NODE.JS FULLSTACK FEJLESZTÉS

2. forduló

A kategória támogatója: Emarsys

Ismertető a feladathoz

Útmutató:

- A **radio button-os kérdésekre** egy helyes válasz van.
- **Ha lejár a feladatlap ideje, a rendszer AUTOMATIKUSAN** beküldi azt az addig megjelölt válaszokkal.
- Az **adatbekérős feladatokra NEM jár részpontszám**, csak a feleletválasztósakra.
- **Badge-ket** a 4.forduló után kapsz majd először.
- Az **adatbekérős kérdéseknél** igyekeztünk minden variációt megadni (kisbetű, nagybetű, szóköz), de ha mégis eltérést tapasztalsz a megoldásokban, kérjük, jelezd felénk!

+1: Azért szólunk, hogy senkit ne a végén érjen meglepetés: a játék nem tipp-mix és csapatkategória sincs! Természetesen akinek nem inge...

Jó versenyzést kívánunk!

2.forduló - Fájlkezelés

Felhasznált idő: 00:00/15:00 **Elért pontszám:** 0/12

1. feladat 0/2 pont

Milyen flaget kell alkalmaznunk az alábbi függvényben **fs.open('sample.txt', ???)**, ha egy fájlt szeretnénk írásra és olvasásra megnyitni, ugyanakkor szeretnénk azt is, hogy a függvény hibát dobjon, ha a fájl nem létezik?

Válasz

- ○ r
- ○ w
- ◉ r+
  Ez a válasz helyes, de nem jelölted meg.
- ○ w+
- ○ wx
- ○ wx+

Magyarázat

https://nodejs.org/api/fs.html#file-system-flags

2. feladat 0/4 pont

Az alábbi kód egy base64-ben kódolt szöveget tartalmazó fájlt olvas be. Melyik megoldást vagy megoldásokat helyettesítsük be a **???** helyére, hogy a console.log a dekódolt szöveget adja vissza?

sample.txt (utf-8):

```
TG9yZW0gaXBzdW0gZG9sb3Igc2l0IGFtZXQsIGNvbnNlY3RldHVyIGFkaXBpc2NpbmcgZWxpdC4=
```

index.mjs:

```
import { promises as fs } from 'fs';

const content = await fs.readFile('sample.txt', 'utf8');

console.log(???);
```

A futás eredménye:

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
```

Válaszok

- [ ] `content.toString('base64', 'utf8')`
- [x] `atob(content)`

  Ez a válasz helyes, de nem jelölted meg.
- [x] `Buffer.from(content, 'base64').toString()`

  Ez a válasz helyes, de nem jelölted meg.
- [ ] `String(content, { from: 'base64', to: 'utf8' })`
- [ ] `content`

Magyarázat

1. válasz: A Buffer.toString paraméterei sorban a következők: (encoding, start, end), tehát a példában megadott második paraméter nem fogja a base64 szövegünket dekódolni.
2. válasz: Deprecated megoldás, a Node nem ajánlja a használatát, viszont a példában meghatározott stringre működik.
3. válasz: Adott utf-8 string base64 dekódolásra talán ez az egyik legegyszerűbb megoldás.
4. válasz: A String függvénynek megadott második paraméter semmit nem csinál.

5. válasz: Egyszerűen csak kiírja a fájl tartalmát, mintha nem base64 kódolt szöveget tartalmazna.

3. feladat 0/6 pont

Mely állítások igazak az alábbiak közül?

Válaszok

- [x] Duplex streamek esetében a streamet olvasni és írni is lehet, és ezeknek nincs közvetlen hatásuk egymásra.
  Ez a válasz helyes, de nem jelölted meg.
- [ ] A NodeJS beépített stream library-jét a Netflix készítette.
- [ ] A stream által használható buffer méretét egy úgynevezett highWatermark property határozza meg és amíg a buffer mérete ez alá az érték alá nem csökken, addig nem tudjuk folytatni az írást.
- [x] A streamek segítségével memóriahatékonyan tudunk nagy fájlokat olvasni és írni.
  Ez a válasz helyes, de nem jelölted meg.
- [ ] Backpressure-ről akkor beszélünk, amikor a writeable stream gyorsabban dolgozza fel a bejövő adatot, mint ahogy azt a readable streamtől kapjuk.
- [ ] A Transform stream csak egy alias a duplex streamre, működésük ugyanaz.
- [x] Átlépve a highWatermark által meghatározott értéket, a writeable stream .write() metódusa már nem true, hanem false értéket ad vissza.
  Ez a válasz helyes, de nem jelölted meg.

Magyarázat

A NodeJS beépített stream library-jét a Netflix készítette: A Netflix már csak használta azt.

A streamek segítségével memóriahatékonyan tudunk nagy fájlokat olvasni és írni: A memóriahatákonyságot az adja, hogy az olvasás illetve írás egy meghatározott méretű bufferen keresztül történik, ami a fájlunknak mindig csak egy kis darabját tartalmazza, nem pedig az egészét.

A *highWatermark* nem egy limit, hanem küszöbérték. Az írás folytatható a küszöb átlépése után is, viszont a writable.write() ilyenkor már nem true-val, hanem false-szal tér vissza.

Backpressure-ről akkor beszélünk, amikor a writeable stream gyorsabban dolgozza fel a bejövő adatot, mint ahogy azt a readable streamtől kapjuk: A Backpressure esetében pont az ellenkező állítás igaz. Gyorsabban küldjük az adatot a Writeable streamnek, mint ahogy az fel tudná dolgozni azt. Ennek hatására az adat feltorlódik, ami pl. memória gondokhoz vezethet.

A Transform stream csak egy alias a duplex streamre, működésük ugyanaz: A két típus eltérő működésű, holott a transform stream is egy duplex.