REST API kurzus

Tartalomjegyzék

Bevezetés a REST API-ba

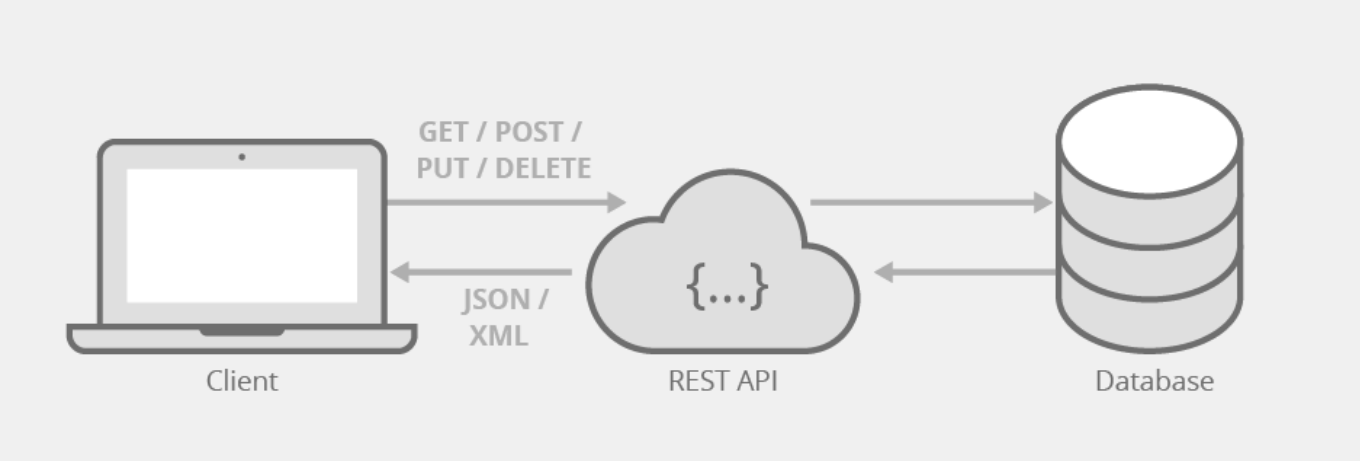
Mi az a REST API

A REST (Representational State Transfer) API egy olyan szoftverarchitektúra, amely lehetővé teszi a kliens és a szerver közötti kommunikációt a HTTP protokollon keresztül. A REST API-k az internetes alkalmazások közötti adatcserét teszik lehetővé, és alapvetően a kérések (HTTP kérések) és válaszok (HTTP válaszok) formájában működnek.

A REST API alapelvei Roy Fielding "Architectural Styles and the Design of Network-based Software Architectures" című disszertációjából származnak, amelyet 2000-ben írt. Fielding ebben a disszertációban lefektette az ún. REST architektúrális stílusát, amely az internetes protokollok, különösen a HTTP számára alkalmas módot kínál a hálózaton elérhető információk megosztására.

A REST API-k előnyei közé tartozik a könnyű skálázhatóság, az egyszerűség, a platformfüggetlenség és a könnyű érthetőség. Ez a módszer nagyon elterjedt a webes alkalmazások fejlesztésében, és ma már számos

olyan szolgáltatás, alkalmazás és webhely használja, amelyek lehetővé teszik a felhasználók számára a különböző adatok elérését és kezelését.



Az API-ok szerepe a modern webfejlesztésben

Az API-k (alkalmazásprogramozási interfészek) nagyon fontos szerepet töltenek be a modern webfejlesztésben, és számos ok miatt alapvető fontosságúvá váltak:

1. **Modularitás és újrafelhasználhatóság**: Az API-k lehetővé teszik a funkcionalitások moduláris felépítését, így különböző alkalmazások könnyen újrahasznosíthatják ezeket a modulokat. Ezáltal időt és erőforrásokat lehet megtakarítani a fejlesztés során.
2. **Platformfüggetlenség**: Az API-k lehetővé teszik az alkalmazásoknak, hogy különböző platformokon (például webböngészők, mobilalkalmazások, szerverek stb.) működjenek anélkül, hogy szorosan összekapcsolódnának az adott platformmal. Ez a rugalmasság lehetővé teszi az alkalmazások számára, hogy többféle eszközön és platformon működjenek.
3. **Adatmegosztás és integráció**: Az API-k lehetővé teszik az adatok megosztását és integrálását más alkalmazásokkal vagy szolgáltatásokkal. Ez lehetővé teszi az alkalmazások közötti összekapcsolást és az adatok könnyű áramlását, ami gazdagabb felhasználói élményt eredményez.
4. **Közösségi fejlesztés**: Az API-k lehetővé teszik más fejlesztők számára, hogy alkalmazásokat és szolgáltatásokat építsenek a meglévő API-okra. Ez egy nyitott ökoszisztémát teremt, amelyben az innováció és a kreativitás virágozhat.
5. **Skálázhatóság**: Az API-k lehetővé teszik az alkalmazásoknak, hogy könnyen skálázhatóak legyenek a felhasználói igények növekedése esetén. Az új funkciók hozzáadása vagy a kapacitás növelése gyakran egyszerűen megvalósítható az API-k révén.

Alapelvek és jellemzők

1. URL-ek (Uniform Resource Locator): Az erőforrásokat (adatokat, szolgáltatásokat stb.) egyedi URL-eken keresztül azonosítják. Ezek a URL-ek egyszerű és értelmezhető formátumban jelennek meg, és általában hierarchikusan szerveződnek.
2. HTTP metódusok: Az HTTP protokoll által definiált metódusokat (GET, POST, PUT, DELETE stb.) használják az adatok kezelésére és manipulálására az erőforrásokkal. Például a GET kérés az erőforrások lekérdezésére, a POST kérés az adatok létrehozására, a PUT kérés az adatok frissítésére és a DELETE kérés az adatok törlésére szolgál.
3. Állapotkódok (HTTP status codes): Az állapotkódokat az HTTP válaszok részeként küldik vissza, és információt nyújtanak a kliensnek az adott kérés eredményéről. Például a 200-as sorozat jelzi a sikeres

kérést, a 400-as sorozat jelzi a kliens hibáját, míg a 500-as sorozat jelzi a szerver hibáját.
4. Állapotmentesség (Statelessness): A REST architektúra állapotmentes, ami azt jelenti, hogy minden egyes kérés kliens által hordozza az összes szükséges információt a szervernek a válasz megfelelő feldolgozásához. A szerver nem tárol semmilyen ügyfél-specifikus információt a kérések között.
5. Reprezentáció (Representation): Az adatokat általában különböző formátumokban (pl. JSON, XML, HTML stb.) reprezentálják, és a kliens kérése alapján a szerver válasza megfelelő formátumban jelenik meg.
6. Egyediség (Uniform Interface): A REST API-knak egységes interfészt kell biztosítaniuk az erőforrásokhoz való hozzáférésre és kezelésre. Ez magában foglalja az egységes URL-eket, az HTTP metódusok használatát, valamint a közvetlen és egyértelmű reprezentációt az erőforrások állapotáról.

A Backend felépítése

Környezet beállítása

1. XAMPP konfiguráció:
    - Indítsa el a XAMPP vezérlőpaneljét.
    - Ellenőrizze, hogy az Apache és a MySQL szerverek futnak-e. Ha nem, indítsa el őket a megfelelő gombokra kattintva.
2. Apache konfiguráció:
    - Az Apache szerver beállításai alapvetően a "httpd.conf" fájlban találhatók. Ezt a fájlt általában a XAMPP telepítési könyvtárában (pl. C:\xampp\apache\conf) találja meg.
    - Nyissa meg a "httpd.conf" fájlt egy szövegszerkesztőben.
    - Itt módosíthatja az Apache beállításait, például a portszámokat vagy a virtuális hosztokat, ha szükséges.

MYSQL adatbázis létrehozása

1. Indítsa el a XAMPP vezérlőpaneljét, és győződjön meg róla, hogy a MySQL szerver fut.
2. Nyissa meg a böngészőjét, és látogasson el a http://localhost/phpmyadmin címre, hogy megnyissa a phpMyAdmin felhasználói felületet.
3. Kattintson az "Új létrehozása" gombra az adatbázis létrehozásához.
4. Adja meg az adatbázis nevét (pl. "mydatabase"), majd kattintson az "OK" gombra az adatbázis létrehozásához.
5. Hozzon létre táblákat az újonnan létrehozott adatbázisban a "Tábla létrehozása" menüpont segítségével.
6. Ha ki van választva az adatbázis, akkor az "Importálás" menüpontra kattintv importálhatóak a már meglévő táblák.

PHP backend kapcsolása az adatbázishoz

```php
<?php
    // Adatbázis kapcsolódási adatok
    $servername = "localhost";
```

```
    $username = "root"; // Alapértelmezett felhasználónév XAMPP esetén
    $password = ""; // Alapértelmezett jelszó XAMPP esetén
    $database = "mydatabase"; // Az ön adatbázisának neve

    // Kapcsolódás az adatbázishoz
    $conn = new mysqli($servername, $username, $password, $database);

    // Kapcsolódás ellenőrzése
    if ($conn->connect_error) {
        die("Sikertelen kapcsolódás az adatbázishoz: " . $conn->connect_error);
    } else {
        echo "Sikeres kapcsolódás az adatbázishoz!";
    }

    //Függvények

    // Kapcsolat bezárása
    $conn->close();
?>
```

Elérési pontok meghatározása

Az alkalmazásban az egyes erőforrásokhoz tartozó elérési pontokat kell meghatározni. Például:

GET /users: Az összes felhasználó lekérése.
GET /users/{id}: Egy adott felhasználó lekérése az azonosító alapján.
POST /users: Új felhasználó létrehozása.
PUT /users/{id}: Egy adott felhasználó frissítése az azonosító alapján.
DELETE /users/{id}: Egy adott felhasználó törlése az azonosító alapján.

Kérések kezelése a backend oldalon

Az egyes elérési pontokhoz tartozó kéréseket megfelelően kell kezelni a backend oldalon. Ez magában foglalja az adatok lekérdezését, frissítését, törlését stb. például adatbázisból vagy külső szolgáltatásokból.

Paraméterek és adatok kezelése

Az URL-ekben és a kérésekben található paramétereket (pl. azonosítók, szűrési feltételek stb.) és adatokat (pl. felhasználói adatok, beállítások stb.) megfelelően kell kezelni a backend oldalon.

HTTP metódusok kezelése

A különböző HTTP metódusokat (GET, POST, PUT, DELETE stb.) megfelelően kell kezelni az elérési pontokhoz tartozó műveletek szerint.
Ezen metódusok kezelését php forrású backend-en egy if-es szerkezettel lehet megvalósítani.

```
<?php
   if($_SERVER['REQUEST_METHOD'] == 'GET') { //ha 'get' metódust használunk
       //válasz lekérése és elküldése
   } elseif ($_SERVER['REQUEST_METHOD'] == 'POST'){ //ha 'POST' metódust
```

```
használunk
        //elem létrehozása
    }elseif ($_SERVER['REQUEST_METHOD'] == 'PUT'){ //ha 'PUT' metódust használunk
        //elem frissítése
    }elseif ($_SERVER['REQUEST_METHOD'] == 'DELETE'){ //ha 'DELETE' metódust
használunk
        //elem törlése
    }
?>
```

Válaszok elküldése

A backendnek megfelelő válaszokat kell küldenie a kérésekre, például JSON vagy XML formátumban az adatokkal és az állapotkóddal együtt.
A válaszokat a '*header()*', az állapotkódokat a '*http_response_code()*' php függvények segítségével tudjuk elküldeni.

```
    header('Content-Type: application/json'); // a header szintaxisa
    echo json_encode(['object_name' => $response_variable]); // a json típusú
adattömböt egyszerűen kiirjuk

    //Státuszkód küldése:
    http_response_code(200); // OK
```

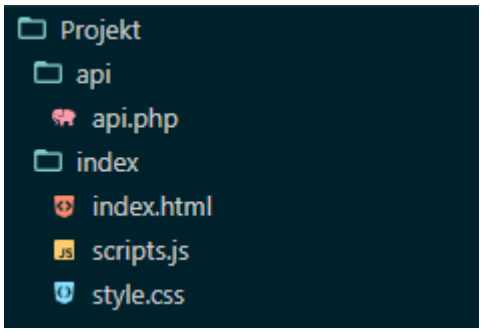
Kód	Jelentés
200	OK
201	Created
202	Accepted
203	Non-Authoritative Information
204	No Content
301	Moved Permanently
302	Found
304	Not Modified
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
500	Internal Server Error

Kód	Jelentés
503	Service Unavailable

Frontend felépítése

A projekt inicializálása

1. Először is, hozzuk létre a projekt mappáját, majd rendezzük el a szükséges fájlokat. Az álatlunk megírt API kódját érdemes elkülönítve tárolnunk.



2. Ha használni szeretnénk, készítsük el az adatbázist a phpMyadmin oldalon.
3. Az API fájlban írjuk meg a szükséges függvényeket.
4. Készítsük el a frontend kinézeti elemeit gombokkal, bemeneti mezőkkel.
5. Adjuk meg a frontend stílusát a styles.css fájlban.

Kérések küldése az API-hoz

Az api-val való kommunikációt javascript fogja biztosítani.

1. A JavaScript fájlban írjuk meg a FETCH API függvényét

```
async function api() {
     const response = await fetch("../api/api.php", {method: "GET"}); //
használni kívánt metódus GET/POST/PUT/DELETE
     const data = await response.json(); // eltárolja a választ
     var válasz = data.value; // a 'value' helyére írjuk azt az
objektumkulcsot amelyben a korábban megírt API-unk a megfelelő adatot
tárolta
 }
```

2. Hívjuk meg a JavaScript függvényt az api használatához.

```
 <input type="button" value="HTTP kérelem küldése" name="Button1"
id="button1" onclick="api()">
```

3. Jelenítsük meg a JavaScriptben megkapott eredményt.

Gyakorlati mintafeladat

Egyszerű példa

Az első feladat egy egyszerű API megírása, ehhez adatbázist sem fogunk használni. A jelenlegi példában egy random jelszó generátort kell készíteni.

1. Hozzunk létre egy "*api*" nevű mappát és benne egy "*password.php*" nevű fájlt.
2. Írjunk egy egyszerű random jelszót generáló függvény és hívjuk meg.

```
function generate_random_string($length = 12) {
        $characters =
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!?_-#&@';
        $random_string = '';

        for ($i = 0; $i < $length; $i++) {
            $random_string .= $characters[rand(0, strlen($characters) - 1)];
        }

        return $random_string;
    }
```

```
if ($_SERVER['REQUEST_METHOD'] == 'GET') // lefut, ha a metódus GET
{
    $password = generate_random_string();
    header('Content-Type: application/json'); // a visszaadott érték json
formátumú
    echo json_encode(['password' => $password]); // válasz (response)
}
```

3. Írjuk meg a REST API-t. A JavaScript fetch függvényét használva indíthatunk kérelmeket (HTTPRequest) és újratöltés nélkül változtathatjuk az oldalt.

```
<!DOCTYPE html>
<html lang="hu">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Példa</title>
    <script>
    async function api() { // async, mert meg kell várnia, amíg a meghívott
fájl lefut és visszaküldi a választ
        const response = await fetch("../api/password.php", {method:
"GET"}); // meghívja az API-t GET metódussal
        const data = await response.json(); // eltárolja a választ
```

```
        document.getElementById('ujjelszo').innerText = data.password; //
kiírja a kapott adatokból azt, ami kell
    }
    </script>
</head>
<body>
    <input type="button" value="Jelszó generálása" name="Jelszo" id="jelszo"
onclick="api()">
    <p id="ujjelszo"></p>
</body>
</html>
```

Ha szükséges, a kész feladat megtalálható a kurzus mappájában.

Összetett példa

Most, hogy az alapok megvannak, jöhet egy összetettebb feladat.
Egy könyvtárkezelő oldalt kell írni, amihez adatbázis is tartozik.
Funkciók:

- Könyvek állapotának és adatainak lekérdezése
- Könyvek hozzáadása
- Könyvek törlése

1. Első feladat az adatbázis létrehozása. A fenti leírás szerint csinálja végig az adatbázis létrehozását és adatok importálását.
   Az importálandó fájl itt található: SQL (ha nem működik, megtalálja a kurzus mellett a mappában)
2. Form kialaktása:

```
<form>
        <h1>Keresés szerző alapján:</h1>
        <label for="sAu">Szerző neve: </label>
        <input type="text" id="sAu" required>
        <input type="button" value="Keresés" onclick="searchAuthor()">

        <h1>Keresés cím alapján:</h1>
        <label for="sTi">Cím: </label>
        <input type="text" id="sTi" required>
        <input type="button" value="Keresés" onclick="searchTitle()">

        <h1>Új könyv felvétele:</h1>
        <label for="aRcim">Cím: </label>
        <input type="text" id="aRcim" required>

        <label for="aRszerzo">Szerző: </label>
        <input type="text" id="aRszerzo" required>

        <input type="radio" id="available" name="elerheto" value="available"
required>
        <label for="available">Available</label><br>
```

```
        <input type="radio" id="not_available" name="elerheto"
value="not_available" required>
        <label for="not_available">Not Available</label><br>

        <input type="button" value="Felvétel" onclick="appendRecord()">

        <h1>Adat módosítása ID alapján:</h1>
        <label for="azon">ID: </label>
        <input type="text" id="azon" required>
        <label for="newAuthor">Új szerző (ha szükséges): </label>
        <input type="text" id="newAuthor">
        <label for="newTitle">Új cím (ha szükséges): </label>
        <input type="text" id="newTitle">
        <label for="newTitle">Új elérhetőség (ha szükséges): </label>
        <input type="radio" id="newavailable" name="newelerheto"
value="available" required>
        <label for="available">Available</label><br>
        <input type="radio" id="newnot_available" name="newelerheto"
value="not_available" required>
        <label for="not_available">Not Available</label><br>
        <input type="button" value="Módosítás" onclick="editRecord()">

        <h1>Törlés ID alapján:</h1>
        <label for="delID">ID: </label>
        <input type="number" id="delID" required>
        <input type="button" value="Törlés" onclick="deleteRecord()">
    </form>
```

3. Kapcsolódás az adatbázishoz.

```
// adatbázis kapcsolása
// Adatbázis kapcsolódási adatok
$servername = "localhost";
$username = "root"; // Alapértelmezett felhasználónév XAMPP esetén
$password = ""; // Alapértelmezett jelszó XAMPP esetén
$database = "books"; // Az ön adatbázisának neve
// Kapcsolódás az adatbázishoz
$conn = new mysqli($servername, $username, $password, $database);
```

4. Endpointok kezelése

```
$filePath = explode('/', $_SERVER['REQUEST_URI']);
```

5. REST API megírása minden metódusra:
    - GET (összes): Ha a kereső mező üres marad, akkor az összes könyvet kilistázza.
    - GET (author és title): Keresés szerző vagy cím alapján. Ugyanúgy kell mindkettőt. (Példa: author)
        - **API:**

```
else  if ($_SERVER['REQUEST_METHOD'] == "GET" &&
$filePath[count($filePath) - 2] == "author") // ha az endpoint
books.php/author/valami akkor ez fut le (valami a keresett szöveg)
{
    $sql = "SELECT * FROM books where author like
'%".$filePath[count($filePath) - 1]."%'";
    $result = $conn->query($sql);
    $books = array();
    if ($result->num_rows > 0) {
        while($row = $result->fetch_assoc()) {
            $books[] = $row;
        }
    }
    header('Content-Type: application/json');
    http_response_code();
    echo json_encode($books);
}
```

- **Javascript:**

```
async function searchAuthor()
{
    //eredményt tartalmazó div törlése ha már létezik
    var existingDiv = document.getElementById("div");
    if (existingDiv) {
        existingDiv.remove();
    }
    //eredményt tartalmazó div létrehozása
    var div = document.createElement('div');
    div.setAttribute("id", "div");
    document.body.appendChild(div);
    //endpoint és http request
    const response = await fetch("../api/books.php/author/" +
document.getElementById("sAu").value, {method: "GET"});
    const data = await response.json();
    //adatok kiírása
    for (let i = 0; i < data.length; i++)
    {
        var p = document.createElement('span');
        p.appendChild(document.createTextNode("Azonosító: " +
data[i].id + " Szerző: " + data[i].author + " Cím: " +
data[i].title + " Elérhetőség: " + data[i].availability));
        var br = document.createElement('br');
        div.appendChild(p);
        div.appendChild(br);
    }
}
```

- POST: Minden adatot megadva lehet új könyvet felvenni az adatbázisba.

- **API:**

```
else if ($_SERVER['REQUEST_METHOD'] == "POST") {
    $title = $filePath[count($filePath) - 3];
    $author = $filePath[count($filePath) - 2];
    $availability = $filePath[count($filePath) - 1];
    if ($availability == "not_available")
    {
        $availability = "not available";
    }
    $sql = "INSERT INTO `books` (`title`, `author`,
`availability`) VALUES
    ('$title', '$author', '$availability') ";
    try {
        $conn->query($sql);
        http_response_code(201);
    } catch (Exception $e) {
        http_response_code(400);
    }
}
```

- **Javascript:**

```
async function appendRecord() {
    const response = await fetch("../api/books.php/insert/" +
document.getElementById("aRcim").value+"/"+
document.getElementById("aRszerzo").value+"/"+
document.querySelector('input[name="elerheto"]:checked').value,
{method: "POST"});
}
```

- PUT: ID alapján módosítás.
  - **API:**

```
else if ($_SERVER['REQUEST_METHOD'] == "PUT")
{
    $title = $filePath[count($filePath) - 3];
    $author = $filePath[count($filePath) - 2];
    $availability = $filePath[count($filePath) - 1];
    $id = $filePath[count($filePath) - 4];
    //ha üres akkor nem kell változtatni, lekérdezzük az eddigit
    if ($title == "")
    {
        $title = $conn->query("select title from books where id =
$id");
        $title = $title->fetch_assoc()['title'];
    }
    if ($author == "")
```

```
    {
        $author = $conn->query("select author from books where id
= $id");
        $author = $author->fetch_assoc()['author'];
    }
    if ($availability == "")
    {
        $availability = $conn->query("select availability from
books where id = $id");
        $availability = $availability->fetch_assoc()
['availability'];
    }
    if ($availability == "not_available")
    {
        $availability = "not available";
    }
    $sql = "update books set title = '$title', author = '$author',
availability = '$availability' where id = $id";
    try {
        $conn->query($sql);
        http_response_code(201);
    } catch (Exception $e) {
        http_response_code(400);
    }
}
```

- **Javascript:**

```
async function editRecord() {
    const response = await fetch("../api/books.php/edit/" +
document.getElementById('azon').value +"/" +
document.getElementById('newTitle').value + "/" +
document.getElementById('newAuthor').value + "/" +
document.querySelector('input[name="newelerheto"]:checked').value,
{method: "PUT"});
}
```

- DELETE: Könvvek törlése.
  - **API:**

```
else if ($_SERVER["REQUEST_METHOD"] == "DELETE")
{
    $id = $filePath[count($filePath) - 1];
    $sql = "delete from books where id = $id";
    try {
        $conn->query($sql);
        http_response_code(201);
    } catch (Exception $e) {
        http_response_code(400);
```

```
    }
}
```

- **Javascript:**

```
async function deleteRecord() {
    const response = await fetch("../api/books.php/delete/" +
document.getElementById("delID").value, {method: "DELETE"})
}
```

Ezen még lehet javítani, például lehet ellenőrizni az adatok helyességét, de a REST API ezzel kész van.
Ha szükséges, a kész feladat megtalálható a kurzus mappájában.

Gyakorló feladat

Az utolsó feladatot önállóan kell elvégezni. A 3.feladat mappában megtalálható a szükséges adatbázis. Az importálandó fájl itt található: SQL (ha nem működik, megtalálja a kurzus mellett a mappában) Ebben a feladatban egy áruház raktárkészletét kell kezelnie.

1. feladat: Adatbázis létrehozása és csatlakozás PHP-val.
2. feladat: Esztétikus főoldal kialakítása (HTML, CSS)
3. feladat: Termékek lekérdezése és kiírása.
4. feladat: Keresés.
5. feladat: Termékek módosítása.
6. feladat: Termékek törlése.

Ellenőrizze a bevitt adatok helyességét (pl. mennyiség egész szám stb.) Használja a kurzusban tanult módszereket.