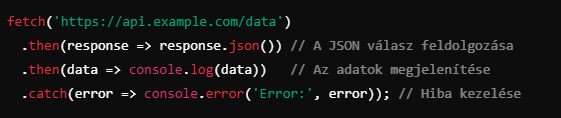
**Fetch API**

A **Fetch API** egy modern JavaScript eszköz, amely lehetővé teszi, hogy hálózati kéréseket küldjünk és válaszokat kezeljünk. Általában HTTP-kérések végrehajtására használják, például adatok lekérésére egy szerverről (GET) vagy adatok elküldésére (POST). Ez egy ígéret-alapú API (Promise), ami könnyebben kezelhetővé és olvashatóbbá teszi az aszinkron műveleteket.

A Fetch API-t a fetch() függvénnyel használjuk, amely egy Promise-t ad vissza. Ez a Promise két állapotban lehet:

* Resolved (megoldott): A kérelem sikeres volt, és visszaad egy Response objektumot.
* Rejected (elutasított): Hiba történt (pl. hálózati probléma).

Ez a módszer lehetővé teszi, hogy aszinkron módon kezeljük a válaszokat, tehát nem akad meg a program futása, amíg a válasz beérkezik.

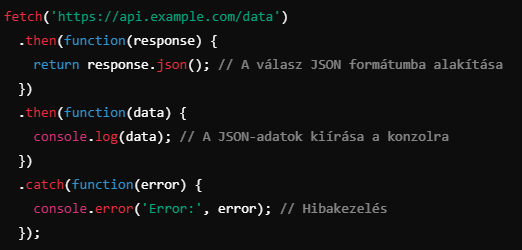


*Egy másik API: https://reqres.in/api/users/*

A példa részletesen:

Ez alapesetben egy GET típusú kérés. A későbbiekben megnézzük, hogyan lehet „bonyolítani” a fetch-et.

**A nyíl függvények használata:** . A => egy **arrow function** (nyílfüggvény) szintaxis JavaScript-ben.Ez egy kompaktabb és egyszerűbb módja függvények definiálásának, mint a hagyományos function kulcsszó használata. A fenti példa hagyományosan így nézne ki:



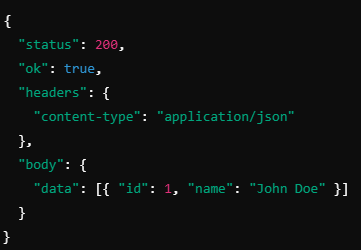
**Az első .then blokk:** Amikor a fetch() sikeresen lefut, visszaad egy Response objektumot, amely tartalmazza az API válaszát. A Response objektum tartalmazza az HTTP állapotkódot, a fejléceket és a válasz törzsét is. Azonban a válasz törzse (body) alapértelmezetten egy olvasható adatfolyam (stream), amit explicit módon kell feldolgoznunk, hogy tényleges adatokat kapjunk.

* response.json(): Ez egy beépített metódus, amely a válasz törzsét JSON formátumban dolgozza fel, és egy újabb Promise-t ad vissza. Ez azért szükséges, mert a JSON feldolgozása szintén aszinkron művelet.
  + Ha a válasz nem JSON, hibát dob.

A fetch() által visszaadott Response objektum a válasz adatait tartalmazza, de a tényleges tartalom (body) külön metódusokkal érhető el, például:

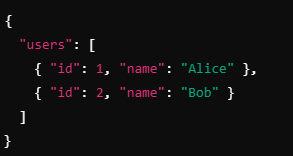
* .json(): JSON válasz feldolgozása
* .text(): Szöveges válasz
* .blob(): Bináris adatok (pl. képek, fájlok)
* .arrayBuffer(): Nyers bináris adatok

Példa egy Response objektumra:



**A második .then blokk:** Ebben a blokkban a response.json() által visszaadott adatokat kezeljük. A data paraméter tartalmazza az API válaszában lévő tényleges adatokat, amelyeket most már JavaScript objektumként érhetünk el.

Példa a data tartalomra:

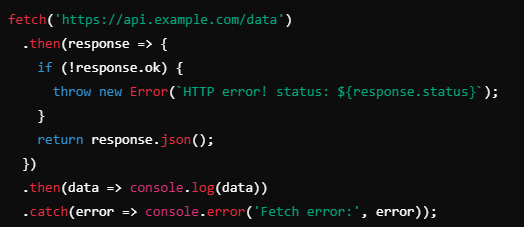


**A catch blokk:** Ez a blokk a hibakezelésért felelős. Ha a fetch() hívás vagy bármelyik then() belsejében hiba történik, akkor a catch() blokkba kerülünk. Példák a lehetséges hibákra:

* A megadott URL nem érhető el (hálózati hiba).
* A válasz nem JSON formátumú, és a response.json() hibát dob.
* Az API egy hibás HTTP választ adott vissza (pl. 404 vagy 500), amit manuálisan kell ellenőrizni.
* console.error(): A hibát a konzolon megjeleníti, hogy könnyebben hibakeresést végezhessünk.

**Hibakezelés:**

A fetch() csak akkor utasítja el a Promise-t, ha tényleges hálózati hiba történt. A HTTP hibák, mint a 404 vagy 500, nem eredményeznek automatikus Promise elutasítást. Ezért érdemes a response.ok tulajdonságot ellenőrizni.



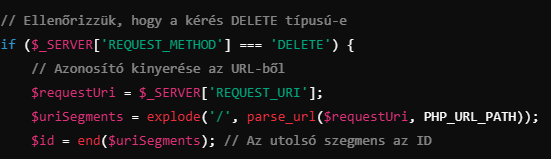
**POST kérés**

Adatok küldésére használható. A második paraméter egy objektum, amely tartalmazza a metódust, a fejlécet (headers), és a törzset (body).

****

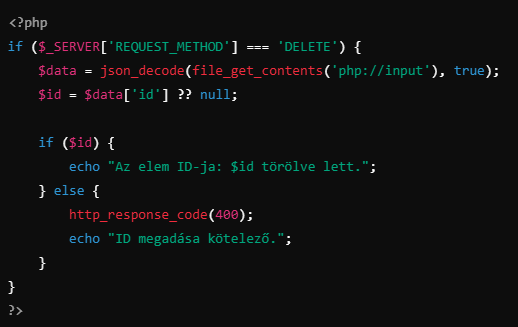
**DELETE kérés**

Adatok törlésére használható. A törlendő adat azonosítóját általában az URL-ben küldjük el, de itt is használhatunk body objektumot az információ továbbításához.

vagy





**PUT / PATCH kérés**

Használható meglévő adatok módosítására.



**Összefoglalva:**

Bár elsőre bonyolultabbnak tűnik mint a hagyományos XMLHttpRequest használata, ezzel a megoldással lehetővé válik, hogy „megüzenjük” a szerver oldalnak, milyen műveletet (GET, POST, DELETE stb.) szeretnénk végezni és elküldjük az ehhez szükséges adatokat.