BKSZC Pogány Frigyes Technikum

Rendezvénykereső

szoftverfejlesztő és -tesztelő
vizsgaremek

2022. május

Készítette:
Czégel Vanessza
Bassa Kristóf
Durmancs-Tóth Bendegúz

Konzulens:
Tóth József

Az alkalmazás rövid bemutatása

Az Eseménykereső egy olyan alkalmazás, amely megjeleníti az adatbázisába feltöltött, változatos eseményeket a kezdőlapján, a felhasználó pedig szabadon böngészhet köztük az általa preferált szűrési lehetőségekkel. A programok és egyéb szabadidős tevékenységek egy letisztult, átlátható és könnyen kezelhető felületen jelennek meg, az eseményekre kattintva pedig részletes leírást kaphatunk róluk. Az adatbázis bővíthető az arra jogosult személyek által, így a legfrissebb nyílt bulik és családi programok mindig csak pár kattintásra lesznek a felhasználótól.

Az alkalmazásunk célja egy kényelmes felület megvalósítása, ahol a felhasználók könnyen tudnak keresgélni szórakozási lehetőségek között, regisztráció nélkül. Találkoztunk már jól működő külföldi eseményeket megjelenítő oldalakkal, de a hazai piacon elavult rendszerekkel és megjelenésekkel futottunk össze. A Facebook közösségi platformja ugyan ideális, viszont laza szabályozások kötik, hivatalos rendezvények mellett különösebb engedély nélkül szervezhet bármelyik felhasználó újabb eseményt, olyan témában, amiben szeretne. Ezt a gondolatot használtuk fel, majd arra a következtetésre jutottunk, hogy korlátozni szeretnénk a megosztási lehetőségeket, így csak hivatalos szervezéseket engednénk ki, amiket az arra jogosult adminisztrátorok tudnak feltölteni az adatbázisba. Ennek előnyei főként a kevesebb adatforgalom, az ellenőrizhetőség, a hitelesség, és így elkerülhető a társadalom számára káros céllal létrejött szerveződések lehetősége is. A szervezőknek pedig egy olyan platformot nyújtunk, ahol egyszerűen tudják feltölteni és megosztani az újabb eseményeket az adminisztrátorok segítségével, akik külön bejelentkezési lehetőséggel rendelkeznek.

A repository elérése:
https://github.com/Aszellem/blackbox/tree/dev

Projekttervezési feltételek, szoftveres eszközök

Fejlesztéshez használt alkalmazások:

- Visual Studio Code (a kód megírása)
- Postman (a kód tesztelése)
- phpMyAdmin (a szerver létrehozása)
- XAMPP (a szerver futtatása)
- Google Chrome / DevTools (egyéb tesztelések)
- Discord (kommunikáció a csapattagok között)
- GitHub (verziókezelés)

Az adatbázis bemutatása

Szerkezet és táblázatok ismertetése

1. Az ***eloado*** tábla adatai:
   - eloado_id: Az adatbázisba feltöltött előadó, művész, együttes egyedi azonosítója, ami elsődleges kulcsként funkcionál
   - eloado_nev: Az adatbázisba feltöltött előadó, művész, együttes neve
   - bio: Az adatbázisba feltöltött előadó, művész, együttes bemutatása, leírása

#	Név	Típus	Illesztés	Tulajdonságok	Nulla	Alapértelmezett	Megjegyzések	Extra
1	eloado_id	int(11)			Nem	*Nincs*		AUTO_INCREMENT
2	eloado_nev	varchar(50)	utf8mb4_hungarian_ci		Nem	*Nincs*		
3	bio	text	utf8mb4_hungarian_ci		Nem	*Nincs*		

2. A ***helyszin*** tábla adatai:
   - helyszin_id: Az adatbázisba feltöltött helyszín egyedi azonosítója, elsődleges kulcsként funkcionál
   - helyszin_nev: Az adatbázisba feltöltött helyszín neve
   - helyszin_cim: Az helyszín címe, ahol események kerülnek megrendezésre
   - helyszin_bemutatas: Az helyszín rövid leírása, bemutatása
   - email: A helyszín e-mail címe, amennyiben rendelkezik ilyen elérhetőséggel
   - weblap: A helyszín weboldala, amennyiben rendelkezik ilyen lehetőséggel

#	Név	Típus	Illesztés	Tulajdonságok	Nulla	Alapértelmezett	Megjegyzések	Extra
1	helyszin_id	int(3)			Nem	*Nincs*		AUTO_INCREMENT
2	helyszin_nev	varchar(100)	utf8mb4_hungarian_ci		Nem	*Nincs*		
3	helyszin_cim	varchar(255)	utf8mb4_hungarian_ci		Nem	*Nincs*		
4	helyszin_bemutatas	text	utf8mb4_hungarian_ci		Nem	*Nincs*		
5	email	varchar(255)	utf8mb4_hungarian_ci		Nem	*Nincs*		
6	weblap	varchar(255)	utf8mb4_hungarian_ci		Nem	*Nincs*		

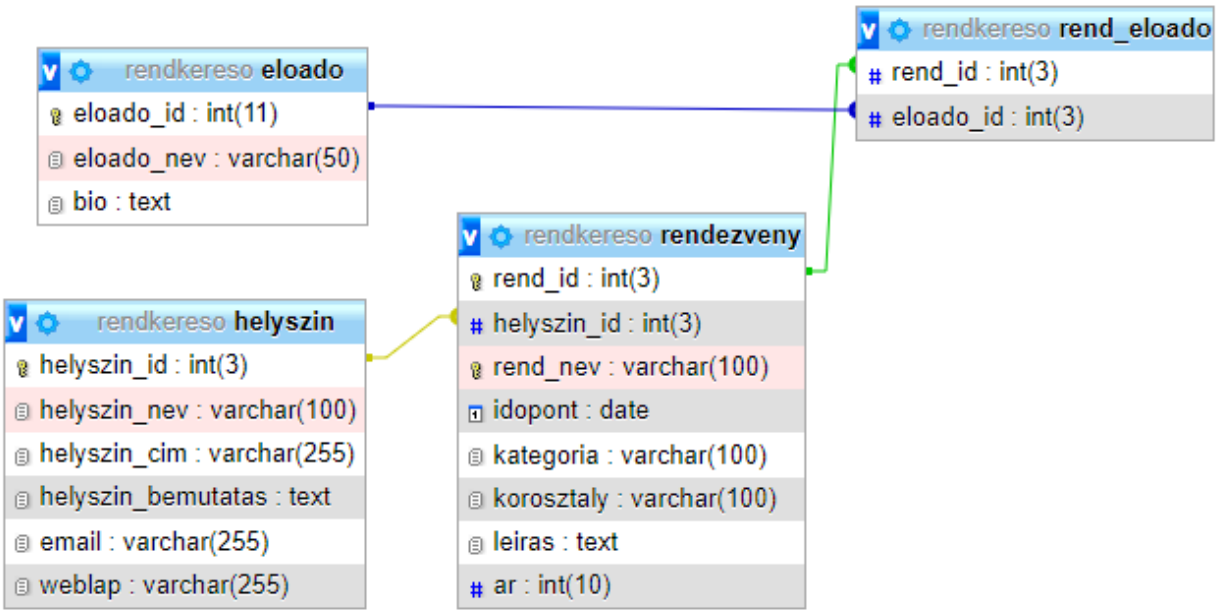
3. A ***rendezveny*** tábla adatai:
   - rend_id: Az adatbázisba feltöltött rendezvény egyedi azonosítója, elsődleges kulcsként funkcionál
   - helyszin_id: Egy korábban felvitt helyszín egyedi azonosítója, idegen kulcsként használva ebben a táblában
   - rend_nev: Az adatbázisba feltöltött rendezvény neve
   - idopont: A rendezvény időpontja
   - kategoria: A rendezvény kategóriája
   - korosztaly: A rendezvény célközönségének a korosztálya
   - leiras: A rendezvény bemutatása, leírása
   - ar: A rendezvény belépési díja

#	Név	Típus	Illesztés	Tulajdonságok	Nulla	Alapértelmezett	Megjegyzések	Extra
1	rend_id	int(3)			Nem	*Nincs*		AUTO_INCREMENT
2	helyszin_id	int(3)			Nem	*Nincs*		
3	rend_nev	varchar(100)	utf8mb4_hungarian_ci		Nem	*Nincs*		
4	idopont	date			Nem	*Nincs*		
5	kategoria	varchar(100)	utf8mb4_hungarian_ci		Nem	*Nincs*		
6	korosztaly	varchar(100)	utf8mb4_hungarian_ci		Nem	*Nincs*		
7	leiras	text	utf8mb4_hungarian_ci		Nem	*Nincs*		
8	ar	int(10)			Nem	*Nincs*		

4. A ***rend_eloado*** tábla adatai:
   - rend_id: Egy korábban felvitt rendezvény egyedi azonosítója, idegen kulcsként használva ebben a táblában
   - eloado_id: Egy korábban felvitt előadó egyedi azonosítója, idegen kulcsként használva ebben a táblában

#	Név	Típus	Illesztés	Tulajdonságok	Nulla	Alapértelmezett	Megjegyzések	Extra
1	rend_id	int(3)			Nem	*Nincs*		
2	eloado_id	int(3)			Nem	*Nincs*		

5. Az adatbázis szerkezeti ábrája:



Komponensek technikai leírása, használatuk és forráskódjaik

Az app.js funkciója, használata

Ez a szerver oldali JavaScript fájl felelős a modulok betöltéséért, az útvonalak meghatározásáért, és a különböző lekérési és feltöltési feladatok elvégzéséért, egy API segítségével.
A kód elején betöltjük a szükséges modulokat:

- express
- mysql
- cors
- dotenv
- bcrypt
- jsonwebtoken

```
const express = require("express");
const app = express();
app.use(express.json());
const mysql = require("mysql");
const cors = require("cors");
app.use(cors());
require("dotenv").config();

const bcrypt = require("bcrypt");
const jwt = require("jsonwebtoken");
```

Ezt követően létrehozunk egy pool-t, hogy a lokális MySQL szerverhez hozzáférjünk a következő utasításokkal:

```
const pool = mysql.createPool({
    host: "localhost",
    port: "3306",
    user: "root",
    password: "",
    database: "rendkereso",
});
```

A felhasználónév 'root', a jelszó pedig nincs meghatározva, így üresen hagyjuk. Az adatbázis neve 'rendkereso', innen fogjuk az adatokat lekérdezni, illetve feltölteni őket.

Az alábbi végpont az adminisztrátori bejelentkezéshez szükséges, ide küldjük el a jelszót (kicsinagy), pontosabban a 'hash' kódját, amelyet programmal előállítottunk, és elhelyeztünk a **.env** nevű fájlban.
A JsonWebToken előállításához szükséges titkos kulcsot is legeneráltuk, ami szintén beírásra került.

```
ADMIN=$2b$10$2F8QUaH3MKqI2ZAzHlz3YuKXAM/KtJ.EGqEIeRIiT6YTQKj8AZdwu
TOKEN_SECRET=0982cedee01235bd81e1706d666b58244c6ba5adaa455220663f8a122d33129f
0d843a5e2788acdb5a7b6e99ec82615639e93ccf58296bb5c5714ed5c3d6ad5d
```

```
app
    .route("/admin")
    .post(function(req, res) {
        const hash = process.env.ADMIN;
        if (!bcrypt.compareSync(req.body.password, hash))
            return res.status(401).send({ message: "Hibás jelszó!" });
        const token = jwt.sign({ password: req.body.password },
            process.env.TOKEN_SECRET, { expiresIn: 3600 }
        );
        res.json({ token: token, message: "Sikeres bejelentkezés." });
    })
```

A token ellenőrzése middleware-el:

```
function authenticateToken(req, res, next) {
    const authHeader = req.headers["authorization"];
    const token = authHeader && authHeader.split(" ")[1];
    if (!token) return res.status(401).send({ message: "Azonosítás szükséges"
});
    jwt.verify(token, process.env.TOKEN_SECRET, (err, user) => {
        if (err) return res.status(403).send({ message: "Nincs jogosultság!"
});
        req.user = user;
        next();
    });
}
```

Először ellenőrizzük az 'authorization' fejlécet és elválasztjuk a szóköznél, majd a második felét használjuk fel. Ha nem küldték a fejlécet vagy abban a tokent, akkor 401-es kódot és egy „Azonosítás szükséges" hibaüzenetet küldünk. Ha megkaptuk a tokent, akkor a titkos kulcs segítségével ellenőrizzük. Ha az ellenőrzés során hiba történik, akkor 403-as hibakódot küldünk. Ha minden sikeres, akkor az abban tárolt felhasználói adatokat hozzáadjuk a kéréshez, és meghívjuk a következő függvényt.

A „**/kezdolap**” útvonalon a főoldalon megjelenő események adatait kérdezzük le (’GET’ metódus) az API segítségével, és időpont szerint csökkenő sorrendbe rendezve őket.

```
app.route("/kezdolap").get(function(req, res) {
    const q =
        "SELECT r.rend_id AS id, r.rend_nev AS 'rend_nev'," +
        " date_format(r.idopont, '%Y %M %d') AS 'idopont'," +
        " h.helyszin_nev AS 'helyszin_nev', e.eloado_id AS eEloadoID," +
        " re.eloado_id AS reEloadoID, re.rend_id AS reRendID," +
        " e.eloado_nev AS 'eloado_nev'" +
        " FROM helyszin AS h INNER JOIN" +
        " rendezveny AS r" +
        " ON h.helyszin_id = r.helyszin_id INNER JOIN" +
        " rend_eloado AS re" +
        " ON r.rend_id = re.rend_id INNER JOIN" +
        " eloado AS e" +
        " ON re.eloado_id = e.eloado_id GROUP BY r.rend_nev" +
        " ORDER BY r.idopont DESC;";
    pool.query(q, function(error, results) {
        if (!error) {
            res.send(results);
        } else {
            res.send(error);
        }
    });
});
```

A „**/bovebben/:id**” útvonalon is egy ’GET’ metódus valósul meg. Ez a rendezvények kártyáiban található „Részletes infók” gombra kattintva, az adott esemény bővebb információit kérdezi le a szerverről.

```
app.route("/bovebben/:id").get(function(req, res) {
    const q =
        "SELECT r.rend_id, r.rend_nev," +
        " date_format(r.idopont, '%Y %M %d') AS 'idopont', r.kategoria," +
        " r.korosztaly, r.leiras, r.ar, h.helyszin_nev, h.email, h.weblap," +
        " e.eloado_nev " +
        " FROM helyszin AS h JOIN" +
        " rendezveny AS r" +
        " ON h.helyszin_id = r.helyszin_id JOIN" +
        " rend_eloado AS re" +
        " ON r.rend_id = re.rend_id JOIN" +
        " eloado AS e" +
        " ON re.eloado_id = e.eloado_id" +
        " WHERE r.rend_id = ?";
    pool.query(q, [req.params.id], function(error, results) {
        if (!error) {
            res.send(results);
        } else {
```

A „**/nevSzerint/**” útvonalon egy olyan ’get function’ fut le, ami a kezdőlapon név szerinti sorrendbe rendezi az eseményeket. A lekérdezés hasonló, mint a fentieknél, ezért az eltérő sorokat sárga nyíllal jelöljük.

```
        " ON re.eloado_id = e.eloado_id" +
        " GROUP BY rend_nev ORDER BY r.rend_nev;"; ←
    pool.query(q, function(error, results) {
```

Az adatbázisba négy űrlap segítségével töltünk fel adatokat, mindegyikhez külön útvonal tartozik. Az űrlapok sorrendje fontos szempont volt, ugyanis az utolsó kettő felhasználhatja az előző kettőben megadott információkat, hogy megkönnyítse az események létrehozását. Ezt a lenti ’POST’ metódust alkalmazó kódrészletek mutatják:

```
app.route("/submitHelyszin")
    .post(function(req, res) {
        const q = "INSERT INTO helyszin (helyszin_nev," +
        " helyszin_cim, helyszin_bemutatas, email, weblap)" +
        " VALUES (?, ?, ?, ?, ?);";
        pool.query(q, [req.body.helyszin_nev, req.body.helyszin_cim,
req.body.helyszin_bemutatas, req.body.email, req.body.weblap],
            function(error, result) {
                if (!error) {
                    res.send(result);
                } else {
                    res.send(error);
                }
            }
        )
    });
```

```
app.route("/submitEloado") ←
    .post(function(req, res) {
        const q = "INSERT INTO eloado (eloado_nev, bio) VALUES (?, ?);"; ←
        pool.query(q, [req.body.eloado_nev, req.body.bio], ←
```

```
app.route("/submitRendezveny") ←
    .post(function(req, res) {
        const q = "INSERT INTO rendezveny (helyszin_id," + ←
        " rend_nev, idopont, kategoria, korosztaly, leiras, ar)" +
        " VALUES (?, ?, ?, ?, ?, ?, ?);";
        pool.query(q,
            [req.body.helyszin_id, req.body.rend_nev, ←
                req.body.idopont, req.body.kategoria,
                req.body.korosztaly, req.body.leiras, req.body.ar],
```

```
app.route("/submitEsemeny") ←
    .post(function(req, res) {
        const r = "INSERT INTO rend_eloado (rend_id, eloado_id) ←
      VALUES (?, ?);";
        pool.query(r, [req.body.rend_id, req.body.eloado_id], ←
```

A „**/submitHelyszin**” az adatbázis ***helyszin*** táblájába küldi az adatokat. Az ***eloado*** és ***rendezveny*** táblák a velük megegyező nevű végpontokkal vannak összeköttetésben, a „**/submitEsemeny**” útvonallal pedig a ***rend_eloado*** tábla.

Az alábbi betöltési útvonalak szintén az űrlapoknál használatosak, mert legördülő sávokba helyezzük a már létrehozott előadók, helyszínek és rendezvények listáját, a komfortosabb kitöltés érdekében.

```
app.route("/rendezvenyek")
    .get(function(req, res) {
        const q = "SELECT * FROM rendezveny AS r";
        pool.query(q, function(error, results) {
            if (!error) {
                res.send(results);
            } else {
                res.send(error);
            }
        })
    })
```

```
app.route("/eloadok")
    .get(function(req, res) {
        const q = "SELECT * FROM eloado as e";
```

```
app.route("/helyszinek")
    .get(function(req, res) {
        const q = "SELECT * FROM helyszin";
```

Az index.js funkciója, használata

Ez a JavaScript fájl tartalmazza a bejelentkezéshez szükséges kódolást, a kijelentkezést, a szűréseket, illetve itt kérjük le az útvonalakon keresztül a 'json' formátumba átalakított adatokat a szerverről, ahol a lekérdezésekhez a 'fetch promise'-t használjuk.
Az alábbi ábra a bejelentkezést hajtja végre. A beírt jelszó értékét ellenőrzi, és ha helyesen adtuk meg, a legördülő menüsorban új opciók válnak elérhetővé.

```
document.getElementById("login").onclick = function() {
    const url = 'http://localhost:3000/admin';
    fetch(url, {
            method: 'POST',
            headers: {
                'Content-type': 'application/json;charset=utf-8'
            },
            body: JSON.stringify({
                "password": document.getElementById("password").value,
            })
        })
        .then(res => {
            ok = res.ok
            return res.json()
        })
        .then(json => {
            document.getElementById("uzenet2").innerHTML = json.message
            if (ok) {
                sessionStorage.token = json.token
                document.location = "menu-bar.html"
            }
        })
        .catch(err => console.log(err));
}
```

A **kijelentkezes()** törli a token érvényességét az index.html oldalon, így megszüntetve az adminisztrátori hozzáférést.

```
function kijelentkezes() {
    delete sessionStorage.token
    document.location = "index.html"
}
```

A **kezdolap()** a saját végpontján keresztül lekéri és 'json' formátumba alakítja az adatokat a szerverről, majd a 'forEach' ciklus segítségével minden eseményt kiír a főoldalra, kártyákba szedve.

```
function kezdolap() {
    const url = 'http://localhost:3000/kezdolap/';
    const lista = document.getElementById("rendezvenyek");
    fetch(url)
        .then((response) => response.json())
        .then(json => {
            lista.innerHTML = "";
            json.forEach(f => {
                id = f.id;
                lista.innerHTML +=
                    "<div class='card h-150 col-lg-3' id='" + id + "'>" +
                    "<p id='rendNev'>" + f.rend_nev + "</p>" +
                    "<p>" + f.idopont + "</p>" +
                    "<p>Helyszín:<br><br>" + f.helyszin_nev + "</p>" +
                    "<a href='bovebben_login.html?id=" + id + "'>" +
                    "<button type='button' class='button' id='button" + id
                    + "' onClick='bovebben(" + id + ")'" +
                    " style='vertical-align:middle'>Részletes infók</button>" +
                    "</a>" +
                    "</div>";
            })
        })
        .catch(err => console.log(err));
}
```

A keresésért felelős kód a rendezvények és előadók nevét használja forrásnak úgy, hogy ha a keresőmezőbe beírt szöveg megtalálható ezek közül valamelyikben, akkor csak ezeket az egyezéseket jeleníti meg az oldal.

```
function kereses() {
    const url = 'http://localhost:3000/kezdolap';
    const lista = document.getElementById("rendezvenyek");
    let val = document.getElementById('kereso').value;
    let kisval = val.toLowerCase();

    fetch(url)
        .then((response) => response.json())
        .then(json => {
            lista.innerHTML = "";
            json.forEach(f => {
                let eloado = f.eloado_nev;
                kiseloado = eloado.toLowerCase();
                let rend = f.rend_nev;
                kisrend = rend.toLowerCase();
                if (kisval.length > 0) {
                    if (kisrend.includes(kisval) || kiseloado.includes(kisval))
{
                        lista.innerHTML +=
                            "<div class='card h-150 col-lg-3' id='" + id + "'>"
+
                            "<p id='rendNev'>" + f.rend_nev + "</p>" +
                            "<p>" + f.idopont + "</p>" +
                            "<p>Házigazda: " + f.eloado_nev + "</p>" +
                            "<p>" + f.helyszin_nev + "</p>" +
                            "<a href='bovebben.html?id=" + id + "'>" +
                            "<button type='button' class='button' id='button" +
                            id + "' onClick='bovebben(" + id + ")' " +
                            "style='vertical-align:middle'>Részletes infók" +
                            "</button>" +
                            "</a>" +
                            "</div>";
                    }
```

Amennyiben nem konkrét tartalomra szeretnénk keresni, választhatunk az előre megírt gombok alapján. A **nevSzerint()** függvény ábécé sorrendbe rakja a kártyákat a rendezvények neve alapján, az időrendi sorrendért pedig a 'Rendezés dátum szerint' gombra kell kattintani.

```
function nevSzerint() {
    const url = 'http://localhost:3000/nevSzerint/';
    const lista = document.getElementById("rendezvenyek");
    fetch(url)
```

A menu-bar.js funkciói és működése

Ez a fájl tartalmazza azokat a scripteket, amik az űrlapok kitöltését szolgáljak. Itt valósul meg a legtöbb 'submit' és 'POST' metódus, küldjük el az információkat az Eseménykereső adatbázisába, és töltjük be a legördülő sávokba a már meglévő adatokat a kitöltés segítéséhez. Néhány funkció azonos az előző JavaScript fájlban látottakhoz, ezért az ábrákon a létrehozáshoz és betöltéshez szükséges kódokat mutatjuk be.
A **submitHelyszin()** a nevének megegyező űrlap tartalmát tölti fel a szintén azonos nevű útvonalon keresztül. A mezők értékét veszi, majd a 'fetch' 'POST' metódussal a megfelelő 'json' formátumba küldi el az adatokat a szerver felé. Sárga nyilakkal mutatjuk a fontosabb sorokat.

```
function submitHelyszin() {
    //console.log("submitHelyszin elindult...");
    const helyszin = document.getElementById("helyszin").value;
    const cim = document.getElementById("cim").value;
    const leiras = document.getElementById("leiras").value;
    const email = document.getElementById("email").value;
    const weblap = document.getElementById("weblap").value;
    const url = 'http://localhost:3000/submitHelyszin'; ←
    fetch(url, {
            method: 'POST', ←
            headers: {
                'Content-type': 'application/json;charset=utf-8'
            },
            body: JSON.stringify({
                "helyszin_nev": helyszin, ←
                "helyszin_cim": cim, ←
                "helyszin_bemutatas": leiras, ←
                "email": email, ←
                "weblap": weblap ←
            })
        })
        .then(json => console.log(json))
        .catch(err => console.log(err));
};
```

A **submitEloado()** ugyanúgy működik mint a feljebbi kódrészlet, ezért csak azokat a sorokat ábrázoljuk, ahol eltérés van.

```
function submitEloado() {
    const eloado = document.getElementById("eloado").value;
    const bio = document.getElementById("bio").value;
    const url = 'http://localhost:3000/submitEloado';
    if (document.getElementById("eloado") != "") {
        fetch(url, { …
                …},
                body: JSON.stringify({
                    "eloado_nev": eloado,
                    "bio": bio,
```

A rendezvény feltöltéséhez a már említett legördülő sávokat is alkalmazzuk. A helyszín megadásához egy szűrő is segédkezik 'helyszinSzur' néven. Ezek a mezők úgy kapnak értéket, hogy a listában megjelenő helyszíneket a **helyszinBetolt()** funkció betölti rádió típusú inputokba a „**/helyszinek**" végponton keresztül, majd ezeken egy kód 'for' ciklussal addig megy, amíg a kijelölt opcióba nem ütközik. Itt a 'helyszinValue' változó megkapja a helyszín egyedi azonosítójának (id) értékét, és az adatbázis ***rendezveny*** táblája idegen kulcsként használva ezt, hozzáfér a helyszín adataihoz.

```
function submitRendezveny() {
    const rendezveny = document.getElementById("rendezveny").value;
    const datum = document.getElementById("datum").value;
    const kategoria = document.getElementById("kategoria").value;
    const korosztaly = document.getElementById("korosztaly").value;
    const es_leiras = document.getElementById("es_leiras").value;
    const ar = document.getElementById("ar").value;
    const url = 'http://localhost:3000/submitRendezveny';

    var helyszinValue = 0;
    var inputHelyszin = document.getElementsByClassName('helyszinRadio');
    for (var i = 0; inputHelyszin[i]; ++i) {
        if (inputHelyszin[i].checked) {
            helyszinValue = inputHelyszin[i].value;
            break;
        }
    }
    fetch(url, {
            method: 'POST',
            headers: {
                'Content-type': 'application/json;charset=utf-8'
            },
            body: JSON.stringify({
                "helyszin_id": helyszinValue,
                "rend_nev": rendezveny,
                "idopont": datum,
                "kategoria": kategoria,
                "korosztaly": korosztaly,
                "leiras": es_leiras,
                "ar": ar,
            })
        })
        .then(json => console.log(json))
        .catch(err => console.log(err));
};
```

```
function helyszinBetolt() {
    const url = 'http://localhost:3000/helyszinek';
    const helyszinek = document.getElementById("helyszinLista");
    fetch(url)
        .then((response) => response.json())
        .then(json => {
            helyszinek.innerHTML = "";
            json.forEach(f => {
                id = f.helyszin_id;
                helyszinek.innerHTML +=
                    "<input type='radio' name='helyszinValaszto'
class='helyszinRadio' value='" + id + "' id='" + id + "'><label style='font-
size:16'>" + f.helyszin_nev + "</label><br>";
            })
        })
        .catch(err => console.log(err));
}

function helyszinSzur() {
    const url = 'http://localhost:3000/helyszinek';
    const lista = document.getElementById("helyszinLista");
    let val = document.getElementById('helyszinSzuro').value;
    let kisval = val.toLowerCase();

    fetch(url)
        .then((response) => response.json())
        .then(json => {
            lista.innerHTML = "";
            json.forEach(f => {
                let helyszin = f.helyszin_nev;
                kishelyszin = helyszin.toLowerCase();
                if (kisval.length >= 0) {
                    if (kishelyszin.includes(kisval)) {
                        id = f.helyszin_id;
                        lista.innerHTML +=
                            "<input type='radio' class='helyszinRadio' value='"
+ id + "' id='" + id + "'><label style='font-size:16'>" + f.helyszin_nev +
"</label><br>";
                    }
                }
            })
        })
        .catch(err => console.log(err));
}
```

A **submitEsemeny()** ugyanezen az elven működik, itt mind a két adat, amit megadhatunk, kiválasztható egy listából, és összeköti a rendezvényeket az előadókkal. Korábban említettük a sorrend fontosságát, ugyanis így, egy teljesen új esemény, új fellépőkkel és helyszínekkel megadhatóvá válik zökkenőmentesen.

```
function submitEsemeny() {
    console.log("submitEsemeny elindult...");
    const url = 'http://localhost:3000/submitEsemeny';

    var rendValue = 0;
    var inputRendezveny = document.getElementsByClassName('rendRadio');
    for (var i = 0; inputRendezveny[i]; ++i) {
        if (inputRendezveny[i].checked) {
            rendValue = inputRendezveny[i].value;
            break;
        }
    }

    var eloadoValue = 0;
    var inputEloado = document.getElementsByClassName('eloadoCheckbox');
    for (var i = 0; inputEloado[i]; ++i) {
        if (inputEloado[i].checked) {
            eloadoValue = inputEloado[i].value;
            break;
        }
    }
    fetch(url, {
            method: 'POST',
            headers: {
                'Content-type': 'application/json;charset=utf-8'
            },
            body: JSON.stringify({
                "rend_id": rendValue,
                "eloado_id": eloadoValue
            })
        })
        .then(json => console.log(json))
        .catch(err => console.log(err));
}
```

A **rendezvenyBetolt()** és **rendezvenySzur()** funkciók kódrészletei, sárga nyíllal jelölve a fontosabb elemeket.

```
function rendezvenyBetolt() {
    const url = 'http://localhost:3000/rendezvenyek'; ←
    const rendezvenyek = document.getElementById("rendezvenyLista");
    fetch(url)
        .then((response) => response.json())
        .then(json => {
            rendDarab = json.length; ←
            rendezvenyek.innerHTML = "";
            json.forEach(r => {
                id = r.rend_id; ←
                rendezvenyek.innerHTML +=
                    "<input type='radio' class='rendRadio' value='" + id + "'
id='" + id + "'><label style='font-size:16'>" + r.rend_nev + "</label><br>";
            })
        })
        .catch(err => console.log(err));
}

function rendezvenySzur() {
    const url = 'http://localhost:3000/rendezvenyek'; ←
    const lista = document.getElementById("rendezvenyLista");
    let val = document.getElementById('rendezvenySzuro').value;
    let kisval = val.toLowerCase();

    fetch(url)
        .then((response) => response.json())
        .then(json => {
            lista.innerHTML = "";
            json.forEach(r => {
                let rendezvenyek = r.rend_nev; ←
                kisrendezvenyek = rendezvenyek.toLowerCase();
                if (kisval.length >= 0) {
                    if (kisrendezvenyek.includes(kisval)) {
                        id = r.rend_id; ←
                        lista.innerHTML +=
                            "<input type='radio' class='rendRadio' value='" +
id + "' id='" + id + "'><label style='font-size:16'>" + r.rend_nev +
"</label><br>";
                    }
                }
            })
        })
        .catch(err => console.log(err));
}
```

A javascript végén meghívjuk az oldal betöltődésekor szükséges függvényeket.

```
helyszinBetolt();
eloadokBetolt();
rendezvenyBetolt();
```

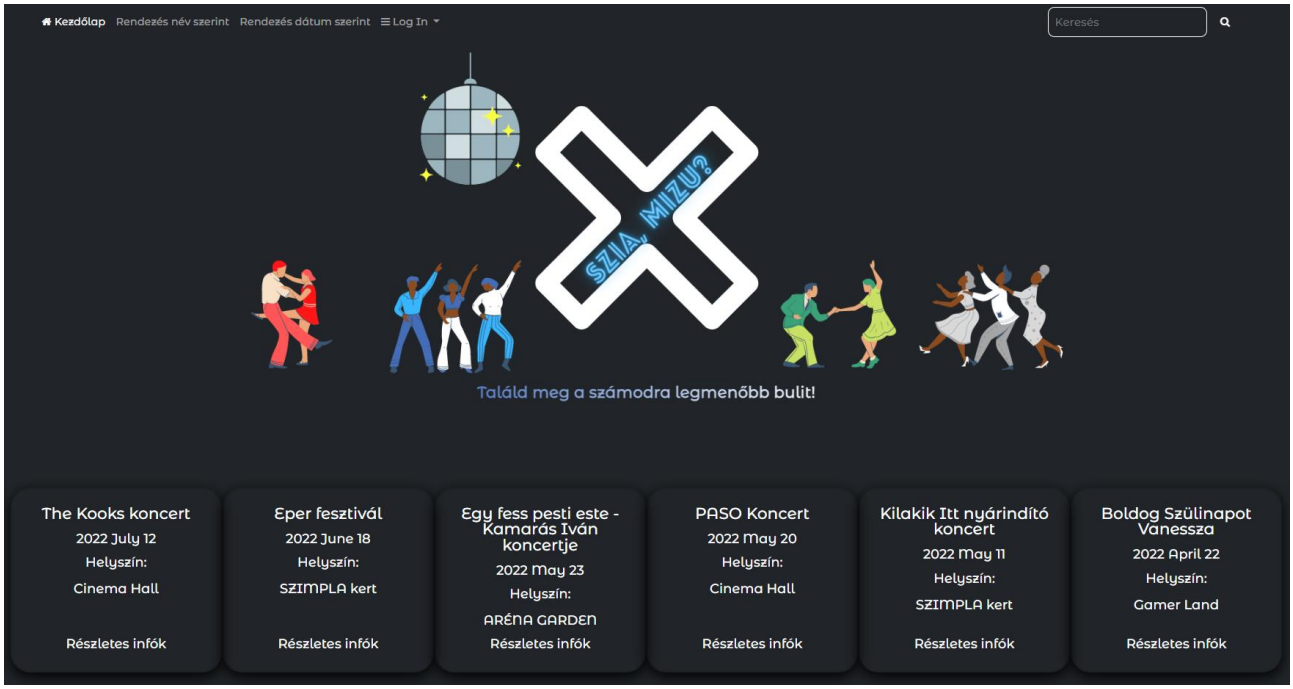
A bovebben.js és bovebben-login.js működése

Ezek a scriptek a kártyák részletes információit szolgáltató megjelenéseket biztosítják. Az események alján található gombra kattintva megtekinthetjük a további információkat, leírásokat a rendezvénnyel kapcsolatban, egy külön felületen megnyitva. A két fájl hasonló kódokkal rendelkezik, ezért a **bovebben.js** -t ábrázoljuk.

```
function bovebben() {
    window.onload = function(linkId) {
        try {
            let url_string = (window.location.href).toLowerCase();
            let url = new URL(url_string);
            linkId = url.searchParams.get("id");
            console.log(linkId);
        } catch (error) {
            console.log("Hiba az URL paraméter beolvasásakor - " + err);
        }
        console.log(linkId);
        const url = 'http://localhost:3000/bovebben/' + linkId;
        const lista = document.getElementById("reszletek");
        let fellepok = document.getElementById("fellepok");
        fetch(url)
            .then((response) => response.json())
            .then(json => {
                console.log(json[0]),
                    json.forEach(f => {
                        fellepok.innerHTML += f.eloado_nev + "<br><br>";
                    });
                lista.innerHTML = "";
                lista.innerHTML +=
                    "<p><h2>" + json[0].rend_nev + "</h2></p>" +
                    "<p>Időpont: " + json[0].idopont + "</p>" +
                    "<p>Helyszín: " + json[0].helyszin_nev + "</p>" +
                    "<p>Helyszín e-mail: " + json[0].email + "</p>" +
                    "<p>Helyszín weblap: <a href='" + json[0].weblap + "'>" +
json[0].weblap + "<a></p>" +
                    "<p>Kategória: " + json[0].kategoria + "</p>" +
                    "<p id='leiras'>Esemény leírása: " + json[0].leiras +
"</p>" +
                    "<p>Ár: " + json[0].ar + "</p>";
            })
```

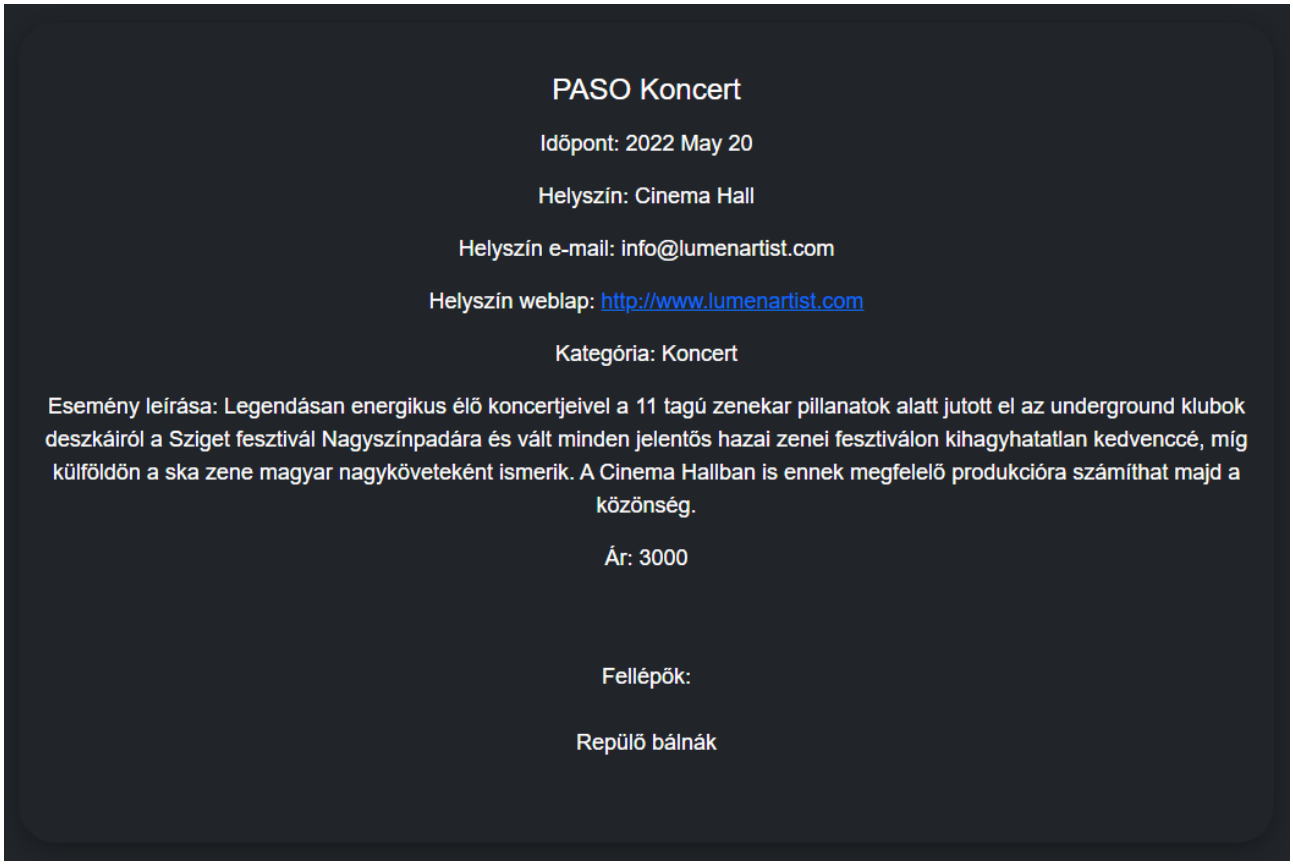
A weboldal kliens oldali megjelenése

A kezdőoldal megjelenése:

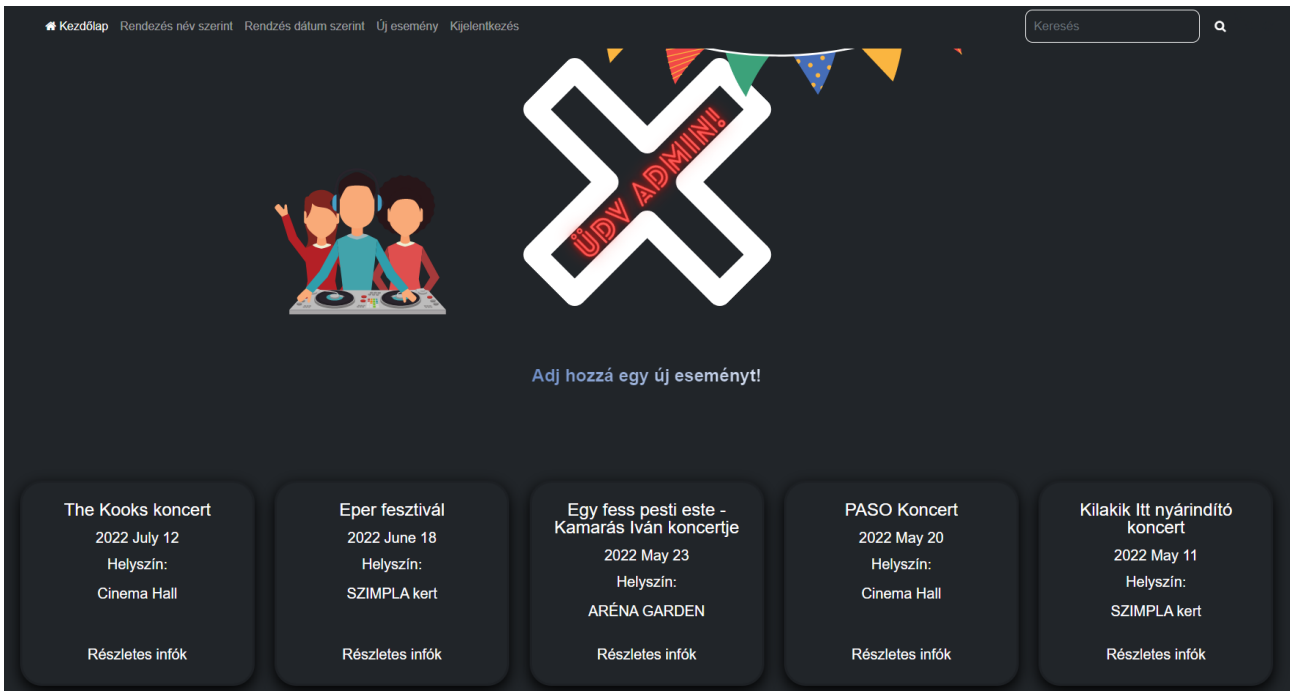


Egy esemény kártyája közelről, kurzor a részletes infók gombon:

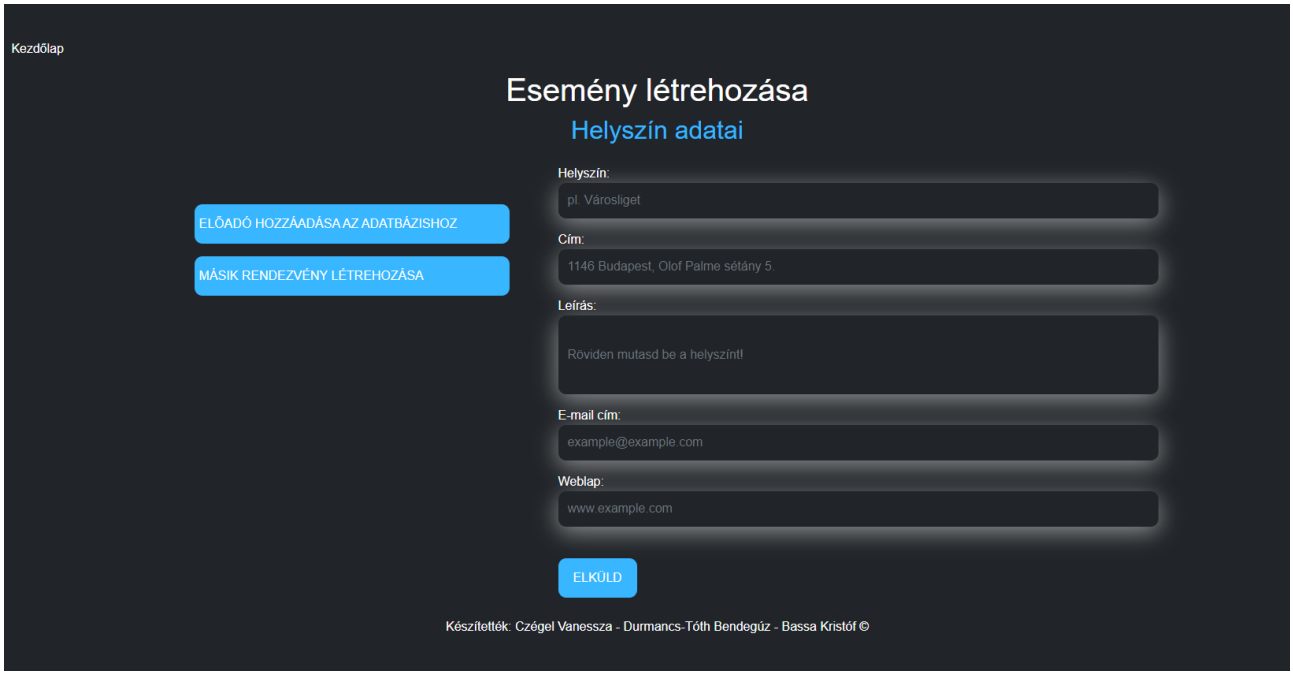
A következő ábra a részletes infókat mutatja. Itt található meg minden információ az adott rendezvényről. Az oldal jobb felső sarkában található gombbal a kezdőlapra térhetünk vissza:



Az adminisztrátori belépés ablaka közelről. A helyes jelszó beírása után, a bejelentkezésre kattintva új lehetőségek nyílnak meg az oldalon:



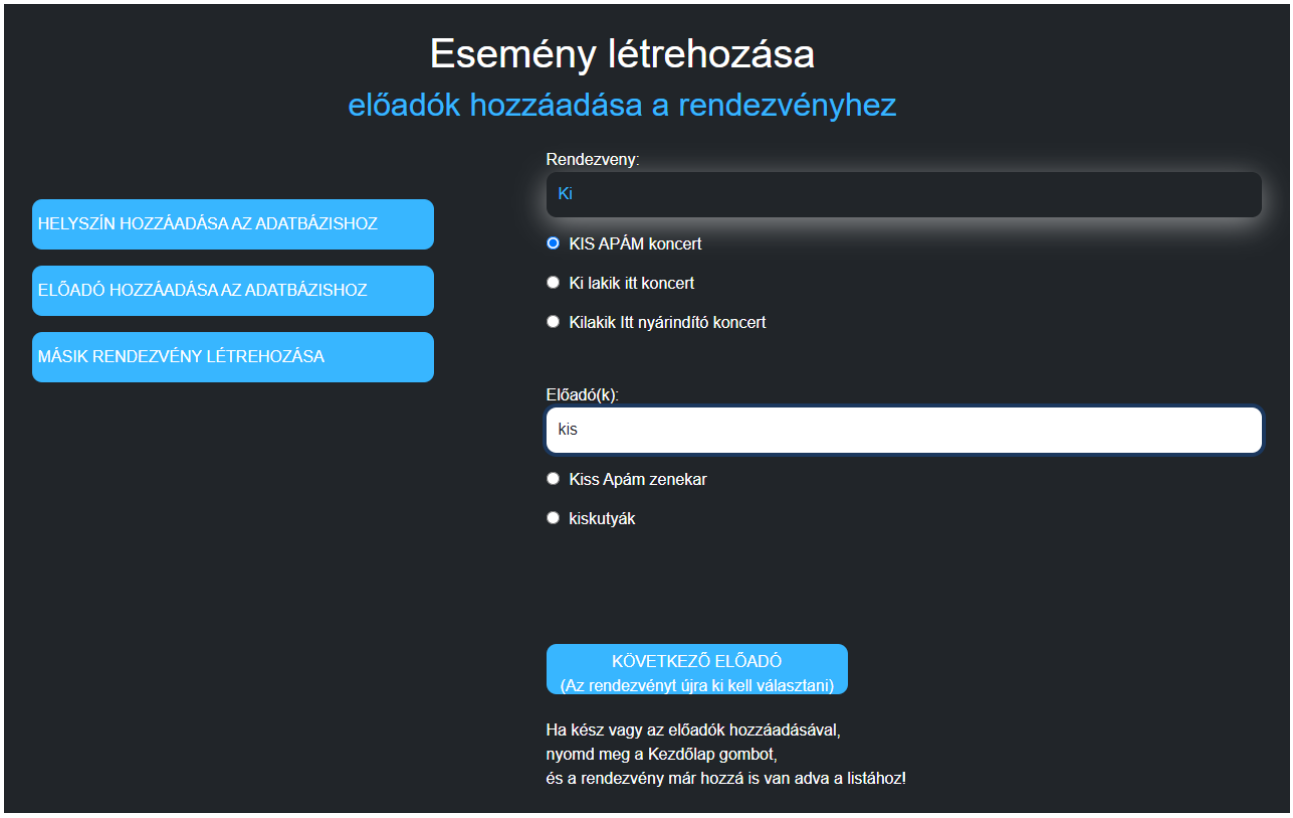
Belépés után az ’Új esemény’-re kattintva az űrlapok elérhetővé válnak, hogy újabb programokkal gazdagítsuk az adatbázist. Az űrlapok bal oldalán navigációs gombokkal lehet váltogatni melyiket szeretnénk kitölteni, de az ’Elküld’ gomb automatikusan a következő oldalra visz kitöltés után, illetve ez viszi fel az adatokat a megfelelő táblába.
A helyszín adatainak megadása:



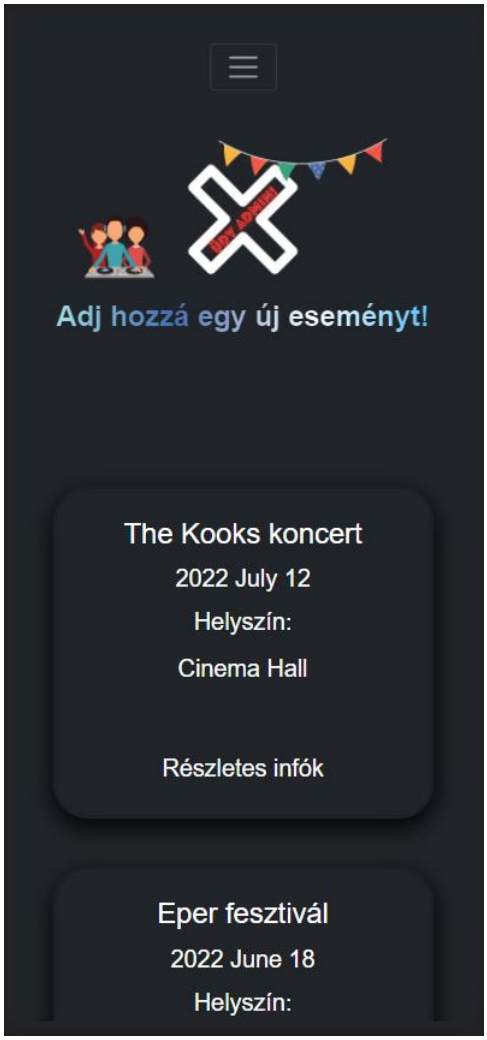
Az előadók hozzáadásánál a bal oldali gombbal elkerülhető a következő űrlap betöltése küldés után, ha több fellépőt szeretnénk hozzáadni az adatbázishoz. Az alsó gomb a fentiekkel megegyezően működik.
Előadók hozzáadása:

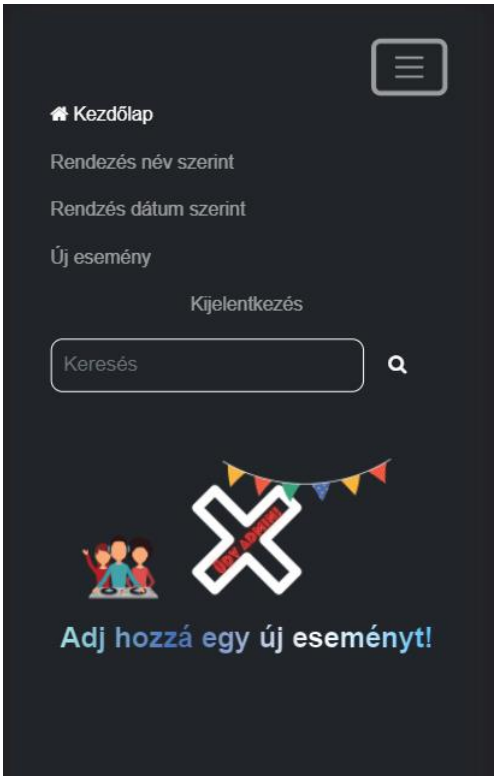
A harmadik űrlapon hozzuk létre a rendezvényt. A helyszíneket egy listába töltjük be, ahol ki lehet választani őket. Egy szűrő segíti a munkát, minden billentyű lenyomása után szűkíti a lehetőségeket, a beírt szöveg alapján.
A rendezvény adatlapja:

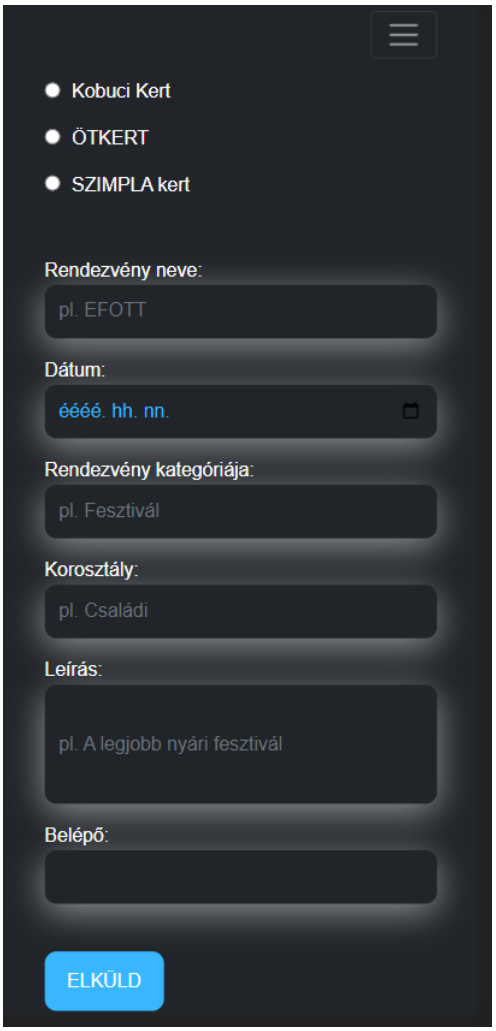
Az utolsó űrlap a ***rend_eloado*** táblába tölti fel a rendezvények és előadók egyéni azonosítóit. Ennek a célja az adott eseményhez több fellépőt csatolni, így a részletes infók megnyitásakor minden előadó megjelenik az oldalon, listába szedve. Itt kettő szűrőt is használunk, csak ki kell keresni az adott rendezvényt, amit frissíteni szeretnénk az újabb előadókkal.



A mobilos felület megjelenítése is megfelelően működik:

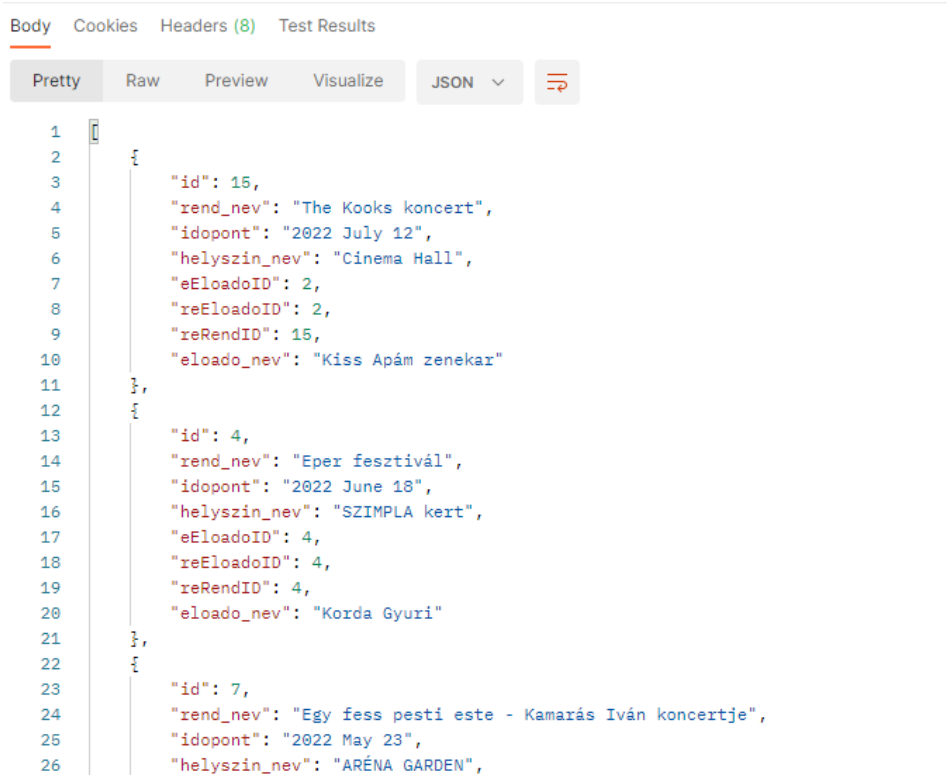




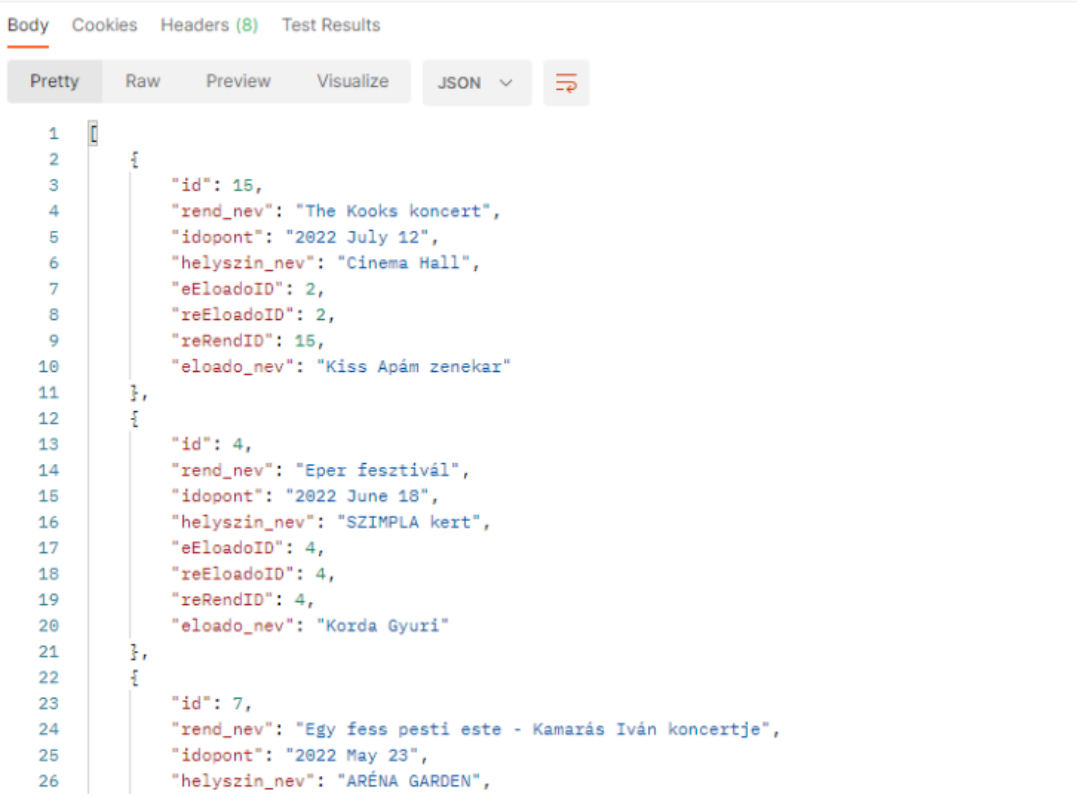


Tesztelési adatok a Postman segítségével

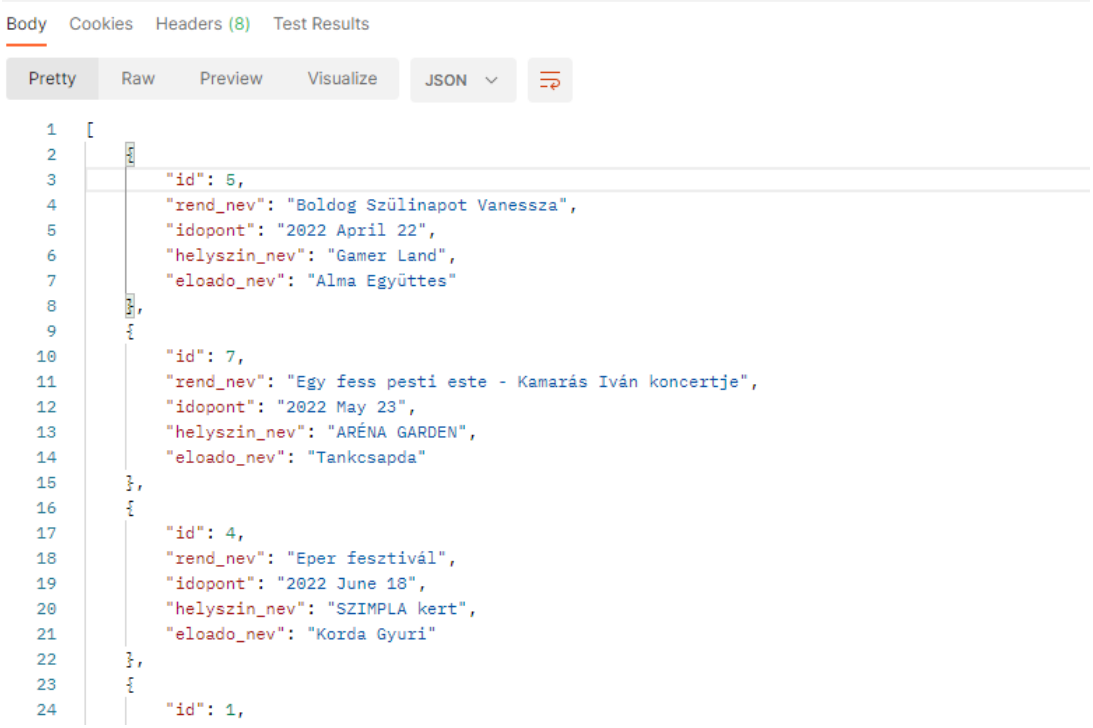
A kezdőlap betöltésénél használt útvonal („**/kezdolap/**”) tesztelése:



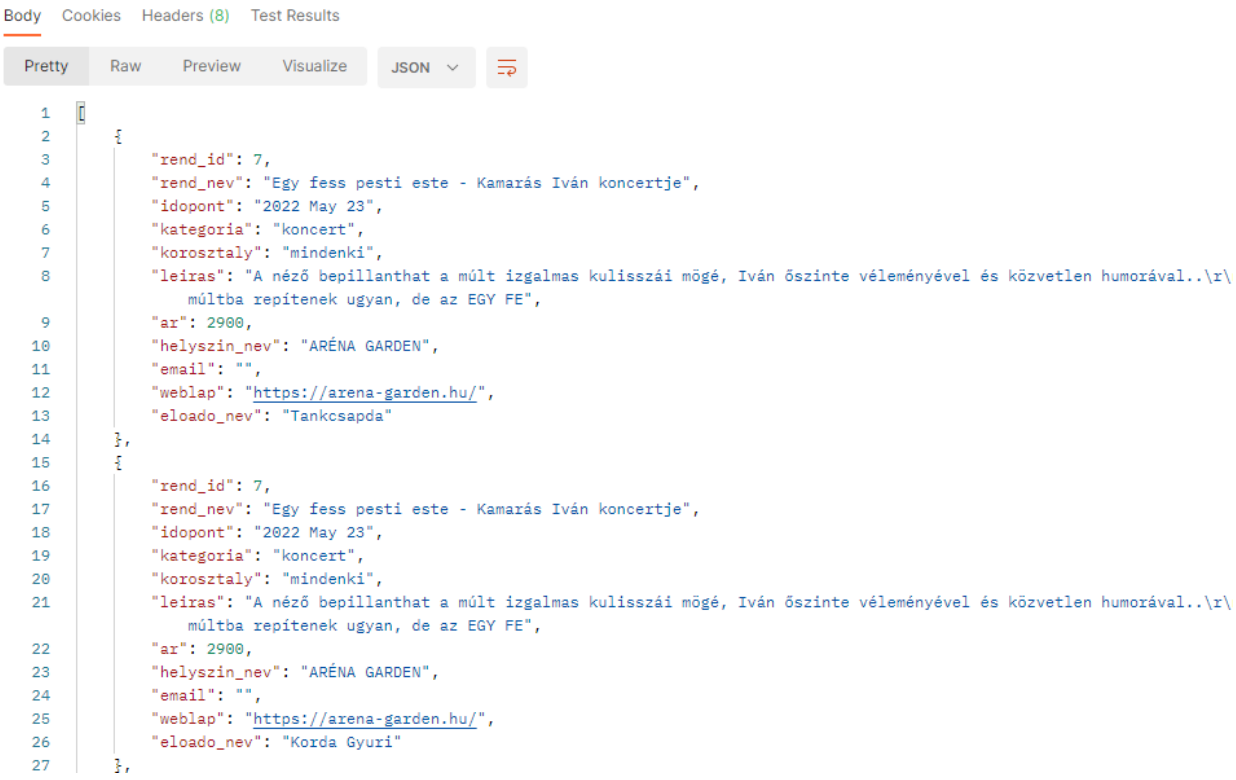
Bejelentkezve is megkapjuk a szükséges adatokat a kezdőlapon:



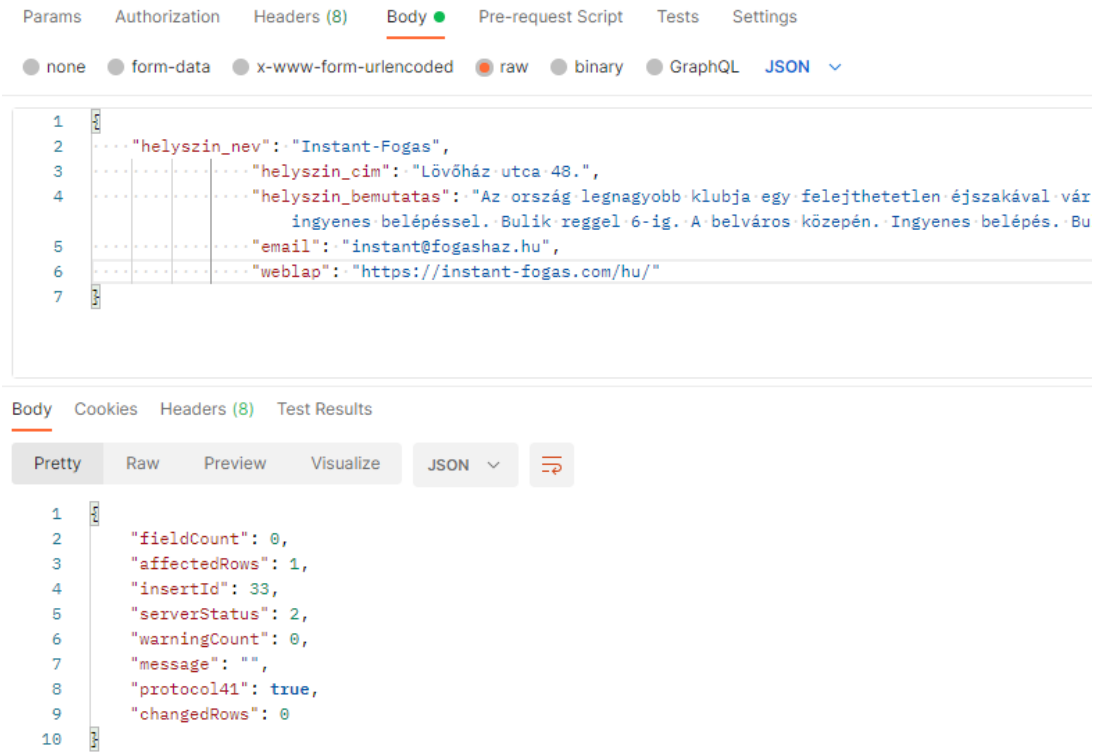
Az ábécé sorrendben való megjelenítés ellenőrzése:



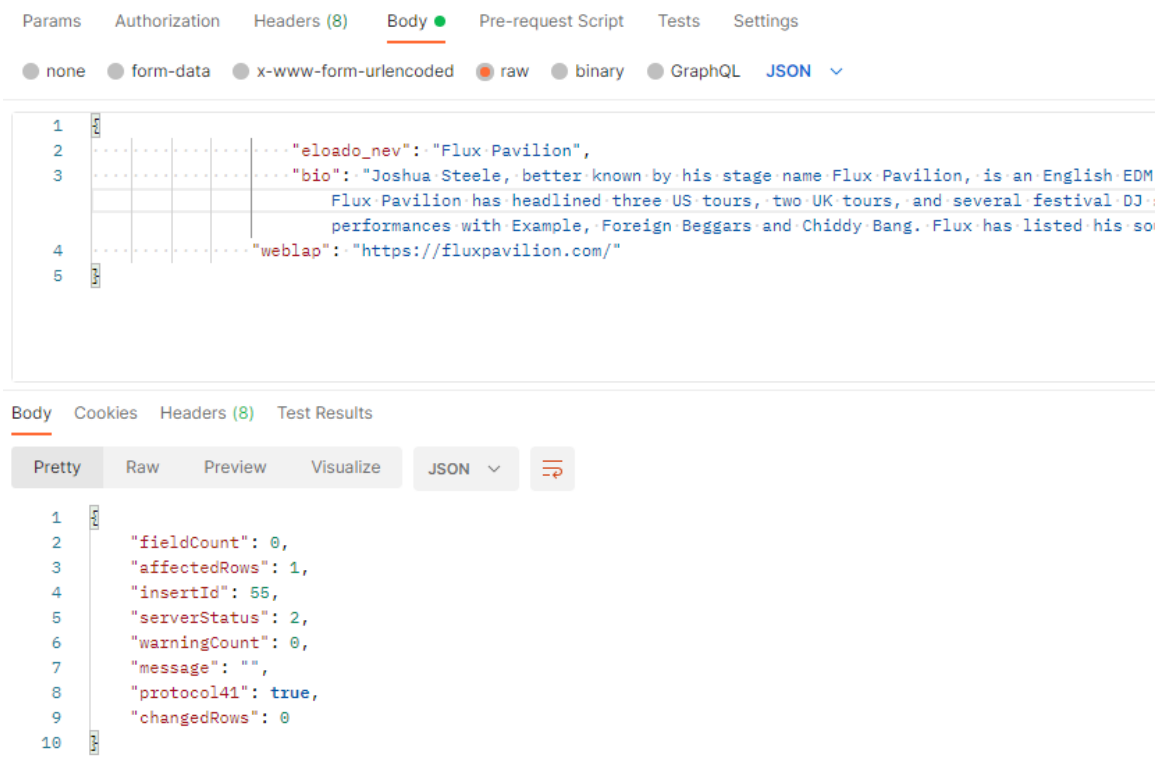
A „**/bovebben/:id**” route tesztelése (valóban betölti az összes előadót):



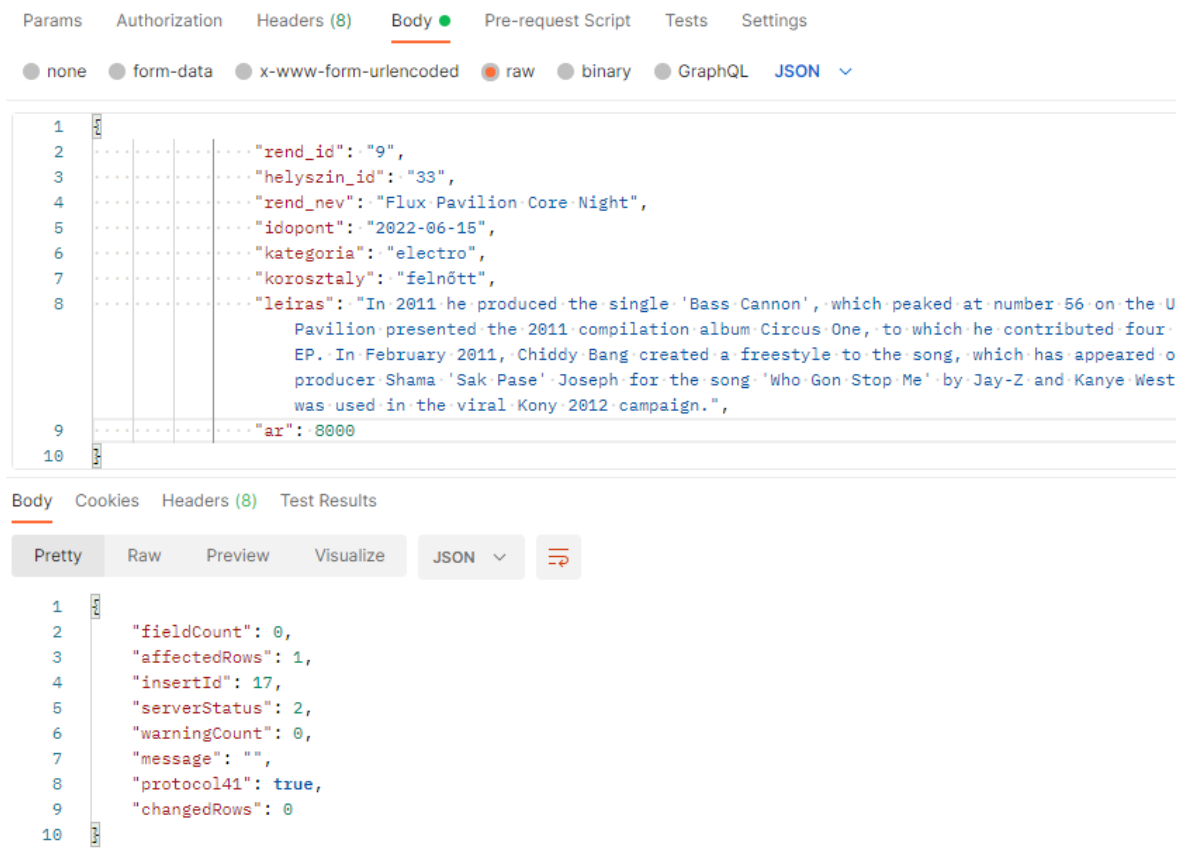
A „**/submitHelyszin/**” végpont ellenőrzése:



Az előadok feltöltéséhez szükséges útvonal(„**/submitEloado/**”) ellenőrzése:



A „**/submitRendezveny/**” tesztelése:



A „**/submitEsemeny**” végpont ellenőrzése: