "SELECT \* FROM apartments WHERE id = ?"; | SQL lekérdezés paraméterrel. | Biztonságos SQL kezelés. |
| 3. | $stmt = $dbconn->prepare($query); | Lekérdezés előkészítése. | SQL injection elleni védelem. |
| 4. | $stmt->bind\_param('i', $id); | Az id értékét beilleszti a lekérdezésbe. | Típusbiztosítás, védelem. |
| 5. | $stmt->execute(); | Lekérdezés végrehajtása. | Az adatbázis megkapja az utasítást. |
| 6. | $result = $stmt->get\_result(); | Eredmények lekérése. | Ellenőrizni tudjuk a találatokat. |
| 7. | if ($result->num\_rows > 0) { | Megnézi, hogy van-e találat. | Csak akkor ad vissza adatot, ha létezik. |
| 8. | echo json\_encode([...]); | JSON formátumban visszaadja az adatokat. | Az API kliens könnyen feldolgozhatja. |
| 9. | http\_response\_code(404); | Ha nincs találat, 404-es hibát küld. | Hibakezelés a kliensek számára. |

**Lehetséges továbbfejlesztések**

1. **Lapozás (LIMIT használata)**: Ha több találat lehet (bár itt id alapján van keresés, így mindig maximum 1 rekordot kapunk).
2. **Hibaellenőrzés ($stmt->execute() visszatérési értéke)**: Ha a prepare() vagy execute() sikertelen, kezelni kell a hibát.
3. **JSON\_PRETTY\_PRINT használata fejlesztési célokra**:

echo json\_encode(['status' => 'success', 'data' => $result->fetch\_assoc()], JSON\_PRETTY\_PRINT);

Ezzel a függvénnyel **biztonságosan és hatékonyan** lehet **egy adott lakás adatait lekérni az API-ból**!



**createApartment($dbconn)**

Ez a függvény egy új **lakás (apartment)** létrehozását végzi az adatbázisban. A kliens **JSON formátumban** küldi el az új lakás adatait, és a függvény ezeket beilleszti az apartments táblába. Nézzük meg részletesen, hogy hogyan működik!

**1. Függvény deklarálása**

function createApartment($dbconn) {

* **Mit csinál?**
  + Létrehoz egy createApartment nevű függvényt.
  + **Bemeneti paramétere:** $dbconn, amely a MySQL adatbázis kapcsolatot jelöli.

**2. Kliens által küldött adatok beolvasása**

$data = json\_decode(file\_get\_contents('php://input'), true);

* **Mit csinál?**
  + file\_get\_contents('php://input'): Beolvassa a **POST kérés törzsét**, amely JSON formátumban tartalmazza az új lakás adatait.
  + json\_decode(..., true): Átalakítja a JSON-t **asszociatív tömbbé**.
* **Példa bejövő JSON adatra (amit a kliens küldhet):**

{

"address": "Kossuth Lajos utca 10.",

"city": "Budapest",

"postal\_code": "1053",

"size": 85,

"rooms": 3,

"price": 75000000,

"owner\_name": "Kovács János",

"owner\_phone": "+36201234567",

"description": "Kiváló állapotú lakás a belvárosban",

"image\_url": "https://example.com/lakas1.jpg"

}

**3. Ellenőrzi, hogy van-e bejövő adat**

if (!$data) {

http\_response\_code(400);

echo json\_encode(['status' => 'error', 'message' => 'Hiányos adatok.']);

return;

}

* **Mit csinál?**
  + Ha $data értéke **null vagy false**, akkor azt jelenti, hogy a JSON nem volt megfelelő.
  + **400 Bad Request hibakódot** küld vissza és egy **hibaüzenetet** ad a kliensnek.
* **Példa hibás válaszra:**

{

"status": "error",

"message": "Hiányos adatok."

}

**4. SQL lekérdezés előkészítése**

$query = "INSERT INTO apartments (address, city, postal\_code, size, rooms, price, owner\_name, owner\_phone, description, image\_url)

VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";

* **Mit csinál?**
  + Létrehoz egy **paraméterezett SQL INSERT lekérdezést**, amely egy új lakás adatait menti az apartments táblába.
  + A ? helyettesítők (placeholders) biztosítják, hogy a beillesztendő adatok **biztonságosan legyenek kezelve**.

**5. Lekérdezés előkészítése**

$stmt = $dbconn->prepare($query);

* **Mit csinál?**
  + Előkészíti az SQL lekérdezést ($stmt), hogy **paraméterezett lekérdezést** tudjunk végrehajtani.
  + **Miért fontos?**
    - **SQL injection elleni védelem**: megakadályozza, hogy rosszindulatú SQL kódokat adjanak meg az adatok között.
    - **Gyorsabb végrehajtás**: az SQL szerver optimalizálni tudja az előkészített lekérdezéseket.

**6. Paraméterek hozzárendelése és adatok beszúrása**

$stmt->bind\_param('sssiiissss',

$data['address'], $data['city'], $data['postal\_code'], $data['size'], $data['rooms'],

$data['price'], $data['owner\_name'], $data['owner\_phone'], $data['description'], $data['image\_url']);

* **Mit csinál?**
  + **Hozzárendeli az adatokat** a megfelelő paraméterekhez.
  + **Mit jelentenek az 'sssiiissss' betűk?**
    - s (**string**) → address, city, postal\_code, owner\_name, owner\_phone, description, image\_url
    - i (**integer**) → size, rooms, price
  + **Miért fontos?**
    - Biztosítja, hogy az adatokat **helyes típusban** adjuk át az adatbázisnak.
    - Elősegíti a hatékonyabb SQL végrehajtást.

**7. SQL parancs végrehajtása**

if ($stmt->execute()) {

* **Mit csinál?**
  + Ha az SQL parancs **sikeresen lefut**, akkor folytatja a sikeres válaszküldést.
  + Ha **hiba történik**, akkor a false értéket kapja vissza.

**8. Sikeres adatbeillesztés esetén válaszküldés**

http\_response\_code(201);

echo json\_encode(['status' => 'success', 'message' => 'Lakás sikeresen létrehozva.', 'id' => $stmt->insert\_id]);

* **Mit csinál?**
  + **201 Created** státuszkódot állít be → ez szabványos válaszkód az új rekordok létrehozására.
  + JSON válaszként visszaküldi a sikeres üzenetet és az **újonnan létrehozott rekord id-ját**.
* **Példa sikeres válaszra:**

{

"status": "success",

"message": "Lakás sikeresen létrehozva.",

"id": 12

}

**9. Hiba esetén válaszküldés**

} else {

http\_response\_code(500);

echo json\_encode(['status' => 'error', 'message' => 'Hiba az adat létrehozásakor.']);

}

* **Mit csinál?**
  + Ha az execute() nem sikerült (pl. adatbázis hiba miatt), akkor **500 Internal Server Error** státuszkódot állít be.
  + JSON válaszként visszaküldi a **hibaüzenetet**.
* **Példa sikertelen válaszra:**

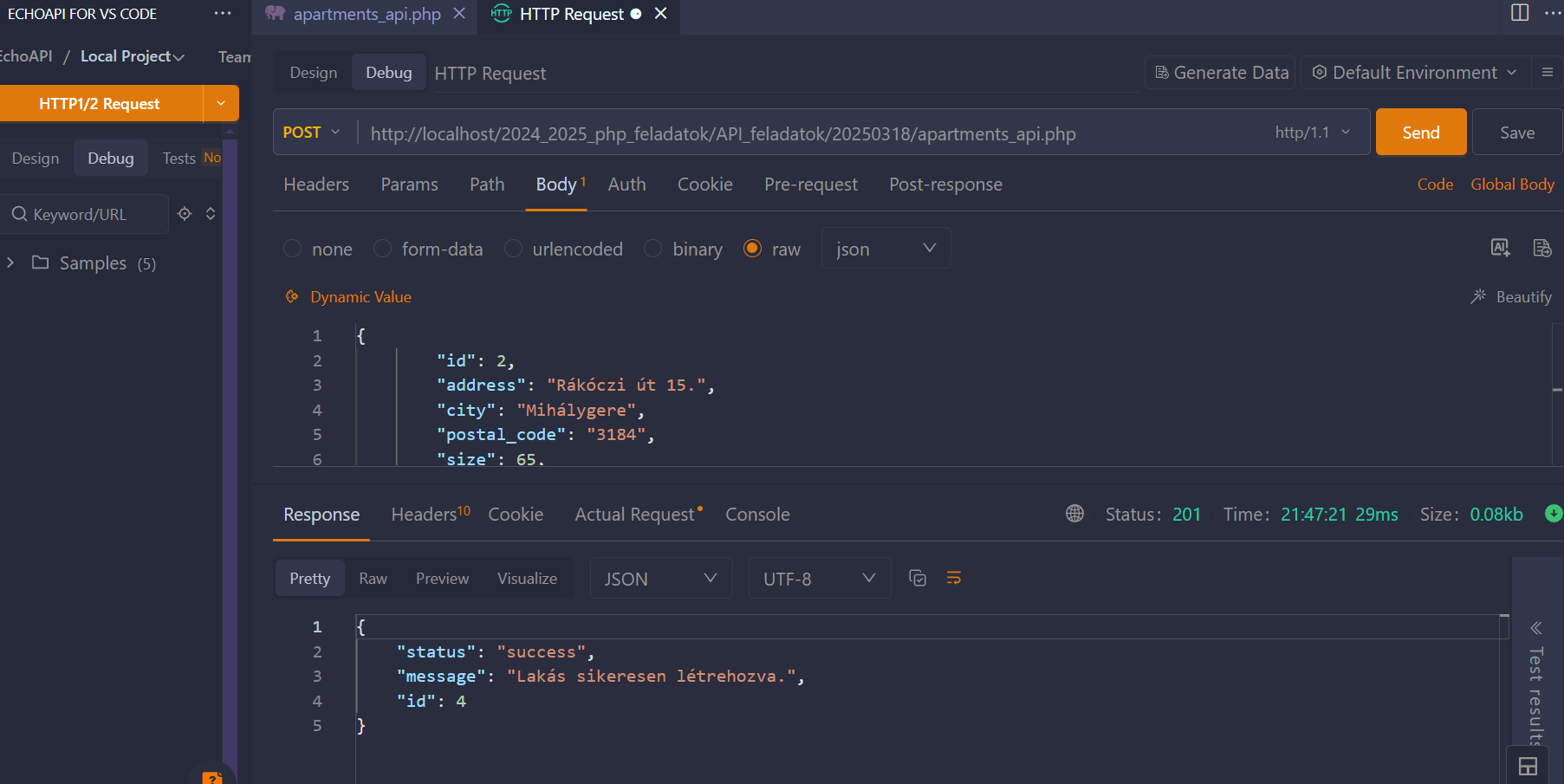
{

"status": "error",

"message": "Hiba az adat létrehozásakor."

}

|  |  |  |  |
| --- | --- | --- | --- |
| **Sor** | **Kód** | **Mit csinál?** | **Miért fontos?** |
| 1. | function createApartment($dbconn) { | Függvény létrehozása. | Az API ezen keresztül hívható. |
| 2. | $data = json\_decode(file\_get\_contents('php://input'), true); | JSON adatok beolvasása a kérésből. | A kliens így küldhet adatokat az API-nak. |
| 3. | if (!$data) {...} | Ellenőrzi, hogy van-e adat. | Hibaüzenetet küld, ha nincs megfelelő bemenet. |
| 4. | $query = "INSERT INTO apartments ..."; | SQL beszúró lekérdezés előkészítése. | Biztosítja, hogy az adatokat helyesen illeszti be. |
| 5. | $stmt = $dbconn->prepare($query); | Lekérdezés előkészítése. | SQL injection elleni védelem. |
| 6. | $stmt->bind\_param(...); | Paraméterek hozzárendelése. | Megakadályozza a hibás adatokat és támadásokat. |
| 7. | if ($stmt->execute()) {...} | SQL végrehajtása. | Ha sikeres, visszaadja az új rekord ID-ját. |

****

**updateApartment($id, $dbconn)**

Ez a függvény egy **létező lakás adatainak frissítését** végzi el az apartments táblában. A kliens **JSON formátumban** küldi az új adatokat, és a függvény ezeket **SQL UPDATE paranccsal** módosítja.

**1. Függvény deklarálása**

function updateApartment($id, $dbconn) {

* **Mit csinál?**
  + Létrehoz egy updateApartment nevű függvényt.
  + **Bemeneti paraméterek:**
    - $id: A frissítendő lakás azonosítója.
    - $dbconn: A MySQL adatbázis kapcsolat.

**2. Kliens által küldött adatok beolvasása**

$data = json\_decode(file\_get\_contents('php://input'), true);

* **Mit csinál?**
  + file\_get\_contents('php://input'): Beolvassa a **PUT kérés törzsét**, amely JSON formátumban tartalmazza a frissítendő adatokat.
  + json\_decode(..., true): Átalakítja a JSON-t **asszociatív tömbbé**.
* **Példa bejövő JSON adatra (amit a kliens küldhet):**

{

"address": "Kossuth Lajos utca 15.",

"city": "Budapest",

"postal\_code": "1053",

"size": 90,

"rooms": 4,

"price": 78000000,

"owner\_name": "Kovács Péter",

"owner\_phone": "+36201234567",

"description": "Felújított lakás a belvárosban",

"image\_url": "https://example.com/lakas2.jpg"

}

**3. Ellenőrzi, hogy van-e bejövő adat**

if (!$data) {

http\_response\_code(400);

echo json\_encode(['status' => 'error', 'message' => 'Hiányos adatok.']);

return;

}

* **Mit csinál?**
  + Ha $data értéke **null vagy false**, akkor azt jelenti, hogy a JSON nem volt megfelelő.
  + **400 Bad Request hibakódot** küld vissza és egy **hibaüzenetet** ad a kliensnek.
* **Példa hibás válaszra:**

{

"status": "error",

"message": "Hiányos adatok."

}

**4. SQL lekérdezés előkészítése**

$query = "UPDATE apartments SET address = ?, city = ?, postal\_code = ?, size = ?, rooms = ?, price = ?, owner\_name = ?, owner\_phone = ?, description = ?, image\_url = ? WHERE id = ?";

* **Mit csinál?**
  + Az SQL UPDATE lekérdezést **paraméterezett módon** hozza létre, hogy frissítsen egy lakást az id alapján.
  + A ? helyettesítők (placeholders) biztosítják, hogy az adatok **biztonságosan legyenek kezelve**.

**5. Lekérdezés előkészítése**

$stmt = $dbconn->prepare($query);

* **Mit csinál?**
  + Előkészíti az SQL lekérdezést ($stmt), hogy **paraméterezett lekérdezést** tudjunk végrehajtani.
  + **Miért fontos?**
    - **SQL injection elleni védelem**: megakadályozza, hogy rosszindulatú SQL kódokat adjanak meg az adatok között.
    - **Gyorsabb végrehajtás**: az SQL szerver optimalizálni tudja az előkészített lekérdezéseket.

**6. Paraméterek hozzárendelése és adatok frissítése**

$stmt->bind\_param('sssiiissssi',

$data['address'], $data['city'], $data['postal\_code'], $data['size'], $data['rooms'],

$data['price'], $data['owner\_name'], $data['owner\_phone'], $data['description'], $data['image\_url'], $id);

* **Mit csinál?**
  + **Hozzárendeli az adatokat** a megfelelő paraméterekhez.
  + **Mit jelentenek az 'sssiiissssi' betűk?**
    - s (**string**) → address, city, postal\_code, owner\_name, owner\_phone, description, image\_url
    - i (**integer**) → size, rooms, price, id
  + **Miért fontos?**
    - Biztosítja, hogy az adatokat **helyes típusban** adjuk át az adatbázisnak.
    - Elősegíti a hatékonyabb SQL végrehajtást.

**7. SQL parancs végrehajtása**

if ($stmt->execute()) {

* **Mit csinál?**
  + Ha az SQL parancs **sikeresen lefut**, akkor folytatja a sikeres válaszküldést.
  + Ha **hiba történik**, akkor a false értéket kapja vissza.

**8. Sikeres adatfrissítés esetén válaszküldés**

http\_response\_code(200);

echo json\_encode(['status' => 'success', 'message' => 'Lakás sikeresen frissítve.']);

* **Mit csinál?**
  + **200 OK** státuszkódot állít be → ez szabványos válaszkód a módosított rekordok esetén.
  + JSON válaszként visszaküldi a **sikeres üzenetet**.
* **Példa sikeres válaszra:**

{

"status": "success",

"message": "Lakás sikeresen frissítve."

}

**9. Hiba esetén válaszküldés**

} else {

http\_response\_code(500);

echo json\_encode(['status' => 'error', 'message' => 'Hiba a frissítés során.']);

}

* **Mit csinál?**
  + Ha az execute() nem sikerült (pl. adatbázis hiba miatt), akkor **500 Internal Server Error** státuszkódot állít be.
  + JSON válaszként visszaküldi a **hibaüzenetet**.
* **Példa sikertelen válaszra:**

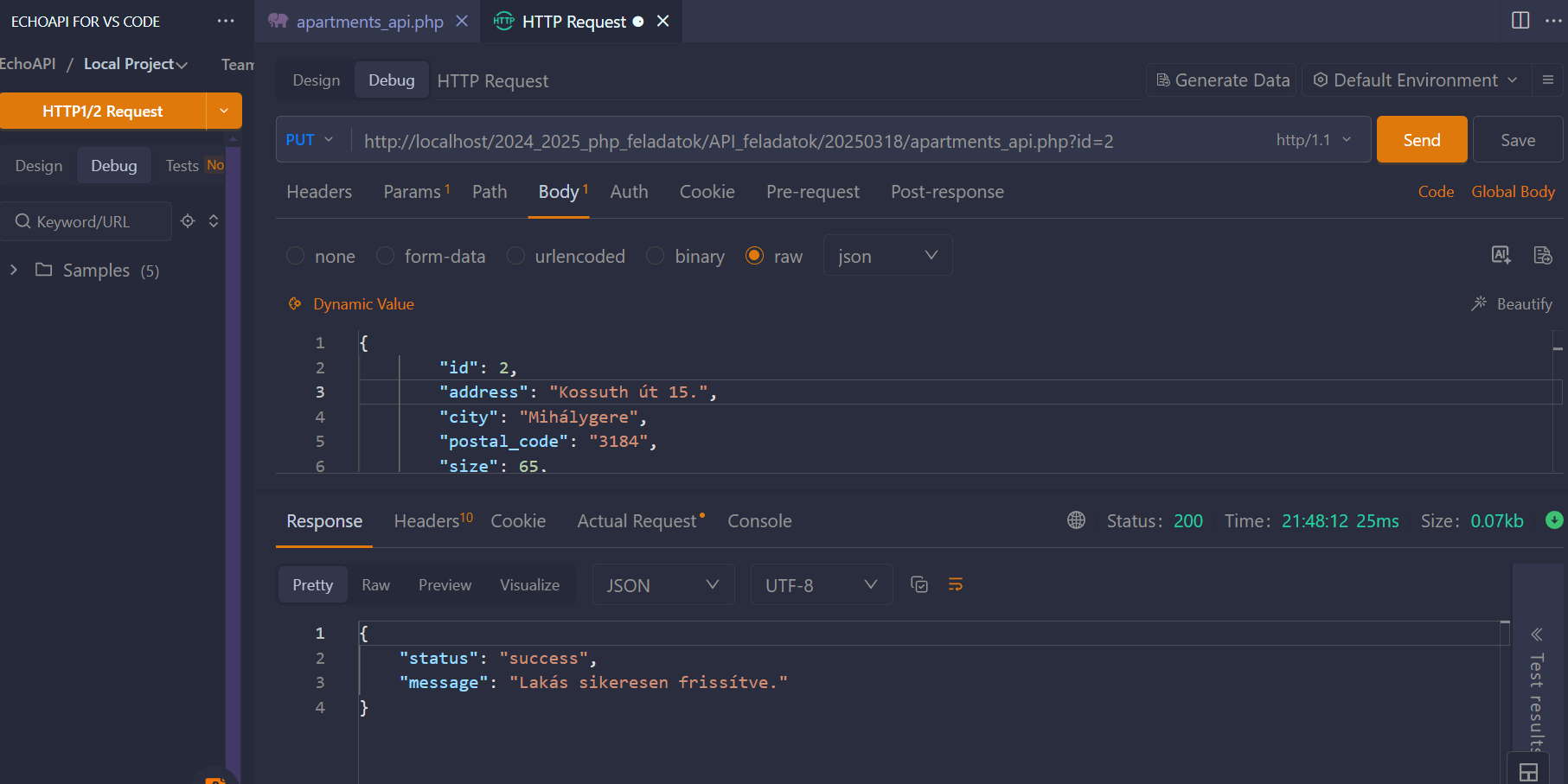
{

"status": "error",

"message": "Hiba a frissítés során."

}

|  |  |  |  |
| --- | --- | --- | --- |
| **Sor** | **Kód** | **Mit csinál?** | **Miért fontos?** |
| 1. | function updateApartment($id, $dbconn) { | Függvény létrehozása. | Az API ezen keresztül hívható. |
| 2. | $data = json\_decode(file\_get\_contents('php://input'), true); | JSON adatok beolvasása a kérésből. | A kliens így küldhet adatokat az API-nak. |
| 3. | if (!$data) {...} | Ellenőrzi, hogy van-e adat. | Hibaüzenetet küld, ha nincs megfelelő bemenet. |
| 4. | $query = "UPDATE apartments SET ... WHERE id = ?"; | SQL frissítő lekérdezés előkészítése. | Biztosítja, hogy az adatokat helyesen módosítja. |
| 5. | $stmt = $dbconn->prepare($query); | Lekérdezés előkészítése. | SQL injection elleni védelem. |
| 6. | $stmt->bind\_param(...); | Paraméterek hozzárendelése. | Megakadályozza a hibás adatokat és támadásokat. |
| 7. | if ($stmt->execute()) {...} | SQL végrehajtása. | Ha sikeres, visszaad egy sikerüzenetet. |



### deleteApartment($id, $dbconn)

Ez a függvény egy **lakás törlését végzi** az apartments táblából az id alapján. A törlést **SQL DELETE paranccsal** hajtja végre, és megfelelő HTTP státuszkódokkal ad visszajelzést.

### ****1. Függvény deklarálása****

function deleteApartment($id, $dbconn) {

* **Mit csinál?**
  + Létrehoz egy deleteApartment nevű függvényt.
  + **Bemeneti paraméterek:**
    - $id: A törlendő lakás azonosítója.
    - $dbconn: A MySQL adatbázis kapcsolat.

### ****2. SQL lekérdezés előkészítése****

$query = "DELETE FROM apartments WHERE id = ?";

* **Mit csinál?**
  + Az SQL DELETE lekérdezést **paraméterezett módon** hozza létre.
  + Az id alapján keresi a törlendő lakást.
  + A ? helyettesítő (placeholder) biztosítja, hogy az adat **biztonságosan** legyen kezelve.

### ****3. Lekérdezés előkészítése****

$stmt = $dbconn->prepare($query);

* **Mit csinál?**
  + Előkészíti az SQL lekérdezést ($stmt), hogy **paraméterezett lekérdezést** tudjunk végrehajtani.
  + **Miért fontos?**
    - **SQL injection elleni védelem**: megakadályozza, hogy rosszindulatú SQL kódokat adjanak meg az adatok között.
    - **Gyorsabb végrehajtás**: az SQL szerver optimalizálni tudja az előkészített lekérdezéseket.

### ****4. Paraméterek hozzárendelése****

$stmt->bind\_param('i', $id);

* **Mit csinál?**
  + Az id értékét hozzárendeli a **lekérdezés ? helyettesítőjéhez**.
  + 'i' azt jelenti, hogy az id **egész szám (integer)** típusú érték.
* **Miért fontos?**
  + Biztosítja, hogy az id mindig szám legyen.
  + Elősegíti a hatékonyabb SQL végrehajtást.

### ****5. SQL parancs végrehajtása és sikeres törlés ellenőrzése****

if ($stmt->execute()) {

* **Mit csinál?**
  + Ha az SQL parancs **sikeresen lefut**, akkor folytatja a sikeres válaszküldést.
  + Ha **hiba történik**, akkor a false értéket kapja vissza.

### ****6. Sikeres törlés esetén válaszküldés****

http\_response\_code(204);

echo json\_encode(['status' => 'success', 'message' => 'Lakás sikeresen törölve lett.']);

* **Mit csinál?**
  + **204 No Content** státuszkódot állít be → ez szabványos válaszkód a törlési műveletekhez.
  + JSON válaszként visszaküldi a **sikeres üzenetet**.
* **Példa sikeres válaszra:**

{

"status": "success",

"message": "Lakás sikeresen törölve lett."

}

### ****7. Hiba esetén válaszküldés****

} else {

http\_response\_code(500);

echo json\_encode(['status' => 'error', 'message' => 'Hiba a törlés során.']);

}

* **Mit csinál?**
  + Ha az execute() nem sikerült (pl. adatbázis hiba miatt), akkor **500 Internal Server Error** státuszkódot állít be.
  + JSON válaszként visszaküldi a **hibaüzenetet**.
* **Példa sikertelen válaszra:**

{

"status": "error",

"message": "Hiba a törlés során."

}

### ****8. Adatbáziskapcsolat lezárása****

mysqli\_close($dbconn);

* **Mit csinál?**
  + **Lezárja az adatbáziskapcsolatot**.
  + Ez segít a szerver erőforrásainak **hatékonyabb kezelésében**.
* **Megjegyzés:**
  + Ha a deleteApartment() függvényt egy másik PHP fájlban használjuk, és ott több SQL művelet is történik, akkor **nem mindig szükséges itt lezárni a kapcsolatot**.

|  |  |  |  |
| --- | --- | --- | --- |
| **Sor** | **Kód** | **Mit csinál?** | **Miért fontos?** |
| 1. | function deleteApartment($id, $dbconn) { | Függvény létrehozása. | Az API ezen keresztül hívható. |
| 2. | $query = "DELETE FROM apartments WHERE id = ?"; | SQL törlő lekérdezés előkészítése. | Biztosítja, hogy az adatokat helyesen módosítja. |
| 3. | $stmt = $dbconn->prepare($query); | Lekérdezés előkészítése. | SQL injection elleni védelem. |
| 4. | $stmt->bind\_param('i', $id); | Paraméter hozzárendelése. | Megakadályozza a hibás adatokat és támadásokat. |
| 5. | if ($stmt->execute()) {...} | SQL végrehajtása. | Ha sikeres, visszaad egy sikerüzenetet. |
| 6. | http\_response\_code(204); | Ha sikeres a törlés, 204 No Content küldése. | A REST API konvenciók betartása. |
| 7. | http\_response\_code(500); | Hiba esetén 500-as kódot küld. | Hibakezelés a kliensek számára. |
| 8. | mysqli\_close($dbconn); | Adatbáziskapcsolat lezárása. | Erőforrások megfelelő kezelése. |

### ****Lehetséges továbbfejlesztések****

1. **Ellenőrzés, hogy az id létezik-e a törlés előtt**
   * Ha az id nem található az adatbázisban, akkor 404 Not Found státuszkódot kellene visszaadni.

$checkQuery = "SELECT id FROM apartments WHERE id = ?";

$checkStmt = $dbconn->prepare($checkQuery);

$checkStmt->bind\_param('i', $id);

$checkStmt->execute();

$result = $checkStmt->get\_result();

if ($result->num\_rows === 0) {

http\_response\_code(404);

echo json\_encode(['status' => 'error', 'message' => 'A megadott lakás nem található.']);

return;

}

1. **Soft delete implementálása**
   * Ahelyett, hogy véglegesen törölnénk az adatokat, inkább jelölhetjük őket töröltként (is\_deleted = 1), így később visszaállíthatók:

$query = "UPDATE apartments SET is\_deleted = 1 WHERE id = ?";

1. **Hibakódok pontosítása**
   * Ha az adatbázis kapcsolat hibás, **502 Bad Gateway** jobb választás lehet az 500 helyett.