Békéscsabai SZC Nemes Tihamér Technikum és Kollégium

Szakképesítés megnevezése: Szoftverfejlesztő és -tesztelő

Azonosító száma: 5 0613 12 03

VIZSGAREMEK

Könyvkölcsönző

Készítették:

Tanuló1: Balog Csaba Osztály: 2-14FE-Szoft-2223 Oktatási azonosító: 7xxxxxxxxxx

Tanuló2: Hoffer Áron Osztály: 2-14FE-Szoft-2223 Oktatási azonosító: 71440874315

Békéscsaba, 2023/2024

NYILATKOZAT

Alulírott ..... , a Békéscsabai SZC Nemes Tihamér Technikum és Kollégium tanulója kijelentem, hogy a(z)

.....

.....

című vizsgaremek elkészítésében való közreműködésem a saját, önálló munkám, az abban általam hivatkozott nyomtatott és elektronikus szakirodalom felhasználása a szerzői jogok szabályainak megfelelően történt.

Tudomásul veszem, hogy a vizsgaremek esetén plágiumnak számít:

• ha a feladat elkészítője bármilyen más forrásból vett következtetést vagy megoldást a sajátjaként tüntet fel,

• ha a dokumentációk szövegét vagy annak egy részét nem a dolgozaton szerzőként feltüntetett személy írja,

• ha a más forrásból nyert forráskódok, programelemek, adatok, ábrák valamint szövegrészek dolgozatban való felhasználása esetén a forrás megjelölése elmarad vagy a hivatkozás alapján nem azonosítható egyértelműen a forrás.

Alulírott kijelentem, hogy a plágium fogalmát megismertem és tudomásul veszem, hogy plágium esetén a vizsgaremek vizsgarész visszautasításra kerül.

Békéscsaba, 2024. év .....hó .....nap

…… ……

Balog Csaba Hoffer Áron

1. Bevezetés

Ajánlás az Online Könyvkölcsönző Szolgáltatások Használatához

A digitális kor új távlatokat nyitott meg az olvasás és az információhoz való hozzáférés terén. Az online könyvkölcsönzők, mint a jövő könyvtárai, lehetővé teszik a felhasználók számára, hogy széles körű irodalmi és tudományos művekhez férjenek hozzá kényelmesen, gyorsan és hatékonyan. Ezek a platformok nem csupán könyveket, hanem audiokönyveket és e-könyveket is kínálnak, így mindenki megtalálhatja a számára legmegfelelőbb formátumot. Az alábbiakban összefoglalom, hogy miért érdemes az online könyvkölcsönzőket választani az olvasás modern formájaként.

1. Kényelem és azonnali hozzáférés

Az online könyvkölcsönzők legnagyobb előnye a kényelem. Nem szükséges elhagyni otthonunkat vagy munkahelyünket, hogy új könyveket szerezzünk, vagy visszavigyük a már elolvasottakat. Elegendő csupán internetkapcsolat, és máris böngészhetünk a különféle műfajok széles kínálatában, legyen szó klasszikus irodalomról, modern fikcióról, szakirodalomról vagy akár gyermekkönyvekről. Az e-könyvek többsége azonnal elérhető, így nem kell várakoznunk a kívánt tartalomra.

2. Gazdag és változatos kínálat

Az online könyvtárak kínálata gyakran meghaladja a hagyományos könyvtárakét, mivel könnyebben bővíthető és frissíthető. Különösen előnyös ez azok számára, akik speciális, ritkább témák iránt érdeklődnek, vagy több nyelven olvasnának. Az online platformok lehetővé teszik, hogy globális szinten hozzáférjünk különböző kultúrák irodalmához, és ezáltal bővíthetjük ismereteinket és horizontunkat.

3. Környezetbarát alternatíva

A digitális könyvek kölcsönzése jelentősen csökkenti a nyomtatott könyvek előállításának és szállításának környezeti lábnyomát. Kevesebb papírt használunk, csökken a szükséges logisztikai tevékenység, így a szén-dioxid-kibocsátás is mérséklődik. Ezáltal az online könyvkölcsönzők nemcsak kényelmes, hanem fenntartható választást is kínálnak az olvasásra.

4. Személyre szabott ajánlások és fejlett keresési lehetőségek

Az online könyvkölcsönzők az AI és a big data technológiák segítségével képesek személyre szabott ajánlásokat készíteni. Ezáltal az olvasók gyorsabban találhatják meg az őket érdeklő tartalmakat, és fedezhetnek fel új szerzőket és műfajokat, amelyek esetleg korábban elkerülték a figyelmüket. Az intelligens keresési funkciók lehetővé teszik, hogy specifikus kulcsszavak alapján szűrjük a keresést, így pontosabban és gyorsabban érhetünk el eredményeket.

Összefoglalva, az online könyvkölcsönzők használata számos előnnyel jár az olvasók számára. Ezek a platformok kényelmes, gyors és hatékony hozzáférést biztosítanak a világ irodalmi és tudományos kincseihez, miközben támogatják a fenntartható fejlődést és a kulturális sokszínűséget. Az online könyvkölcsönzők így nem csupán a modern olvasás eszközei, hanem a jövő könyvtárai is egyben.

2. Fejlesztői dokumentáció.

Az alábbi fejezetben bemutatásra kerül a webalkalmazásunk fejlesztői környezete

2.1. Fejlesztői környezet

A webalkalmazásunk elkészítése során az alábbi környezetben dolgoztunk. Rendszerprogramnak a *Microsoft Windows 10 Pro* operációs rendszert választottuk. Mind frontend és backend oldalon a *Visual Studio Code 1.88.1*-es verzióját használtuk. A VS Code egy ingyenes, nyílt forráskodú kódszerkesztő program, amelyet a Microsoft fejlesztett ki. Platformfüggetlensége, több nyelvű támogatottsága, GIT kezelése, intelligens kódkiegészítő funkciója (IntelliSense), könnyű bővíthetősége (kiegészítők hozzáadása) miatt széles körben elterjedt a programozók körében. A frontend rész React Js-, Tailwind CSS keretrendszerekkel, és Vite build eszközzel került elkészítésre. A *React Js* Javascript könyvtárrendszer, amelyet elsősorban a könnyen újrahasznosítható komponens alapú fejlesztés miatt vált közkedvelté. A *Tailwind CSS* egy segédosztály-alapú stílus keretrendszer, amely előre definiált osztályokat bocsájt rendelkezésünkre, amiknek a segítségével gyorsabban, hatékonyabban lehet

weboldalak kinézetét kialakítani. A *Vite* egy modern build eszköz frontend fejlesztéshez, amelynek fő célja, hogy gyorsabb fejlesztési élményt nyújtson. Egyik kulcsjellemzője az azonnali szerverindítás.

A *XAMPP* egy ingyenes, nyílt forráskódú keresztplatform webszerver-szoftvercsomag, amelyet főleg helyi szerverkörnyezetek kialakításáért, futtatásáért, teszteléséért használnak a fejlesztők. Amiatt választottuk a szoftvercsomagot, mert könnyen telepíthető és használható környezetet biztosít számunkra anélkül, hogy külön kellene az egyes összetevőket telepíteni, konfigurálni a gépünkön. A XAMPP csomagunk 8.2.0. verziójú, amely az alábbi szoftvereket tartalmazza:

- *Apache 2.4.54 HTTP Server*. A világ egyik, talán legnépszerűbb webszerver szoftvere, amelynek fő funkciója a HTTP kérések fogadása és a weboldalak kiszolgálása.
- *MariaDB 5.4.27* nyílt forráskódú, SQL alapú relációs adatbázis-kezelő rendszer, amely kompatibilis a MySQL-lel, és gyakran használják webalkalmazások adatbázisaként.
- *PHP 8.2.0* széles körben elterjedt szerveroldali, HTML oldalba is ágyazható scriptnyelv, amelyet kifejezetten webfejlesztésre terveztek. A PHP kód a szerveroldalon fut és ott is generálódik a tartalom.
- *phpMyAdmin 5.2.0* egy ingyenes, népszerű webalapú eszköz MySQL és MariaDB adatbázisok vizuális kezelésére. A szoftver intuitív webes felületet biztosít az adatbázis-adminisztrátorok, webfejlesztők részére, amellyel jelentős mértékben elősegítik az adatbázis műveletek elvégzését.

A backend oldal *Node.js Express*-el lett kialakítva, amely egy népszerű webalkalmazás keretrendszer Node.js platformhoz. Főbb előnyei közé tartozik az egyszerű használat, a könnyű bővíthetőség middleware-kel és modulokkal. Használatával könnyen kialakíthatunk és fejleszthetünk webalkalmazásokat és API-kat.

Hardware-es környezetről részletes bemutatás nem készül, mivel az alkalmazásunk nem túl gépigényes, gyakorlatilag a program indításához szükséges szoftverek megléte a kritérium.

2.2. A vizsgaremek elkészítéséhez használt adatbázis bemutatása.

Az adatok tárolására a konyvkolcsonzo adatbázist hoztuk létre. Az adatbázisunk UTF-8 (Unicode Transformation Format - 8-bit) karakterkódolású, ami támogatja a magyar nyelvben használt karaktereket, beleértve az ékezetes betűket is, továbbá az illesztése

utf8_hungarian_ci. Az utóbbi esetében a *hungarian* arra utal, hogy a magyar nyelvű szövegek kezelésére optimalizált, a magyar nyelvi szabályoknak megfelelően kezeli a kis- és nagybetűk közötti különbségeket, és az ékezetes karaktereket is megfelelően sorrendbe állítja. A *ci* a Case Insensitive rövidítése, ami azt jelenti, hogy a nagy- és kisbetűk közötti különbséget nem veszi figyelembe az összehasonlítás során. Ez fontos lehet olyan esetekben, ahol nem szeretnénk, hogy a keresés vagy rendezés érzékeny legyen a betűk nagyságára. Vizsgaremekünkben a könyvek kezdőbetűje alapján történő keresése tökéletes példája ennek.

1. ábra Konyvkolcsonzo adatbázis táblái, mezői és kapcsolatai.

Forrás:

http://localhost/phpmyadmin/index.php?route=/database/designer&db=konyvkolcsonzo

2.1.1. books tábla.

Ez a tábla a könyvkölcsönző rendszer alapját képezi, amely a különböző könyvek adatait tárolja. Az egyes mezők a következő adatokat tartalmazzák:

- BookID(INT(11)). A könyvek egyedi azonosító száma (PRIMARY KEY). Egész szám típusú (integer) érték. A mező AUTO INCREMENT, azaz további könyv rögzítése során automatikusan növekvő számsor.
- author(VARCHAR(255)). A könyv szerzőjének neve. Karakterlánc típusú (varchar), maximum 255 karakter hosszúságú.
- title(VARCHAR(255)). A könyv címe, ami az egyik legfontosabb attribútum a könyvkeresés során.
- publisher(VARCHAR(255)). A könyv kiadójának neve.
- the_year_of_publishing(INT(11)). A könyv kiadásának éve. Egész szám.
- description(TEXT). A könyv rövid leírása, amely elősegíti a tartalom előzetes megismerését. Típusát tekintve text, amely hosszabb szöveg tárolására is alkalmas.
- language (VARCHAR(50)): A könyv nyelve, ami fontos szempont lehet a kölcsönzők számára VARCHAR típusú, 50 karakterig.
- number_of_pages(INT(11)). Az oldalszám, egész szám, ami információt nyújt a könyv hosszáról.
- cover(VARCHAR(50)). A könyv borítójának típusa (pl. puha, kemény stb.).
- weight(VARCHAR(50)). A könyv súlya. A rekord tartalmazza a mértékegységet is a nemzetközi szabványoknak megfelelően.
- ISBN(VARCHAR(50)). Nemzetközi szabványosított könyvszám, amely egyedileg azonosít minden kötetet világszerte. A rekord tartalmazhat kötőjelet is. Mind a régi- és új formátum tárolására alkalmas.
- itemcode(VARCHAR(50)). Egy másik egyedi kód, ami a belső nyilvántartásban használható.
- price(DECIMAL(10,2)). A könyv eredeti ára. Tizedes pontosságú (DECIMAL) típusú, amely 10 számjegyet tartalmaz, ebből 2 a tizedes utáni rész.
- discounted_price(DECIMAL(10,2)). Akciós, kedvezményes ár.
- book_url(VARCHAR(255)). Egy URL, amely a könyv képére vagy részleteire mutat.
- categoryID(INT(11)). A könyv kategóriájának száma, amely megmutatja, hogy mely kategóriába tartozik a könyv. A mező idegen kulcs(FOREIGN KEY).

2.1.2. borrowedbooks tábla.

A könyvkölcsönzési tranzakciókat rögzítő tábla, amely a következő mezőket tartalmazza:

- Id(INT(11)). A kölcsönzés egyedi azonosító száma tétel szerint (PRIMARY KEY). A mező AUTO INCREMENT, azaz új megrendelés rögzítése során automatikusan növekvő számsor.
- BookID(INT(11)). Kapcsolat a books táblához, a kölcsönzött könyv egyedi azonosítóját tartalmazza.
- cartID(INT(11)). A kölcsönzés során generált „rendelésszám". Lehetővé teszi több könyv egyidejű kölcsönzésének nyomon követését.
- end_of_borrowment(DATE). A kölcsönzés lejáratának dátuma. Dátum típusú mező.
- quantity(INT(11)): A kölcsönzött könyvek száma.
- end_of_borrowment(DATE). A kölcsönzés kezdetének dátuma.
- UserID(INT(11)). A kölcsönző felhasználó egyedi azonosító száma. Kapcsolat a users táblához.
- bringedBack(TINYINT(1)). Logikai érték(1-Igaz, 0-Hamis). Jelzi, hogy az adott könyv vissza lett-e már hozva.

2.1.3. categories tábla.

A könyvkategóriákat tartalmazó tábla, amely segít a könyvek tematikus csoportosításában:

- categoryID(INT(11)). A könyvkategória egyedi azonosítója.
- category(VARCHAR(255)). A kategória neve, pl. „Család és szülők", "Ezotéria" stb.

2.1.4. users tábla.

A rendszer felhasználóinak (user és admin jogosultsággal rendelkezők is) adatait tartalmazó tábla, beleértve a következőket:

- UserID:(INT(11)). A kölcsönző felhasználó egyedi azonosító száma (PRIMARY KEY). A mező AUTO INCREMENT, azaz új regisztráció során automatikusan növekvő számsor.
- IsAdmin(TINYINT(1)). Logikai érték(1-Igaz, 0-Hamis). Értéke meghatározza, hogy a felhasználó milyen jogosultsággal bír (user jogosultság esetén értéke 0, míg admin esetében az érték 1).

- Email(VARCHAR(100)). A felhasználó email címe. A mező értéke UNIQUE, azaz egyedi lehet. A megkötés azért szükséges, hogy a regisztráció során ne lehessen kétszer ugyan azzal az email címmel regisztrálni.
- Name(VARCHAR(255)). A felhasználó neve.
- Phone(INT(11)). A felhasználó telefonszáma.
- ZIP(SMALLINT(6)).Irányítószám. Kisebb tartományú egész szám típus.
- City(VARCHAR(100)). A felhasználó lakhelyének települése.
- Address(VARCHAR(100)). A felhasználó lakcíme.
- Pass(VARCHAR(128)). A regisztrált felhasználó jelszavának hash-elt (speciális hasítófügvénnyel generált titkosított forma) alakja.

2.1.5. Adatbázis kapcsolatok.

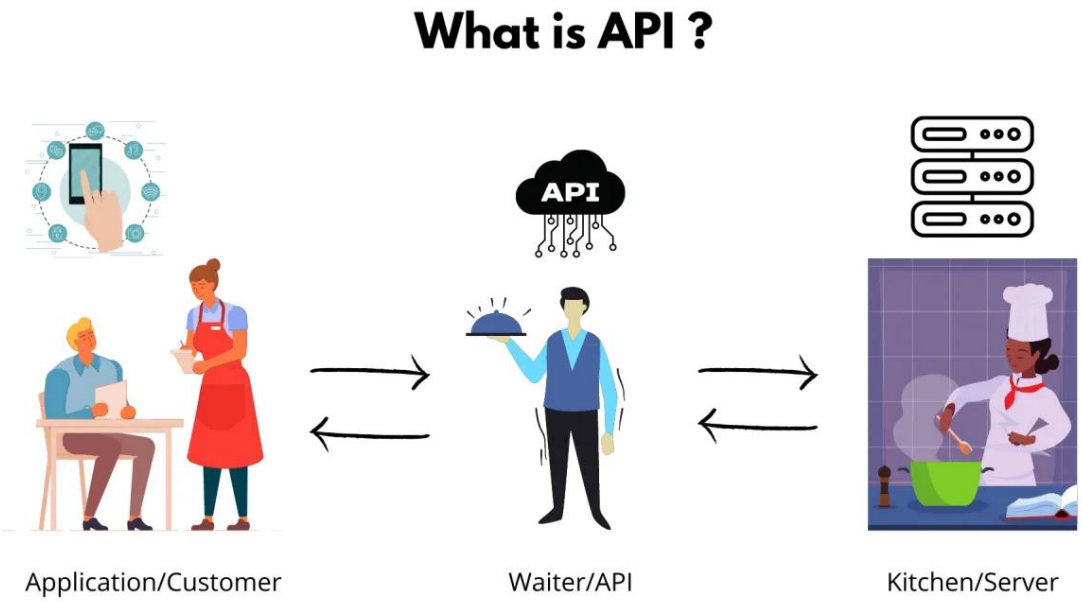
- A borrowedbooks tábla BookID mezője kapcsolódik a books tábla BookID mezőjéhez.
- A borrowedbooks tábla UserID mezője kapcsolódik a users tábla UserID mezőjéhez.
- A books tábla categoryID mezője kapcsolódik a categories tábla categoryID mezőjéhez.

2.3. Webalkalmazás felépítése.

A webalkalmazásunk két fő komponensre szeparálható, frontendre és backendre.

Frontend, másnéven a kliens oldali rész egyik funkcionalitása a felhasználóval történő interakció. Ez magában foglalja a weboldalak megjelenítését, a felhasználói felületet (UI), valamint a felhasználói élményt (UX) meghatározó elemeket. Úgy is szoktak fogalmazni, hogy a frontend fejlesztők általi produktum az, amit a felhasználó lát, érzékel.

A backend a szerver oldali rész, amelyről elmondható, hogy az alkalmazás „agyát” képezik. Ez a réteg kezeli az adatbázis-műveleteket, felhasználói hitelesítéseket, adatok kezelését és tárolását, felhasználói kéréseket dolgoz fel stb. A backend fejlesztők feladati közé tartozik az API-k kialakítása, amelyeken keresztül a frontend adatokat kérhet. Az Application Programming Interface, magyarul hívhatjuk alkalmazásprogramozási felületnek is. Az alábbi ábra jól szemlélteti működését:



2. ábra API működése Forrás: https://medium.com/@pawan329/what-is-an-api-a-step-by-step-guide-e6858b6e1016

A fenti ábra szemlélteti egy ételrendelés folyamatát, amelyet egy mobilapplikáción küld el a felhasználó. A „pincér” az API, míg a szerver a „konyha”. A felhasználó a rendelés során módosításokat hajthat végre a kívánt ételen, pl. extra sajtot kér rá, vagy hagyma nélkül kéri. A rendelés leadásával a kérés a „konyhához” érkezik. Az API biztosítja, hogy az étterem megértse a rendelésedet, az azokon végrehajtott módosításokat, és képes visszaküldeni a várható szállítási időt, valamint megerősíteni a rendelést.

2.3.1. Szerver oldal kialakítása, annak logikája.

A szerver oldal kialakításához Node.js Express keretrendszert és MySQL relációs adatbázis-kezelő rendszert használtunk. A projekt kezdeti beállításainak létrehozását követően (npm init) az alábbi csomagok kerültek feltelepítésre:

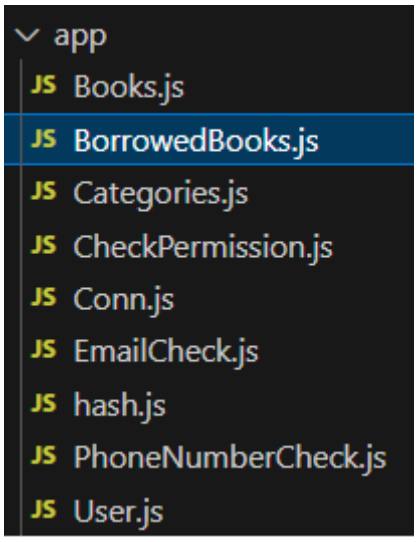
- npm install express : Express.js keretrendszer installálása
- npm install nodemon: Nodemon fejlesztői környezet telepítése. Célja a Node.js alkalmazás automatikus újraindítása változtatásokat követően
- npm install mysql2: Mysql csomag hozzáadása. Célja a kommunikáció és kapcsolatteremtés kialakítása a MySQL adatbázissal

- npm install cors: Cross-Origin Resource Sharing middleware-t telepítése. Funkciója a CORS biztonsági mechanizmus okozta hozzáférés-megtagadás (pl. eltérő domainekről származó források elérése) feloldása.
- npm install cookie-parser: Cookie-parser middleware installálása. Kezelési a böngésző cookie-jait a Node.js alkalmazásban.
- továbbá a package.json fájlba a „type”:”module” módosítás beállításra került annak érdekében, hogy a „klasszikus-nak” mondható *require* CommonJS modulbetöltési módszer helyett az *import* ECMAScript modulbetöltési (ESM) lehetőséggel éljünk.

A fent részletezett telepített függőségek állapota jól láthatóak a projektmappában elhelyezkedő package.json fájlban is.

3. ábra package.json fájl tartalma

A projektmappában az alábbiak találhatóak: *app* mappa, *node_modules* mappa továbbá *index.js*, *package-lock.json* és *package.json* fájl. A package.json fájlt a kezdeti beállításokat követően hozza létre a rendszer, majd a package-lock.json fájlt és a node_moduls mappát az npm istall express parancs kiadását követően (gyakorlatilag npm install parancs megadását követően attól függetlenül ,hogy expesst használunk-e).



4. ábra app mappa tartalma

A kialakított Node.js környezetünkben a belépési pont az *index.js*. A belépési pont az a fájl, amelyből a futtatási környezetünk elindítja/betölti az alkalmazásunkat. Mondhatni úgy is, hogy az alkalmazásunk „indító motorja”. Itt kerülnek beállításra az alapvető konfigurációk, importálásra a külső csomagok, modulok, végpontok kialakítása (routing -útválasztás definiálása), alkalmazás különböző részeinek összekapcsolása, a szerver portjának beállítása. Tekinthetünk erre a fájlra úgy is, mint egy összekötő elem.

2.3.1.1. Az app mappa tartalma

Az *app* mappa tartalmazza az egyes funkciók megvalósításáért felelős Javascript fájlokat.

Minden egyes fájl funkcionálisan került kialakításra. A következőkben röviden ismertetem, hogy az egyes fájlok milyen funkciók ellátásáért felelősek, a későbbiekben kijelölök egyet közülük és azt részletesebben elemzem. Struktúráját tekintve a legtöbb scriptfájl hasonlóan épül fel.

1. *BorrowedBook.js scriptfájl* a kölcsönzések kezelésért felelős. 1 db. osztályt (BorrowedBooks) definiál és 8 db. aszinkron metódust tartalmaz.
    - getBorBooks() – az összes kölcsönzött könyv lekérése, adattábla sorok összekötésével(INNER JOIN) .
    - getBorrowsByUser(userID) – a cookies-ban tárolt userID alapján a kikölcsönzött könyvek megjelenítése. Mivel a feldolgozásra kerülő userID a cookies-ból érkezik, gyakorlatilag arra szolgál, hogy a rendszerbe belépett felhasználó a saját rendeléseit kilistáztassa.
    - getBorrowsByUserID(userID, isAdmin) – a felhasználó azonosítója alapján (userID), amelyet URL paraméterként kap a metódus, a kikölcsönzött könyvek kilistázása. Az előzőtől annyiban tér el, hogy paraméterként érkezik a userID és nem

a sütikből, ezáltal admin jogosultság szükséges hozzá. A megfelelő hozzáférési szintet az isAdmin paraméter alapján a metódus ellenőrzi.

- getBorrowsByCartID(CartID, userID, isAdmin) - a rendelés azonosító száma (cartID) alapján a kikölcsönzött könyvek megjelenítése. A paraméter URL-ben érkezik, rendszergazdai jogosultság kell hozzá. A megfelelő hozzáférési szintet az isAdmin és a userID paraméterek alapján a metódus ellenőrzi.ű
- checkBorrows(bookID, userID) – ellenőrzi a felhasználó korábbi rendeléseit. Paraméterként két értéket kap, a rendszerbe belépett felhasználó userID-ját és a könyv egyedi azonosítóját. Amennyiben az adott user olyan bookID-val rendelkező könyvet szeretne kivenni, mint amilyet korábban kölcsönzött és nem vitt vissza, abban az esetben ***„Már kivettél egy ilyen könyvet, és még nem hoztad vissza!”*** üzenettel tér vissza.
- addBorrow(borrowDatas) – a rendelés rögzítéséért felelős. Paraméterként egy vagy több objektumot kap. ??
- updateOrder(borrowData, isAdmin) – a visszahozatal (bringedBack) logikai (1-IGAZ, 0-HAMIS) értékmódosítója. Az admin jogosultsággal bíró személy beállíthatja, hogy a könyv vissza lett-e hozva. Hozzáférési jogosultságot is ellenőriz.
- deleteOrder(cartID, userID, isAdmin) – egy rendelés törlése az URL-ben kapott rendelés azonosító alapján. Használatához rendszergazdai jogosultság szükséges (metódus ellenőrzi).

2. *Categories.js scriptfájl* a könyvkategóriák lekérdezéséért felelős, 1 db. osztályt (Categories) és 2 db. aszinkron metódust tartalmaz.

- getCategory(id) – könyvkategória lekérése URL paraméterként kapott érték alapján. Egy konkrét kategória kikeresésére szolgál.
- getCategories() – az összes könyvkategória kiíratása

3. *CheckPermisson.js scriptfájl* az adminisztrátori jogosultságot ellenőrző függvény. Bizonyos műveletekhez (módosítás, törlés, új könyv felvitele stb.) rendszergazdai szint szükséges.

4. *Conn.js scriptfájl* az adatbázishoz (konyvkolcsonzo) való csatlakozásért felelős.

5. *EmailCheck.js scriptfájl* a regisztrációkor megadott email cím formai követelményeit hivatott ellenőrizni reguláris kifejezés használatával.

6. *PhoneNumberCheck.js* a regisztrációkor megadott telefonszám formai követelményeit hivatott ellenőrizni reguláris kifejezés használatával.

7. *hash.js scriptfájl* a regisztrációkor megadott jelszó titkosításáért felel. A user táblába a megadott jelszó a crypto modul segítségével SHA-512 hash algoritmussal hexadecimális formátumba kerül rögzítésre.

8. *User.js scriptfájl* a felhasználókkal kapcsolatos műveletekért felelős. 1 db. osztályt(User) és 7 db. aszinkron metódust tartalmaz.

- register(user) – a felhasználók regisztrációját elvégző eljárás. 5 hiba ellenőrzését végzi(email cím formátuma, jelszó és megerősített jelszó egyezősége, jelszó hosszúsága, telefonszám formátuma, email cím adatbázisban történő szereplése), amennyiben hibát talál visszatérési értéke egy tömb (**errors**). Sikeres regisztráció esetén **200**-as státuszkód küldése.
- login(user, res) – a felhasználók belépéséért felelős metódus. A nem megfelelő felhasználónév/jelszó páros esetén hibaüzenetet küld. Sikeres azonosítás/belépés esetén cookies-kat küld vissza, „userID”,”isAdmin”,”email” kulcs megnevezéssel, az értékeket a regisztrált felhasználó rekordjából állítja be. A süti lejárati időpontját 7 napra korlátozza.
- getUsers(isAdmin) – az összes regisztrált felhasználót kilistázza. Visszatérési értéke egy tömb (**data[0]**).
- deleteUser(userID, c_userID, isAdmin) – a regisztrált felhasználók törléséért elvégző metódus. A paraméter az URL-ból érkezik (userID), továbbá ellenőrzésre kerül a jogosultság is (isAdmin, c_userID). Siker esetén **200**-as státuskód ***„Sikeres törlés!”*** megjegyzéssel, ellenkező esetben **404** – ***„A felhasználó nem található a megadott ID alapján!”***
- checkEmail(email) – a megadott paraméter (emailcím) alapján ellenőrzi (gyakorlatilag megszámolja), hogy az adatbázisban szerepel az email cím. Ha van egyezőség visszaadja egész számban (ha szerepel az adatbázisban értéke 1, amennyiben nem, 0).

- resetPass(user) – a regisztrációkor beállított jelszó módosítására szolgáló metódus. A kapott paraméter a felhasználó email címe. Amennyiben nem szerepel az adatbázisban a megadott email cím (itt van alkalmazva a korábban bemutatott checkEmail eljárás) ***„A megadott email cím nem szerepel az adatbázisban!”*** üzenettel és **404**-es státuszkóddal tér vissza.
- searchUsers(char) – regisztrált felhasználók szűrése kezdőbetű(k) alapján.

9. A Books.js scriptfájl a könyvkezelésekhez tartozó osztályt (Books) és metódusokat tartalmazza. Struktúráját tekintve a többi Javascript fájl is hasonlóan épül fel.

A Book.js tartalmaz egy Book osztályt, aminek 7 db. aszinkron és 1 db. szinkron metódusa van. Minden metódus az általa elvégzett műveletekre utal. Az összes aszinkron metódus *try-catch* hibakezelési szerkezetet tartalmaz, annak az érdekében, hogy az applikációnk kezelni tudja futási időben a fellépő hibákat vagy váratlan eseményeket anélkül, hogy leállna vagy összeomlana. A *try blokk* tartalmazza azt a kódsort, amely hibaforrás lehet (pl. referenciahiba, típushiba stb.). A *catch blokk* „lép életbe”, amennyiben a *try blokkban* kivétel(hiba) keletkezett. A *catch blokk* paraméterként kap egy hibát(exception), amely információt szolgáltat a kivételről, a hiba típusáról. Részletesebb magyarázat a Book.js metódusaira:

- checkErrors(book): Új könyv felvitele során ellenőrzi, hogy a megadott adatok nem hiányosak-e. 14 szempontot ellenőriz. Amennyiben hibát talál egy **errors** nevű tömbbel tér vissza.
- searchBook(char): A könyv nevére szűrési funkciót lát el kezdőbetű(k) alapján, URL paraméterként érkezett betű, vagy betűkombináció az átadott paraméter. Amennyiben a kapott paraméter alapján talál könyvneveket, visszatér egy tömbbel (***data[0]***), amely tartalmazza az összes rekordot. A tömbben található minden rekord objektumként tárolódik. A tömb mellet egy ***200***-as státuszú kóddal tájékoztatjuk a frontend-et, hogy sikeres volt a lekérdezés. Amennyiben a kapott feltételeknek nem felel meg egyetlen könyvnév sem, ***404***-es státuszkód tér vissza, ***„A keresett erőforrás nem található!”*** üzenettel párosítva.
- getBooksCatID(id): Könyvek kilistázása a paraméterként megadott kategóriaszám alapján. A paraméter szintén az URL-ben „érkezik”. Példának okáért. A beérkező érték a 2-es, a hozzá tartozó kategória elnevezés „Életmód, egészség”. A metódus visszaad minden olyan könyvet, amelyek a fenti kategóriában szerepelnek.

- Amennyiben a kategória száma alapján talál könyveket visszatér egy tömbbel (***data[0]***), továbbá egy ***200***-as státuszú kóddal, ellenkező esetben ***404***-es státuszkód, továbbá ***„A keresett erőforrás nem található!”*** üzenet a visszatérő érték.

➢ getBook(id): Könyv keresése paraméterként átadott(URL-ben) egyedi BookID azonosító alapján. Az adatbázisban szereplő összes könyv egyedi BookID azonosítóval van ellátva. Ha az adatbázis tartalmaz a kapott ID alapján könyvet, a visszatérési érték egy objektum lesz (***data[0][0]***), amely tartalmazza a keresett könyv adatait. A sikeres művelet elvégzéséről ***200***-as státuszkódot küldünk vissza. Ellenkező esetben: ***404***-es státuszkód, továbbá ***„A keresett erőforrás nem található!”***

➢ getBooks() – összes könyv kilistázása

```
async getBook(id) {
    const sql = `SELECT * FROM books WHERE BookID = ?`;

    try {
        const data = await conn.promise().query(sql, [id]);

        if(data[0].length !== 0) {
            return {
                status:200,
                messages: data[0][0]
            };
        } else {
            return {
                status:404,
                messages:["A keresett erőforrás nem található!"]
            };
        }
    } catch(err) {
        console.log(err);
        console.log(err.errno);
        console.log(err.sqlMessage);

        return {
            status:503,
            messages:["A szolgáltatás jelenleg nem elérhető! Próbálja meg később!"]
        };
    }
}
```

5. ábra getBook(id) metódus forráskódja

- deleteBook(id, userID, isAdmin) – Könyv törlése az adatbázisból URL paraméterként érkezett ID(könyvazonosító) alapján. Jogosultságot ellenőriz. Adatbázisban nem szereplő ID esetén ***„Az erőforrás nem található”*** hibaüzenettel tér vissza.
- addBook(book, userID, isAdmin) – Új könyv felvitele az adatbázisba. A metódus jogosultságot is ellenőriz (userID, isAdmin). A kapott objektum értékeinek rögzítése esetén ***„Sikeres felvitel”*** üzenettel és **200**-as státuszkóddal tér vissza.
- updateBook(book, userID, isAdmin) – adatbázisban szereplő könyv adatainak módosítását/felülírását eredményezi. Admin hozzáférést ellenőriz.

2.3.1.2. Az index.js fájl tartalma

Ahogy az korábban bemutatásra került, az index.js scriptfájl a szerver oldali belépési pont. Kezdetben importálásra kerülnek a használatos modulok, továbbá az app mappában definiált funkcionalitást jelentő scriptek is. Majd a CORS biztonsági mechanizmus okozta hozzáférés-megtagadás elkerülése érdekében beállítjuk a cors middleware-jét, hogy a http://localhost:5173-as port-ról jövő kéréseket (a vizsgaremekünk frontendoldalától) engedélyezze (***origin: http://localhost:5173”).*** Lehetővé tesszük, hogy a böngészők küldhessenek hitelesítő adatokat (pl. sütiket) a különböző eredetű (cross-origin) kérések során a ***credentials:true*** beállítással, ezt követően meghatározzuk, hogy a fenti konfiguráció milyen metódusokra vonatkozik („GET”,”POST”,”DELETE”. stb.).

```
const app = express();
app.use(cors({
    origin:"http://localhost:5173",
    credentials:true,
    methods: "GET,HEAD,PUT,PATCH,POST,DELETE"
}));
```

6. ábra CORS beállítás az index.js fájlban

A tényleges műveleteket végző definiált osztályokból (Categories, Books, User, BorrowedBooks) úgynevezett objektum-példányokat (c, b, u, bo) hozunk létre (példányosítás). Az egyes példányok rendelkeznek az osztályban meghatározott tulajdonságokkal és

metódusokkal (pl. c példány rendelkezik a Categories osztály metódusaival). Ezeket a példányokat fogjuk a későbbieken felhasználni a végpontok kialakításához.

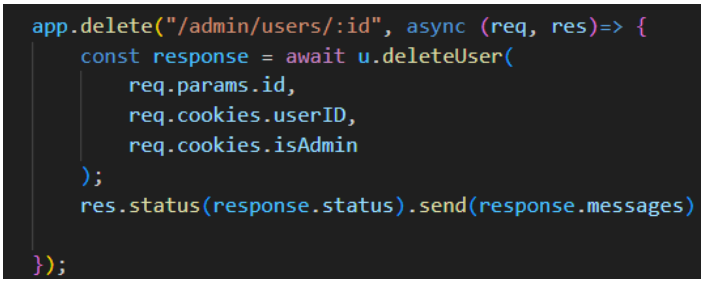
```
const c = new Categories();
const b = new Books();
const u = new User();
const bo = new BorrowedBooks();
```

7. ábra Példányok létrehozása az index.js fájlban

Az applikációban 22 db. végpont került kialakításra (2 db. kategória műveletek, 7 db könyvműveletek, 6 db. felhasználó műveletek és 7 db. kölcsönzés műveletek kapcsán).
A végpontok kialakítása hasonló módon történnek, logikai felépítésük azonosak, emiatt egy kerül részletesen bemutatásra. A végpont ismertetéséhez meg kell osztanom a hozzá tartozó metódust is.

8. ábra deleteUser metódus tartalma

```
async deleteUser(userID, c_userID, isAdmin) {
    const sql = `DELETE FROM users WHERE UserID = ?`;

    if(!CheckPermission(c_userID, isAdmin, true)) {
        return {
            status:403,
            messages:["Nincs jogosultságod törölni a bejegyzést!"]
        }
    }

    try {
        const response = await conn.promise().query(sql, [userID]);

        if(response[0].affectedRows === 1) {
            return {
                status:200,
                messages:["Sikeres törlés!"]
            }
        } else {
            return {
                status:404,
                messages:["A felhasználó nem található a megadott ID alapján!"]
            }
        }
    } catch(err) {
        console.log(err);
        console.log(err.errno);
        console.log(err.sqlMessage)

        return {
            status:503,
            messages:["A szolgállalás jelenleg nem elérhelő! Próbálja meg később!"]
        };
```



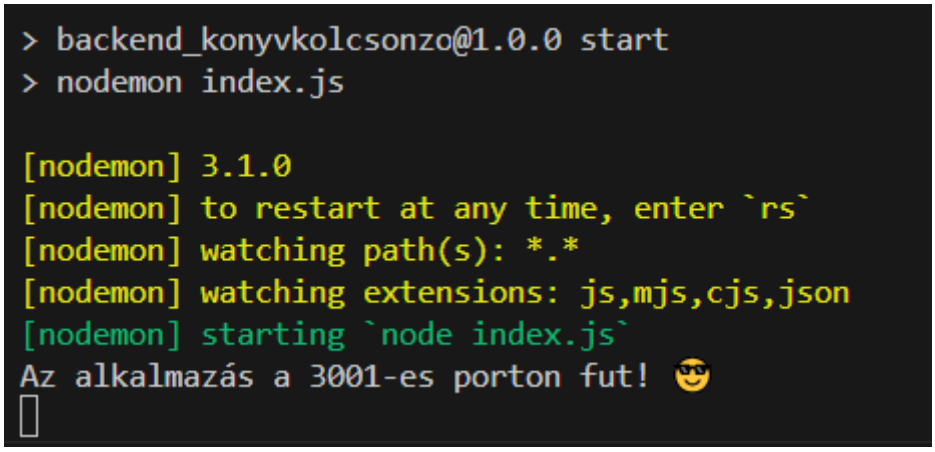
9. ábra felhasználó törlésének végpontja

A felhasználó törlésének végpont kialakításánál fontos meghatározni, hogy mely HTTP metódusokra kell reagálnia, milyen metódussal érkező kéréseket kell kiszolgánia. Ezt „az ***app.delete***”-résznél végezzük el. Ezt követi az útvonal definiálása (***„/admin/users/:id”***), amely útvonalat használni fog majd a frontend. A ***:id*** egy úgynevezett dinamikus paraméter, amely az URL részben fog érkezni, ez fogja reprezentálni a törölni kívánt felhasználó egyedi azonosítóját (userID). Az aszinkron függvények két paramétere van, amely a ***request*** (kérés – ez tartalmazza az összes információt, amelyet a böngésző küldött) és a ***response*** (válasz – módszerek és tulajdonságok, amelyeket a szerver küld vissza) ***objektum***. A response nevű változó értéke lesz a deleteUser metódus által visszaadott értékeknek.

Térjünk vissza egy pillanat erejéig a deletUser metódushoz(8. ábra). Az aszinkron metódus 3 paramétert vár, userID, c_userID és isAmdin. Az sql változó tartalmazza a konkrét sql lekérdezést(törlést). A CheckPermission függvény az adminisztrációs jogosultság ellenőrzésért felelős, amennyiben megvan a szükséges hozzásférési szint ***true*** értékkel tér vissza, ezért is negáljuk azt a feltételben(ha nincs megfelelő jogosultság térjen vissza ***403***-as státusszal és „***Nincs jogosultságod törölni a bejegyzést!***”).A ***try-blokkban*** elvégzi a függvény a megfelelő sql lekérdezést és a ***response*** változóban tárolja annak értékét. A következő feltétel azt vizsgálja, hogy a response változóban tárolt tömb első elemén egy sor módosulás történt-e (’***response[0].affectedRows ===1’***), azaz egy sor eltávolításra került? Ha igen, abban az esetben térjen vissza ***200***-as státuszkóddal és „***Sikeres törlés!***” üzenettel, ellenkező esetben ***404***-es státuszkóddal és „***A felhasználó nem található a megadott ID alapján!***”. A ***catch*** ág a nem sikeres sql művelet elvégzése után lép életbe, a konzolra kiírja a lekérdezésből eredő hibát, az sql hiba numerikus értékét(***errno***) és az sql által visszaadott hibaüzenetet (***sqlMessage***).

Kisebb kitérő után térjük vissza az eredetileg vizsgált végponthoz (9-es ábra). Részletezésre került a deleteUser metódus így tudjuk, hogy 3 paramétert vár. Az első paraméter a ***userID***, amelynek értéke az URL-ből fog érkezni (***req.params.id***), a második a c_userID értéke a sütiből fog érkezni (a bejelentkezett felhasználó azonosítója; ***req.cookies.userID***), a harmadik az isAdmin, amelynek értékét szintén a sütiből kapjuk (jogosultsági szintet jelző érték 1-Admin, 0-felhasználó; ***req.cookies.isAdmin***). Ezt követően beállításra kerül a válasz objektum státusza (***res.status***) a response változó státusz értékére (***response.status***), illetve visszatérítjük a response változó üzenet tulajdonságát (***response.message***).

A Node.js környezetben létrehozott alkalmazásunk futását a 3001-es portra állítottuk be, amelyről az elindítást követően tájékoztatást is nyújt.



10. ábra Futó alkalmazás tájékoztatása

2.3.1.3. Rossz specifikáció, gyötrelmesebb implementáció. Egy személyes tanulság megosztása backend oldalról.

A webapplikációk tervezésekor tudtuk, hogy kelleni fog olyan funkció, amely a könyv megrendeléseket kezeli. Létrehoztuk a borrowedbooks táblát (a 2.2. fejezetben bemutatásra került), és a cartID-hoz (a megrendelés azonosítója) UNIQUE megkötés került beállításra, mivel feltételeztük, hogy a rendelés azonosítója egyedi lesz és nem fog az adatbázisban ismétlődni. Az implementálás során felmerült az ötlet, hogy a könyv kikölcsönzése ne korlátozódjon egy darabra, azaz, a felhasználó egy könyvkölcsönzés során több könyvet is ki

tudjon venni, ne kelljen kilépnie és újrakezdeni. Frontend és backend oldalról is elkészült a kódsor, már csak „össze kellett a kettőt kapcsolni”. A végpont „megszólítását” követően az alábbi hibaüzenetek fogadtak minket mind backend-, mind frontend oldalról.

```
  code: 'ER_BAD_NULL_ERROR',
  errno: 1048,
  sql: 'INSERT INTO borrowedbooks \n' +
    '        (BookID, cartID, end_of_borrowment, quantity, start_of_borrowment, UserID)\n' +
    "        VALUES(NULL,NULL,NULL,NULL,NULL,'1')",
  sqlState: '23000',
  sqlMessage: "Column 'BookID' cannot be null"
}
1048
Column 'BookID' cannot be null
```

11. ábra Hiba backend oldalon könyvkölcsönzés során

```
▶POST http://localhost:3001/borrows 503 (Service Unavailable)        Checkout.jsx:62
Success: ▶['A szolgáltatás jelenleg nem elérhető! Próbálja meg később!']   Checkout.jsx:72
```

12. ábra Hiba frontend oldalon könyvkölcsönzés során

A szerver elutasította a „POST” kérést, azonban többször is ellenőrzésre került POSTMAN-nel (egy széles körben használt API tesztelő- és fejlesztő program) és a tesztelés során minden rendben zajlott. Az adatbázisban megjelentek az új rendelési adatok, továbbá 200-as státuszkóddal és „Sikeres kölcsönzés!” üzenettel tért vissza a metódus. A backend oldalon kezdtük keresni a hibát mivel a frontend oldalon a konzolra kiírattuk a küldendő adatokat és nem véltük hibásnak.

```
▼ 0: {UserID: "1", BookID: 1, cartID: 263876, quantity: 1, price: "3500.00",…}
    BookID: 1
    UserID: "1"
    cartID: 263876
    end_of_borrowment: "2024-06-01"
    price: "3500.00"
    quantity: 1
    start_of_borrowment: "2024-05-02"
▼ 1: {UserID: "1", BookID: 5, cartID: 263876, quantity: 1, price: "3900.00",…}
    BookID: 5
    UserID: "1"
    cartID: 263876
    end_of_borrowment: "2024-06-01"
    price: "3900.00"
    quantity: 1
    start_of_borrowment: "2024-05-02"
```

13. ábra Konzolra kiíratott rendelés tartalma

A backend konzolja a „BookID nem lehet nulla értékű” üzenettel tért vissza. A borrowedbooks tábla szerkezetét vizsgálva körvonalazódott, hogy valóban, a „NOT NULL” megkötés került meghatározásra létrehozáskor. A hiba további elemzésével rájöttünk, hogy minden egyes érték NULL-ként adódik át (*11.ábra sql sora*), kivéve a userID értéke(1 kerül átadásra, azért nem NULL értékű, mert a sütikből származik). Hosszas kutakodást, próbálkozást követően rájöttünk, hogy a végpont-és a hozzá tartozó metódus kialakítása során követtük el a hibát, mivel az csak egy darab rendelés fogadására volt alkalmas (egyszerű json objektum átvételére), nem pedig egy tömbbe ágyazott objektumok feldolgozására. A beérkező összetett adatszerkezet iterálását egy for ciklus hozzáadásával megoldottuk, azonban továbbra sem sikerült a rendelés felvitele az adatbázisba, mivel a konzolon a „Duplicate entry(cartID) for key PRIMARY” üzenet jelent meg. A hibaüzenet arról ad tájékoztatást, hogy megpróbáltunk egy elsődleges kulccsal ellátott oszlophoz olyan értéket adni, ami nem egyedi. A kezdetekben úgy véltük, hogy a rendelési azonosító csak egyszer fog szerepelni az adatbázisban, azonban azzal a lehetőséggel, hogy több könyvet is lehet kölcsönözni egy rendelés során, a kezdeti adatbázisunk módosítását vonta maga után. A cartID megkötéseit el kellett vetnünk.
A fenti személyes tapasztalatom is alátámasztja azt a tényt, hogy a hiányos specifikáció megnehezíti az implementálás fázisát.

2.3.2. FRONTEND

Fejlesztői környezet beállítása Node.js-sel, React-tel, TailwindCSS-el, DaisyUI-val, és EmailJS-sel

Node.js és npm telepítése:
Töltsd le és telepítsd a Node.js-t a hivatalos oldalról, amely magában foglalja az npm csomagkezelőt is.

```perl
npx create-react-app my-react-app
cd my-react-app
```

Ellenőrizd a telepítést a node -v és npm -v parancsokkal a terminálban.

```bash
node -v
npm -v
```

React projekt létrehozása

Használd a Create React App eszközt a React projekt alapjainak létrehozásához:

```bash
npx create-react-app my-react-app
```

- Várj, amíg a parancs befejezi a projekt generálását és a függőségek telepítését.
- Lépj be az új projekt könyvtárába:

```bash
cd my-react-app
```

Könyvtárak telepítése:

```bash
npm install react-router-dom
npm install react-toastify
```

TailwindCSS és DaisyUI telepítése:
Telepítsd a TailwindCSS-t és inicializáld a konfigurációs fájlokat:

- Telepítsd a TailwindCSS-t a projektbe:

```bash
npm install -D tailwindcss@latest postcss@latest autoprefixer@latest
```

- Inicializáld a Tailwind konfigurációját:

```bash
npx tailwindcss init -p
```

Ez létrehozza a `tailwind.config.js` és a `postcss.config.js` fájlokat a projekt gyökérkönyvtárában.

- Állítsd be a TailwindCSS-t, hogy a projekt összes stílusát kezelje. Nyisd meg az `src/index.css` fájlt, és illeszd be az alábbi direktívákat:

```css
@tailwind base;
@tailwind components;
@tailwind utilities;
```


- Telepítsd a TailwindCSS-t a projektbe:

```bash
npm install -D tailwindcss@latest postcss@latest autoprefixer@latest
```

- Inicializáld a Tailwind konfigurációját:

```bash
npx tailwindcss init -p
```

Ez létrehozza a `tailwind.config.js` és a `postcss.config.js` fájlokat a projekt gyökérkönyvtárában.

- Állítsd be a TailwindCSS-t, hogy a projekt összes stílusát kezelje. Nyisd meg az `src/index.css` fájlt, és illeszd be az alábbi direktívákat:

```css
@tailwind base;
@tailwind components;
@tailwind utilities;
```

```bash
npm install daisyui
```

- Add hozzá a DaisyUI-t a `tailwind.config.js`-hez a `plugins` részhez:

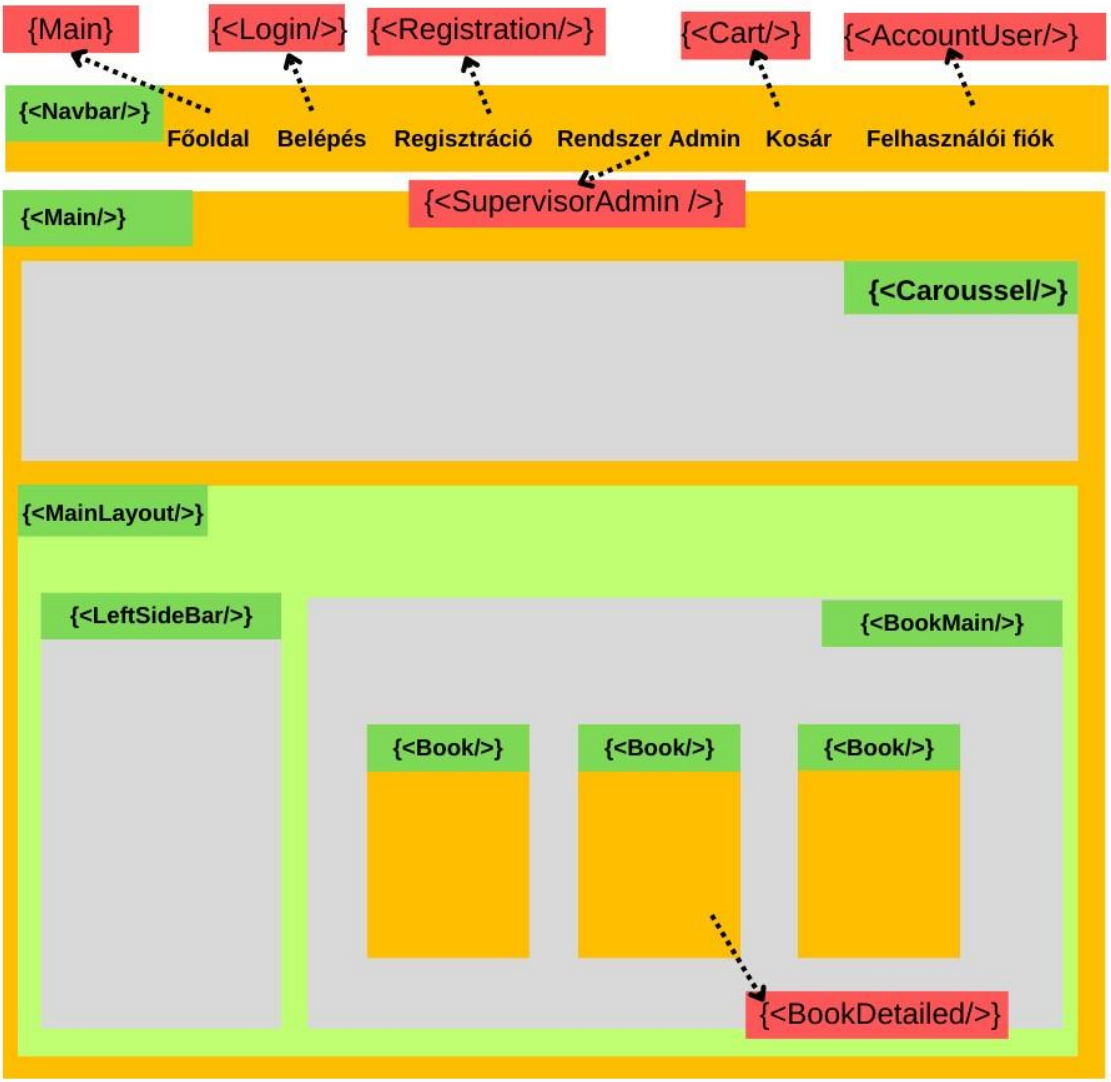
```javascript
module.exports = {
  // other settings
  plugins: [
    require('daisyui'),
  ],
}
```

EmailJS integrálása:

Telepítsd az EmailJS-t:

```bash
npm install emailjs-com
```

A lenti ábrának megfelelően a komponenseket külön-külön bemutatjuk, hogy mi miért felelős a frontend részen.



2.3.2.1 Navbar Komponens Leírása

A fent bemutatott Navbar komponens a React alkalmazás navigációs sávját kezeli, és több fontos feladatot is ellát:

- Állapotok és Hookok: Az állapotok (email, isAdmin, UserID) kezelésére szolgál. Ezek az állapotok tárolják a felhasználó alapvető adatait, és az alkalmazás bizonyos részeinek megjelenítését szabályozzák. useEffect: Az alkalmazás oldalváltásainak észlelésére használt hook, amely a localStorage-ból olvassa ki az isAdmin, email, és UserID értékeit, így biztosítva, hogy a felhasználó állapota friss maradjon a navigáció során.
- Navigációs Funkciók: A Link komponenseket használva biztosítja a navigációt az alkalmazáson belül. A felhasználó különböző oldalakra (pl. főoldal, adminisztrációs felület, kosár) léphet át egyszerűen. Feltételes Megjelenítés: A komponens különböző elemeket jelenít meg a felhasználó

jogosultságainak függvényében. Ha a felhasználó admin, akkor megjelenik az Rendszer Admin link, míg egy sima felhasználónak csak a kosár és egyéb alapvető opciók érhetőek el. Kijelentkezés: A handleLogout függvény eltávolítja a felhasználó adatait a localStorage-ból és a kezdőoldalra navigál, ezzel biztosítva a felhasználói munkamenet biztonságos lezárását.

- Interaktív Elemek: A felhasználói fiókhoz tartozó lefelé nyíló menü, amely a bejelentkezett felhasználó számára további opciókat kínál, mint például az Adataim oldalra való navigáció, amely a felhasználó személyes beállításait teszi elérhetővé. Reszponzív Kialakítás: A komponens reszponzív kialakítása biztosítja, hogy minden eszközön megfelelően jelenjen meg, melyet a CSS osztályok és stílusok, mint a flex, justify-between stb., segítségével ér el.
- Grafikus Megjelenés: A navigációs elemek animációkkal vannak ellátva (pl. nagyítás hover eseményre), amelyek javítják a felhasználói élményt, és vizuálisan is vonzóbbá teszik az interfészt.

2.3.2.2 BookMain, Book és a LeftSideBar komponensek leírása

BookMain Komponens

A BookMain komponens felelős a könyvek megjelenítéséért adott kategórián belül. Az állapotok és a router hookok segítségével dinamikusan kezeli a kategóriák és a könyvek adatait. Itt történik a könyvek adatbázisból történő lekérése és megjelenítése.

- useState: Az állapotkezelésért felelős, itt a 'books' állapot tartalmazza a lekért könyvek adatait.
- useParams: A 'categoryID' segítségével olvassa ki az aktuális kategória azonosítóját a URL-ből.
- useEffect: A komponens betöltődése vagy a 'categoryID' változása esetén fut le, frissíti a 'books' állapotot a szerverről lekért adatokkal.
- Dinamikus URL: A fetch hívásban a URL dinamikusan változik a 'categoryID' értékétől függően, így biztosítva, hogy a megfelelő kategóriájú könyveket kérje le.

Book Komponens

A Book komponens egyetlen könyv adatait jeleníti meg. Felelős a könyv címének, szerzőjének, árának és borítóképének megjelenítéséért, és lehetőséget ad a részletes adatok megtekintésére egy külön oldalon keresztül.

- Dekonstrukció: A komponens a 'book' prop-ból dekonstruálja ki a szükséges adatokat, mint például a 'BookID', 'title', 'author', és 'price'.
- Reszponzív Kialakítás: A könyv adatlapja reszponzívan van megformálva, megfelelően jelenik meg különböző kijelzőméretek és eszközökön.

Interaktivitás: A könyv kártyája interaktív, hover eseményre árnyékkal emeli ki a könyvet, és a 'Megnézem' gombra kattintva átirányít a könyv részletes adatlapjára

LeftSideBar Komponens Leírása

A LeftSideBar komponens egy oldalsáv, amely a könyvkategóriákat jeleníti meg. A felhasználók a kategóriák között válogathatnak, és a kiválasztott kategória alapján a rendszer megjeleníti a hozzá tartozó könyveket. Ezáltal könnyebbé teszi a könyvek böngészését.

- useState: A 'categories' állapot tartalmazza a lekérdezett kategória adatokat, a 'selectCategory' pedig az aktuálisan kiválasztott kategória azonosítóját.
- useEffect: Az oldal betöltődésekor fut le, és lekéri a kategóriák listáját egy API hívással.
- fetch: A kategóriák lekérdezéséért felelős, kezeli a sikeres lekérdezéseket és a hibákat egyaránt.
- handleCategoryClick: A kategória kiválasztásának logikáját kezeli. Frissíti a 'selectCategory' állapotot a kiválasztott kategória azonosítójával, így elősegítve a további logikai lépéseket.

2.3.2.3 DetailedBook komponens leírása

A DetailedBook komponens egy specifikus könyv részletes adatait jeleníti meg. Ez a komponens React technológiát használ állapotkezelésre és az URL paraméterek kezelésére, amelyek segítségével dinamikusan kérdezi le a kiválasztott könyv adatait.

- useState: Kezeli a 'selectedBook' állapotot, amely tárolja a kiválasztott könyv adatait.
- useParams: A 'BookID' értéket nyeri ki az URL-ből, amely az adott könyv azonosítója.
- useEffect: Az adatlekérésért felelős. A 'BookID' változásakor újra lefut, és frissíti a 'selectedBook' állapotot a szerverről lekért adatokkal.
- fetch: A könyv adatait lekérő API hívást végez, amely hibakezeléssel is rendelkezik.
- handleCart: A kosárhoz adás logikáját kezeli. Ellenőrzi, hogy a kiválasztott könyv már szerepel-e a kosárban, és alkalmazza a megfelelő logikát (új hozzáadása vagy figyelmeztetés).

Kölcsönzési díj: 3900.00 Ft

Kosárba

Jamie Oliver 5

Szerző: Jamie Oliver

Penguin Books | 2014 | angol nyelvű | keménytáblás kötésű | 320 oldal

Kölcsönzési díj:3900.00 Ft **Státusz:** Kölcsönözhető

Az Jamie Oliver 5 egy egyszerű és ízletes recepteket tartalmazó szakácskönyv, amely az egészséges és gyorsan elkészíthető ételek iránt érdeklődőknek szól. Minden recept csak öt fő összetevőt tartalmaz, és könnyen elkészíthető.

Kiadó	Penguin Books
Kiadás éve:	2014
Nyelv	angol
Borító	keménytáblás
Lapok száma	320 oldal
ISBN	9780718157807
Árukód	892371 / 743982

-

2.3.2.4 Caroussel és Main komponensek leírása

A Main komponens a weboldal főoldalának alapvető elrendezését és funkcióit biztosítja, míg a Caroussel komponens egy vizuálisan vonzó, interaktív képváltó (carousel) elemet tartalmaz, amely különböző kategóriákat és témákat mutat be a felhasználók számára.

Main Komponens

A Main komponens az alkalmazás központi oldalaként funkcionál, ahol a Caroussel és a MainLayout komponensek kerülnek megjelenítésre. Ezek az elemek biztosítják, hogy a felhasználó közvetlen hozzáférést kapjon a weboldal különböző részeihez.

Caroussel Komponens

A Caroussel komponens a weboldal egyik kiemelt vizuális eleme, amely többféle médiaobjektumot is képes megjeleníteni dinamikusan. Az egyes kategóriákhoz tartozó képek interaktív linkekkel vannak ellátva, amelyek megkönnyítik a felhasználó navigációját és javítják a felhasználói élményt.

- Dinamikus Képváltó: A Caroussel több oszlopot és sorot tartalmaz, amelyek dinamikusan változnak a képernyő méretétől függően.
- Linkelt Kategóriák: Mindegyik kép egy link, amely egy adott kategóriához tartozó oldalra irányítja át a felhasználót.
- Animációs Hatások: A képek hover eseményre nagyobbak lesznek, amely vizuális visszajelzést ad a felhasználónak.
- Reszponzív Design: A komponens alkalmazkodik a különböző eszközökön és képernyőméretekhez, így biztosítva, hogy minden felhasználó számára megfelelően jelenjen meg.

2.3.2.5 Registration komponens leírása

A Registration komponens egy űrlapkezelő komponens a React-ben, amely lehetővé teszi a felhasználók számára, hogy regisztráljanak az alkalmazásba. Az alábbiakban a kód fontos részei kerülnek bemutatásra:

Komponens Struktúra

A komponens több állapotváltozót használ (`useState`), amelyek az űrlapmezők értékeit tárolják: `fullName`, `phone`, `email`, `pass`, `passAgain`, `zip`, `city`, `address`. Ezek az adatok a felhasználó által történő megadás során frissülnek.

Az űrlap adatait egy aszinkron `register` függvény küldi el, amikor a felhasználó a 'Regisztráció' gombra kattint. Ez a függvény összeállítja a regisztrációs objektumot (`regObj`), amely tartalmazza a felhasználó által megadott összes adatot, és egy POST hívást indít az `/regisztracio` végpontra.

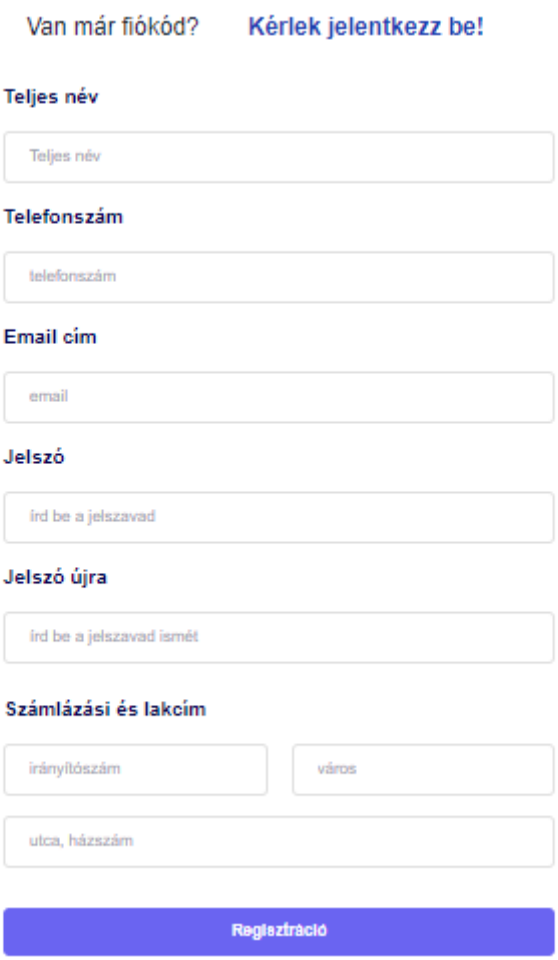
Aszinkron Kommunikáció

A `register` függvény az alábbi módon van implementálva:

```javascript
const register = async () => {
  const regObj = {
    fullName: fullName,
    phone: phone,
    email: email,
    pass: pass,
    passAgain: passAgain,
    zip: zip,
    city: city,
    address: address
  };
  try {
    const response = await fetch('http://localhost:3001/regisztracio', {
      method: 'POST',
      body: JSON.stringify(regObj),
      headers: { 'Content-type': 'application/json' }
    });
    const json = await response.json();
    if (response.ok) {
      toast.success('Sikeres regisztráció');
      navigate(`/belepes`);
    } else {
      json.forEach((msg) => toast.error(msg));
    }
  } catch (err) {
    toast.error('Hiba történt a regisztráció során.');
    console.error(err);
  }
};
```

Felhasználói Interfész

A felhasználói felület HTML és CSS segítségével van kialakítva, ahol a form elemek (`input`, `button`) a React komponens állapotváltozóival vannak összekötve, így biztosítva a reaktív adatfrissítést és a felhasználói interakciók kezelését.



2.3.2.6 Login, ForgottPassword és EMAILJS

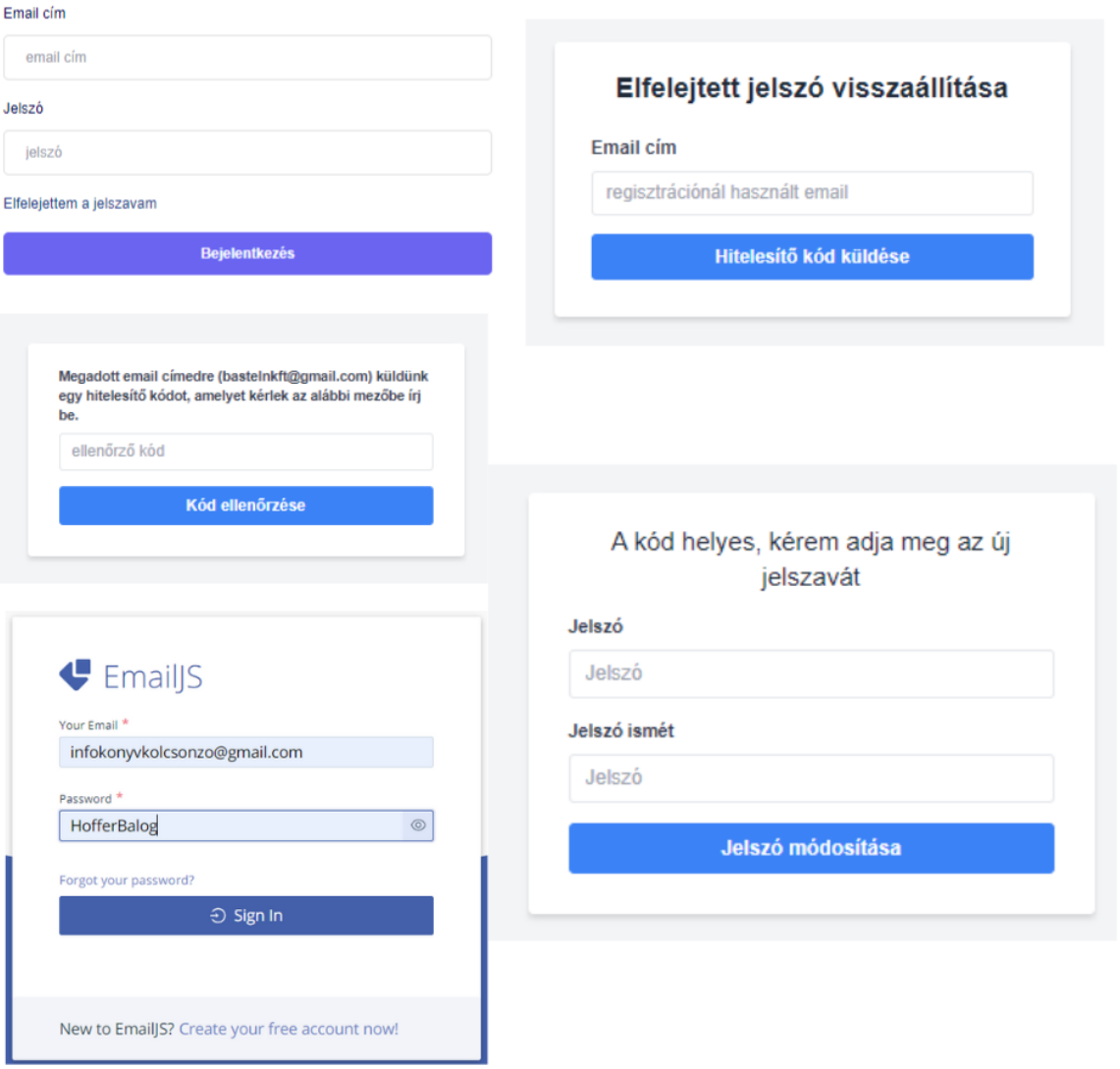
Login komponens leírása

A Bejelentkezés komponens Reactban készült, és számos állapotot (`useState`), navigációs (`useNavigate`), és segédfüggvényeket használ a működéséhez.

1. **useState Hookok**:
- `email`: Tárolja a felhasználó által megadott e-mail címet.
- `pass`: Tárolja a felhasználó által megadott jelszót.
- `errMessages`: Tárolja a hibák listáját, amelyek a bejelentkezés során keletkezhetnek.
- `displayMb`: Egy boolean, amely irányítja, hogy a hibaüzenetek megjelenjenek-e a `MessageBox` komponensben.

2. **useNavigate Hook**: A `navigate` függvényt használják a felhasználó átirányítására sikeres bejelentkezés után.

3. **signIn Függvény**: Ez a függvény végzi a tényleges adatküldést a szerver felé. A `fetch` API-t használja, és kezeli az aszinkron válaszokat a szerverről. Sikeres válasz esetén beállítja a szükséges adatokat a `localStorage`-ba, és irányítja a felhasználót a megfelelő oldalra.



ForgottenPassword Komponens leírása

Az Elfelejtett Jelszó komponens lehetővé teszi a felhasználóknak, hogy visszaállítsák a jelszavukat egy ellenőrző kód segítségével, amit e-mailben kapnak meg.

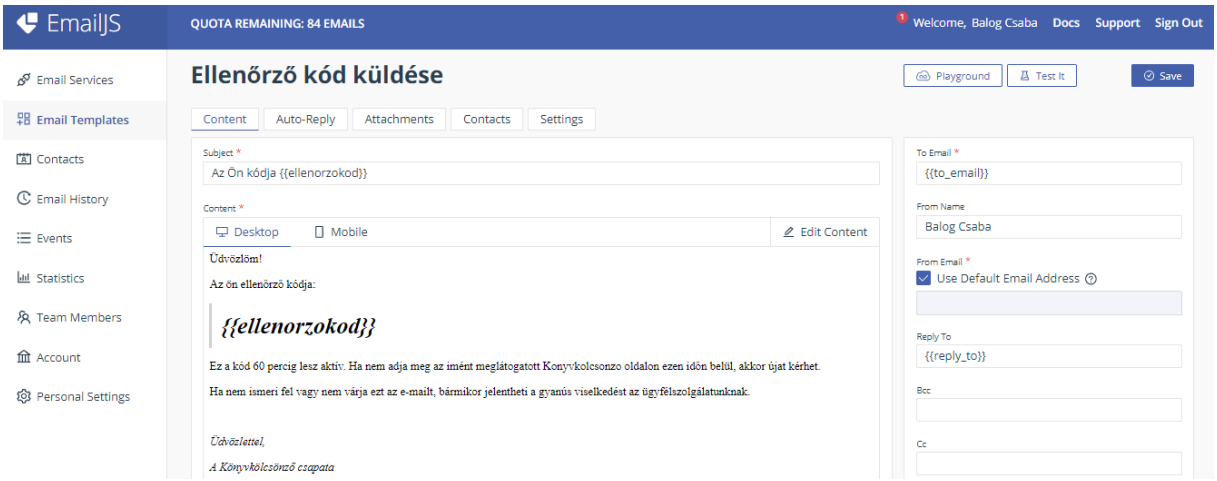
1. **useState Hookok**:
- `email`: Tárolja a felhasználó által a formon megadott e-mail címet.
- A függvény a `handleSubmit` eseménykezelővel aktiválódik, amely kezeli az űrlap benyújtását.

2. **E-mail Küldés**: A `handleSendEmail` függvény az `emailjs` könyvtárat használja az e-mail küldésére. A függvény egy előre definiált sablont tölt ki a kapott adatokkal, beleértve az ellenőrző kódot is.

3. **Navigáció és Ellenőrzés**: A `handleSubmit` függvény generál egy véletlenszerű ellenőrző kódot, amit a `localStorage`-ba ment el, majd aktiválja az e-mail küldés folyamatát. Sikeres e-mail küldés után átirányítja a felhasználót a jelszó módosító oldalra.

```
// Email küldési logika, ami megkapja a verificationCode-ot paraméterként
const handleSendEmail = (verificationCode) => {
    const templateParams = {
        to_email: email,
        from_email: 'infokonykolcsonzo@gmail.com',
        subject: 'Ellenőrző kód küldése',
        ellenorzokod: verificationCode,
    };

    emailjs.send('service_mipbx9a', 'template_z4cxcwt', templateParams, 't2P0egLC6EXkHDhyE')
        .then((response) => {
            console.log('Email sent successfully:', response);
            toast.success('Sikeres email küldés!')
        })
        .catch((error) => {
            console.error('Hiba az email küldés során:', error);
            toast.error(error.message)
        });
};
```

EmailJS felülete:



2.3.2.7 SupervisorAdmin és onnan elérhető komponensek

Admin (isAdmin értéke ebben az esetbe 1) jogosultsággal érhető el ez a komponens, ahol az adminisztrálást végezzük.

Könyv Adminisztrálás

BookAdmin Komponens

A BookAdmin komponens egy táblázat sorát kezeli, ahol a könyvek adatai (cím, szerző) és a kezelési lehetőségek (szerkesztés, törlés) jelennek meg. A komponens lehetőséget nyújt a könyvek szerkesztésére és törlésére, ahol a törlés megerősítéséhez egy párbeszédablakot használ.

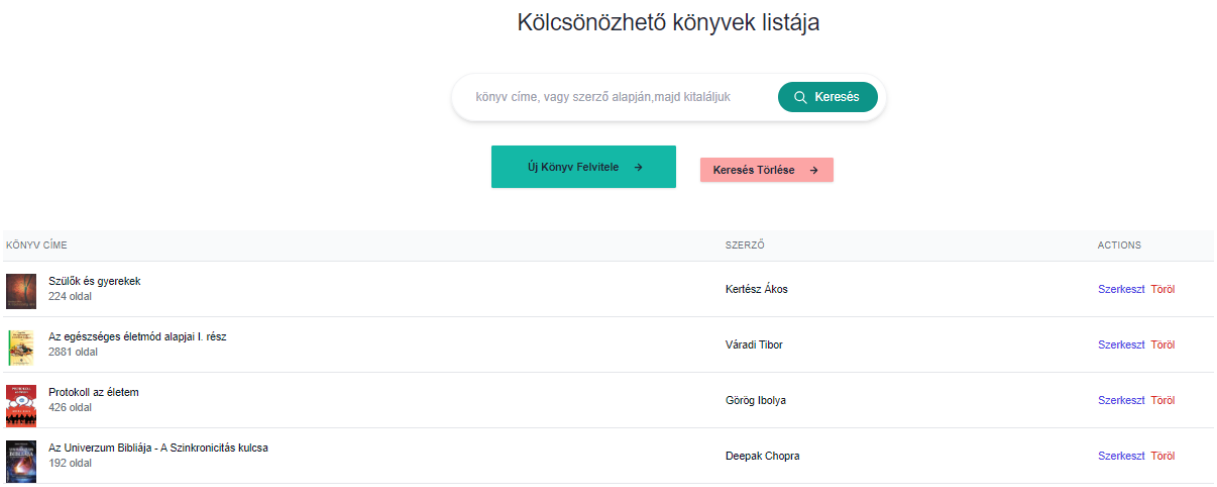
A törlés funkciót a `torles` függvény valósítja meg, amely a `fetch` API-t használja a könyv adatbázisból történő eltávolításához. Sikeres törlés esetén egy sikerüzenet jelenik meg, hibás törlés esetén pedig hibaüzenetet kapunk.

BookList Komponens

A BookList komponens a könyvek listáját jeleníti meg, és lehetőséget biztosít a keresésre és új könyvek felvitelére. Egy dinamikus keresőmező segítségével a felhasználók megadhatják a keresett könyvek címét vagy szerzőjét.

Az állapotkezelés két fő részből áll: a `books` állapot, amely a lekérdezett könyvek listáját tartalmazza, és a `letter` állapot, amely a keresőmezőben bevitt karaktert tárolja. A komponens `useEffect` hookokat használ az adatok automatikus lekérése és frissítése érdekében az API-tól.

Az `inputLetterChange` eseménykezelő frissíti a `letter` állapotot, és a `handleClick` gombra kattintva az aktuális keresési feltétel alapján újra lekéri a könyvek listáját. A `refreshBooks` funkció lehetővé teszi a könyvlista frissítését akár manuálisan is.



BookFormModify komponens leírása:

A BookFormModify komponens egy űrlapot biztosít, amellyel a felhasználók könyveket vehetnek fel vagy módosíthatnak a könyvtári rendszer adatbázisában. Ez a komponens React használatával készült, és több különböző React Hookot és funkciót használ az állapotkezelésre és az eseménykezelésre.

Főbb Funkciók és Működés

1. **Állapotkezelés (`useState`):** A komponens `useState` hookokat használ a könyvadatok (mint cím, szerző, kiadó, stb.) kezelésére. Ezek az adatok egy objektumban (`formData`) tárolódnak, amit a felhasználó módosíthat.

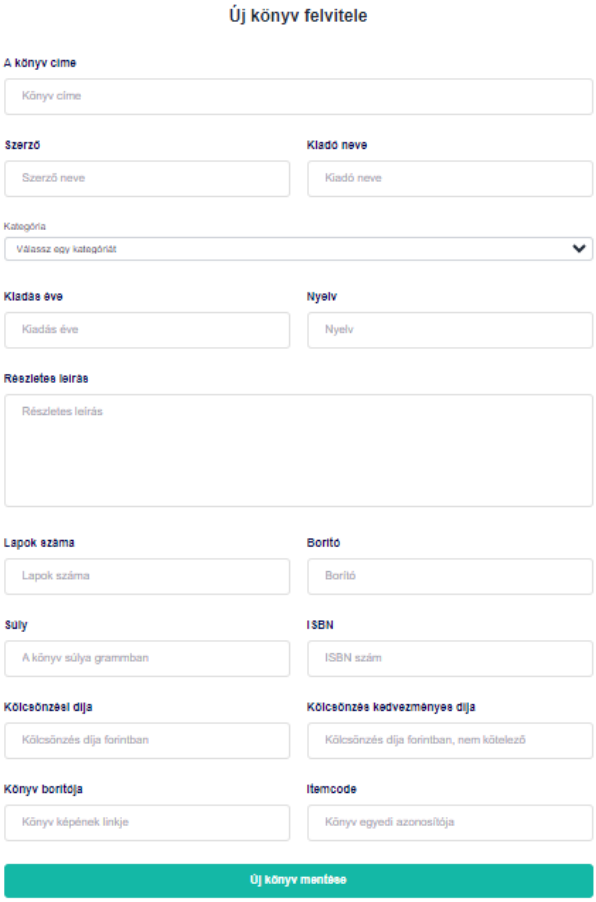
2. **Adatbetöltés (`useEffect`):** Amikor a komponens betöltődik, egy `useEffect` hook automatikusan lekéri a megadott könyv adatait az API-tól, ha a `BookID` paraméter meg van adva. Ez a folyamat az adatokat a `formData` objektumba tölti.

3. **Form adatkezelés:** A form minden mezője a `formData` állapot egy-egy attribútumát kezeli, amelyek módosításakor a `writeData` eseménykezelő frissíti az állapotot. Ez a dinamikus állapotkezelés lehetővé teszi, hogy a felhasználói bevitel azonnal megjelenjen a felületen.

4. **Űrlap beküldése (`onSubmit`):** Az űrlap beküldésekor a `onSubmit` eseménykezelő hívódik meg, amely egy aszinkron API kérést indít az adatok mentésére vagy módosítására. Sikeres művelet esetén a felhasználó visszairányításra kerül a könyvadminisztrációs oldalra.

Biztonsági és Hálózati Kommunikáció

A biztonságos kommunikáció érdekében az API hívásoknál használt `credentials: 'include'` opció biztosítja, hogy a munkamenet-azonosítók és egyéb biztonsági tokenek megfelelően kezelődjenek. A hálózati kommunikáció során keletkező hibák kezelésére hibakezelési rutinok vannak beépítve, amelyek informálják a felhasználót a problémákról.



Felhasználó Adminsztrálás

UserList Komponens leírása

A UserList komponens egy felhasználókezelő rendszer részeként funkcionál, ahol a felhasználók listáját kezeli és különféle műveleteket tesz lehetővé, mint a keresés, frissítés és felhasználói adatok megjelenítése.

Ebben a komponensben több állapot is van kezelve, beleértve a `users`, `letter`, `inputLetter`, és a `searchPerformed`. Ezek az állapotok segítenek a keresési funkciók kezelésében és a felhasználói felület dinamikus frissítésében.

Az `inputLetterChange`, `handleClick`, és `handlecClick2` függvényeket használjuk az állapotok frissítésére és az API hívások kezdeményezésére. Ezek a függvények biztosítják, hogy a keresési eredmények mindig naprakészek legyenek.

A `fetch` hívások segítségével kommunikálunk a szerverrel, ahol a felhasználók adatait JSON formátumban kérjük le vagy frissítjük.

A `useEffect` hookok segítségével automatikusan frissítjük a felhasználók listáját, valamint reagálunk a keresési szűrő állapot változásaira.

A komponens felhasználói felületet biztosít a felhasználók táblázatos megjelenítésére, keresésre és új adatok felvitelére. A dinamikus keresőmező és a keresés törlése funkció a felhasználók hatékony kezelését segíti.

User Komponens leírása

A `User` komponens egy felhasználói sor kezelésére szolgál egy táblázatban, ahol az egyes felhasználók adatait jeleníti meg, és lehetőséget biztosít a felhasználók törlésére.
Komponens részletei és funkciók:
Állapot és Prop kezelés:
- A komponens a `props` segítségével kapja meg a felhasználói adatokat (`props.user`), amely tartalmazza az egyes attribútumokat, mint a felhasználó neve, email címe, címe, és adminisztátori státusza.
- A `props` tartalmazza a `onDelete` függvényt is, ami akkor hívódik meg, ha a felhasználó törlése sikeresen megtörtént, így frissítve a felhasználói lista megjelenítését.

Törlés kezelése:
- A `torles` függvény végzi a felhasználó törlését. Ez egy aszinkron HTTP kérés a szerver felé (`DELETE` metódussal), amely a megadott felhasználó azonosítóval (`UserID`) történik.
- Sikeres törlés esetén a `toast.success()` metódus hívódik meg, ami sikeres üzenetet jelenít meg, hiba esetén pedig a `toast.error()` metódus figyelmeztet a problémára.

Modal (párbeszédpanel) kezelése:
- A törlés gombra kattintva egy modal (párbeszédpanel) jelenik meg, amely megerősítést kér a felhasználótól a törléshez.
- A modal-ban található "Ok" gombra kattintva hívódik meg a `torles` függvény, míg a "Mégsem" gomb bezárja a modal-t.

Felhasználói felület:
- A felhasználói adatokat egy táblázatsorban (`<tr>`) jeleníti meg.
- A felhasználó képét, nevét, email címét, lakcímét, és státuszát jeleníti meg, illetve egy törlés gombot, amely aktiválja a modal-t a törlés megerősítéséhez.

Stílus és elrendezés
- A komponens Bootstrap és Tailwind CSS stílusokat használ a vizuális megjelenés és a felhasználói interakciók kezelésére.
- A komponens kódja jól strukturált, és a CSS osztályok segítségével könnyen testreszabható.

Ez a komponens egy hatékony módja a felhasználók kezelésének egy adminisztrációs felületen belül, lehetővé téve az azonnali adatkezelést és interaktív visszajelzést a felhasználói műveletekről.

Kölcsönzések adminisztrálása

BorrowList komponens leírása

A BorrowList egy React funkcionális komponens, amely kezeli egy könyvtár kölcsönzött könyveinek listáját.

Állapot Hook-ok:
borrows: Egy tömb, amely a kölcsönzött könyvek listáját tárolja.

letter: Egy sztring állapot, a felhasználó által beírt szűrési értéket tárol.

lastCartID: Egy állapot, amely az utolsó elérési kosár azonosítóját tárolhatja.

search: Egy logikai állapot, amely a keresési mód váltását szabályozza.

useEffect Hook:

A komponens betöltésekor meghívja a loadBorrows funkciót a kölcsönzött könyvek listájának betöltésére.

Függvények:

loadBorrows: Lekéri a kölcsönzött könyvek listáját és frissíti az állapotot.

updateBorrows: Frissíti a kölcsönzött könyvek listáját.

inputLetterChange: Kezeli a szöveges beviteli mező változását.

handleClick és handlecClick2: Kezelik a keresési funkciókat és állapot változásokat.

Borrow Komponens leírása:

Borrow egy funkcionális React komponens, amely paraméterként kap egy props objektumot, ami tartalmazza a szülő komponenstől kapott adatokat és funkciókat.
Állapotkezelés:
formData: Egy állapot, amely a form adatokat tartalmazza. Kezdetben egy objektum, ami tartalmazza a "bringedBack" kulcsot "1" értékkel.
Funkciók:
torles: Ez a funkció egy DELETE kérést küld a szervernek a cartID alapján, ami törli a megadott kölcsönzést. Sikeres törlés esetén egy sikerüzenetet jelenít meg a toast.success segítségével, hiba esetén pedig hibaüzenetet toast.error-rel.
lezaras: Ez a funkció egy PUT kérést küld a szervernek, ami frissíti a kölcsönzés állapotát az adott cartID alapján. Itt is hasonlóan kezeli a sikerességet és a hibákat, mint a törlés funkció.
Renderelés:
A komponens egy táblázatsor (<tr>) elemet renderel, amelyen belül több cella (<td>) található. Ezek a cellák tartalmazzák a kölcsönzés adatait, mint például a kölcsönzés azonosítója, kezdő dátuma, a kölcsönző neve, a kölcsönzés tárgya, és a kölcsönzés végének dátuma.
Kondicionális renderelést használ a "Kölcsönözve" és "Visszahozva, lezárt" állapotok megjelenítésére. Tartalmaz interaktív elemeket is, mint a "Lezárás" és "Töröl" gombok, amelyek modális párbeszédpaneleket nyitnak meg a kölcsönzés lezárásához vagy törléséhez.

2.3.2.8 Cart és Checkout komponensek

Cart Komponens

Komponens Magyarázata

Importálás és Komponens Definíció:
- React, useEffect, useState importálása szükséges az állapotkezeléshez és az életciklus-metódusok kezeléséhez.
- Link és useNavigate a react-router-dom-ból a navigáció kezelésére.

Állapotok:
- navigate: A navigációs funkció használata a programozott navigációhoz.
- total, totalBooks, totalBooksPrice: Állapotok, amelyek a kosár teljes értékét, a könyvek számát és azok teljes árát tárolják.
- carts: A localStorage-ból betöltött kosár elemeit tároló állapot.
- email, UserID: Felhasználói azonosító és email állapotok.

Funkciók és Metódusok:
- useEffect: Kosár változásainak figyelése és a teljes ár újraszámítása.
- removeProduct: Eltávolít egy terméket a kosárból.
- handleCheckout: Végrehajtja a kosár ellenőrzését, beleértve az egyes termékek adatainak összegyűjtését és a vásárlási folyamat előkészítését.
- handleClearCart: Törli a kosár tartalmát a localStorage-ből.

Renderelés:

- A kosár üres állapotát egy üzenettel kezeli.
- Dinamikusan jeleníti meg a kosár tartalmát, beleértve a termékek képeit, mennyiségét és árát.
- Navigációs linkeket biztosít a vásárlás folytatásához és a kosár ellenőrzéséhez.
- Védelmi mechanizmust tartalmaz, hogy a vásárlási folyamat csak bejelentkezett felhasználó számára elérhető legyen.

Checkout komponens

Állapotok és Hook-ok
- formObj: Tárolja a vásárlási adatokat, amik a localStorage-ból vannak betöltve.
- userEmail: A felhasználó e-mail címét tárolja, szintén a localStorage-ból olvasva.
- A useEffect hook inicializálja ezeket az adatokat a komponens betöltésekor.

Eseménykezelők és Funkciók:
- handleSendEmail: Elküld egy e-mailt a felhasználónak a kölcsönzés részleteivel. Az emailjs könyvtárat használja az e-mail küldéshez.
- sendOrderDetails: Elküldi a kölcsönzési adatokat egy szervernek POST kéréssel.
- emptyLocalStorage: Törli a localStorage-ban tárolt adatokat.
- end_of_order: Összekapcsolja a fent említett funkciókat, hogy egy teljes vásárlási folyamatot kezeljen.

Renderelés:
- A felhasználót köszöntő üzenetet jelenít meg, és tájékoztatja a kölcsönzési folyamatról.
- Tartalmaz egy gombot, ami aktiválja az end_of_order funkciót, ezzel véglegesítve a kölcsönzést.
- Az oldalon egy kép is megjelenik, ami a kölcsönzés témájához kapcsolódik.

2.4. Tesztdokumentáció

A webapplikációnk tesztelését egységekre bontva, mind backend, mind frontend oldalon elvégeztük. Olyan eseteket igyekeztünk kiválasztani a programunk tesztelése során, ahol az admin jogosultság feltétel és ott ahol nem („sima felhasználó”).

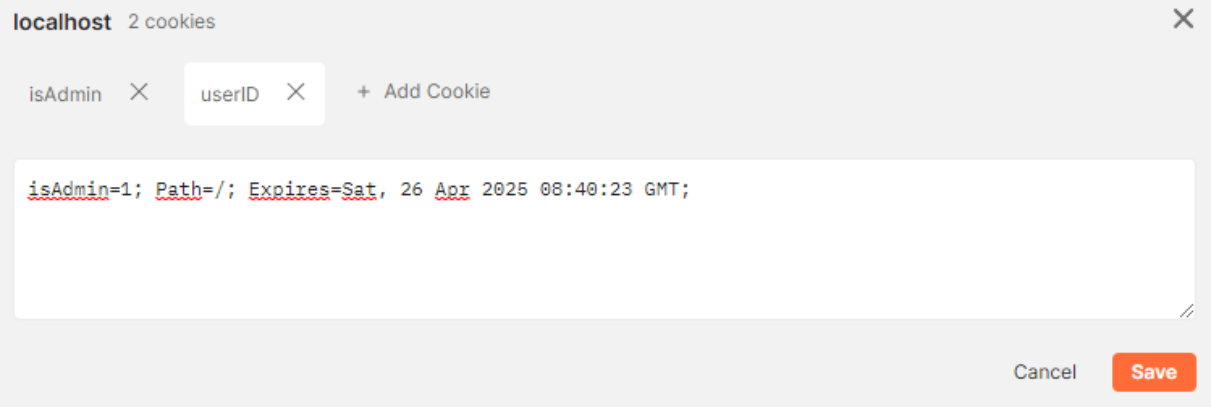
2.4.1. Backend oldalon elvégzett teszt

A teszt eszközéül a Postman programot választottuk, amely széles körben használt API (Application Programming Interface) tesztelő- és fejlesztő eszköz. Lehetővé teszi, hogy különböző metódusú („POST”,”PUT”, „DELETE” stb.) HTTP kéréseket küldjünk végpontokra és annak sikeréről, kudarcáról információt kapjunk vissza.

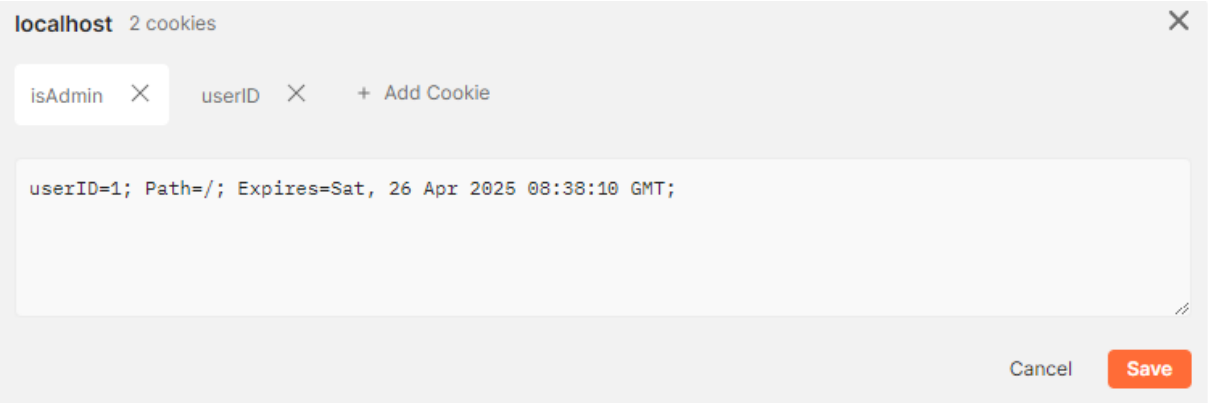
2.4.1.1. Admin jogosultsággal rendelkező felhasználói tevékenység tesztelése

Jelen pontban a felhasználó törlése funkció kerül megvizsgálásra. A művelet végrehajtásához a http://localhost:3001/admin/users/? végpontot kell használnunk. A kérdőjel helyére az adatbázisból kikeresett userID-t kell helyettesítenünk. A beérkező http metódus DELETE, ezért ennek beállítását el kell végezzük.

- Első teszteset: A művelet elvégzéséhez rendszer adminisztrációs jogosultság megléte szükséges, ezért első lépésként végezzük úgy el a tesztet, hogy a **felhasználó admin** (isAdmin értéke 1-es). Az utóbbi eléréséhez be kell állítani a cookie-kat. A userID süti beállítása is szükséges, ez vizsgálja a belépés tényét.

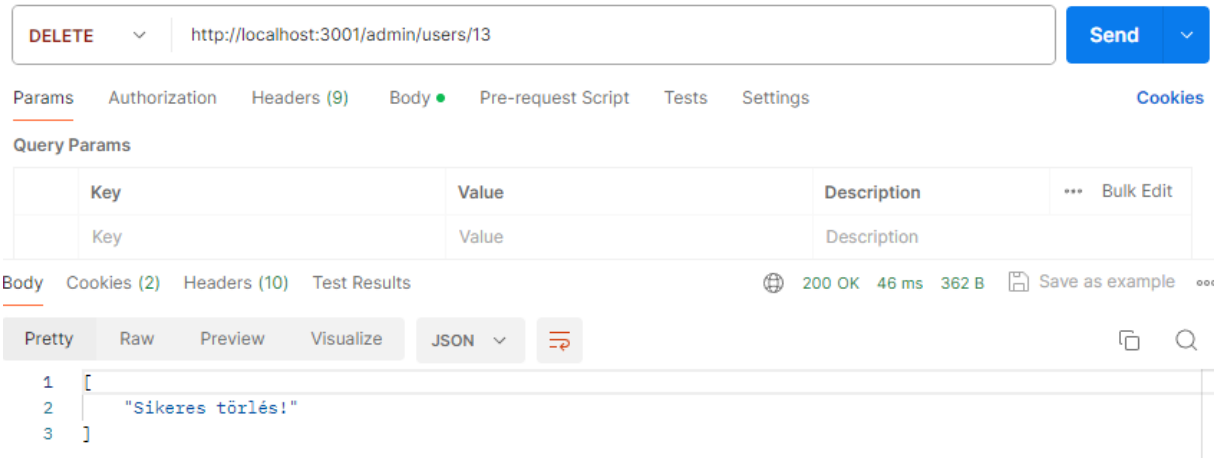


14. ábra Postman isAdmin cookie érték beállítása 1-re



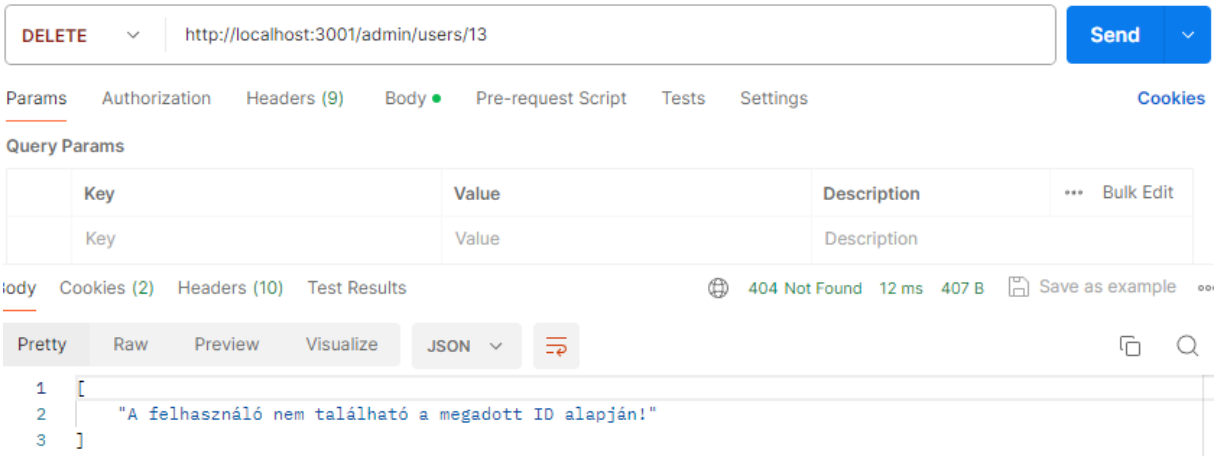
15. ábra Postman userID cookie érték beállítása

Amennyiben a felhasználó rendszergazdai jogosultsággal rendelkezik és az adatbázisban szereplő, **valós userID**-t ad meg paraméterként (a lenti példában mi a 13-ast választottuk), abban az esetben „***Sikeres törlés***!” üzenettel és ***200 OK*** státuszkóddal kell visszatérnie.



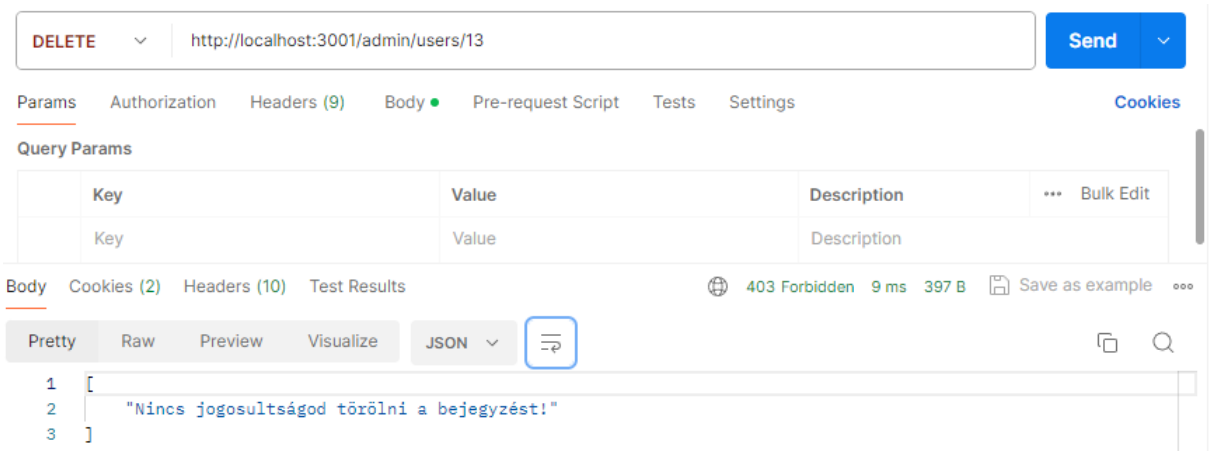
14. ábra Törlés tesztelése POSTMAN-NEL. Admin jogosultsággal, létező felhasználóazonosítóval.

- Második teszteset: A sikeres törlést követően vegyük alapul, hogy a **rendszergazdai jogosultság megmarad**, azonban olyan felhasználó azonosító iránt irányul a törlési kísérlet (meghagytuk az előző tesztesetben kitörölt 13-as azonosítójú felhasználót), **ami nem szerepel az adatbázisban**. Jelen esetben azt várnánk, hogy a visszatérő státuszkód ***404 Not Found*** , a hibaüzenet ***„A felhasználó nem található a megadott ID alapján!”.***



15. ábra Törlés tesztelése POSTMAN-NEL. Admin jogosultsággal, nem létező felhasználóazonosítóval.

- Harmadik teszteset: A **felhasználó ne kapjon rendszergazdai jogosultságot**, az isAdmin cookies értékét módosítsuk 0-ra. Azt várnánk, hogy a sütiben tárolt isAdmin=0 érték alapján a CheckPermission metódus visszatérési értéke false lesz, ezáltal a felületen a ***403 Forbidden*** státuszkód, „***Nincs jogosultságod törölni a bejegyzést!***” üzenet jelenik meg.



16. ábra Törlés tesztelése POSTMAN-NEL. Admin jogosultság nélkül

2.4.1.2..Admin jogosultsággal nem rendelkező felhasználói tevékenység tesztelése.

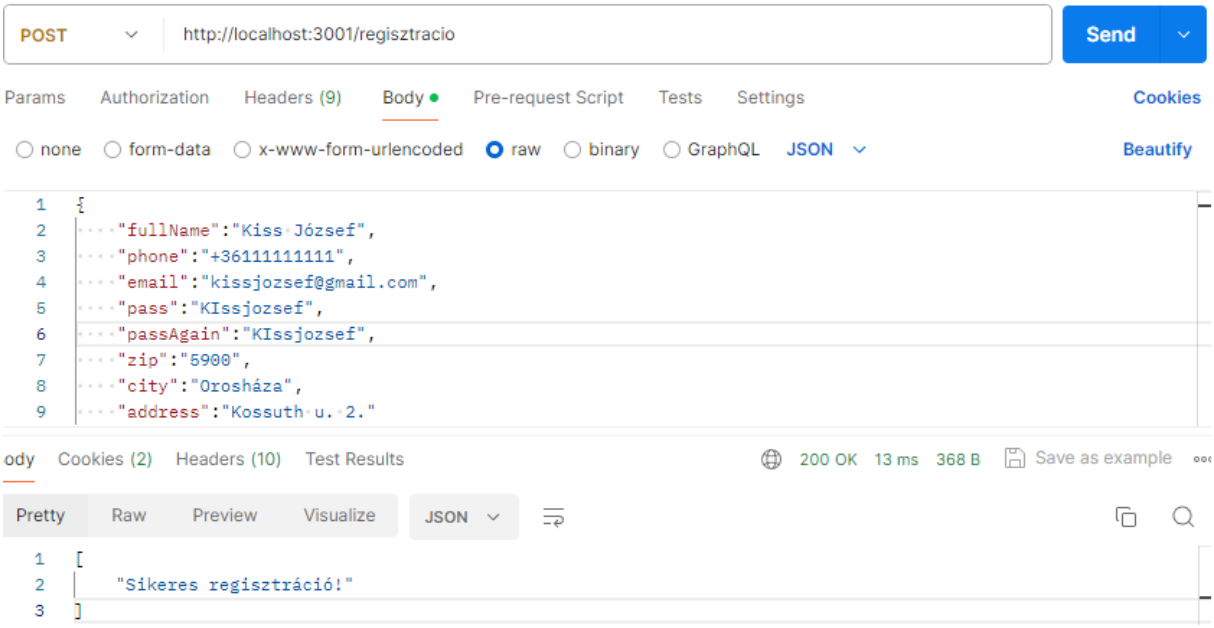
A törölni, módosítani nem tudó felhasználó esetében a regisztrációs tevékenységet vizsgáltuk meg. A tesztelés elkezdéséhez a http://localhost:3001/regisztracio végpontot szükséges beállítanunk. Mivel új adatok kerülnek elküldésre „POST" metódust kell kiválasztanunk. A regisztráció során a következő adatokat kell megadnunk:

- *Teljes név*
- *Telefonszám*
- *Jelszó*
- *Jelszó megerősítése*
- *Irányítószám*
- *Város*
- *Lakcím*

A fent részletezett űrlap elemek helyes formában történő kitöltését ellenőrzi az applikáció.

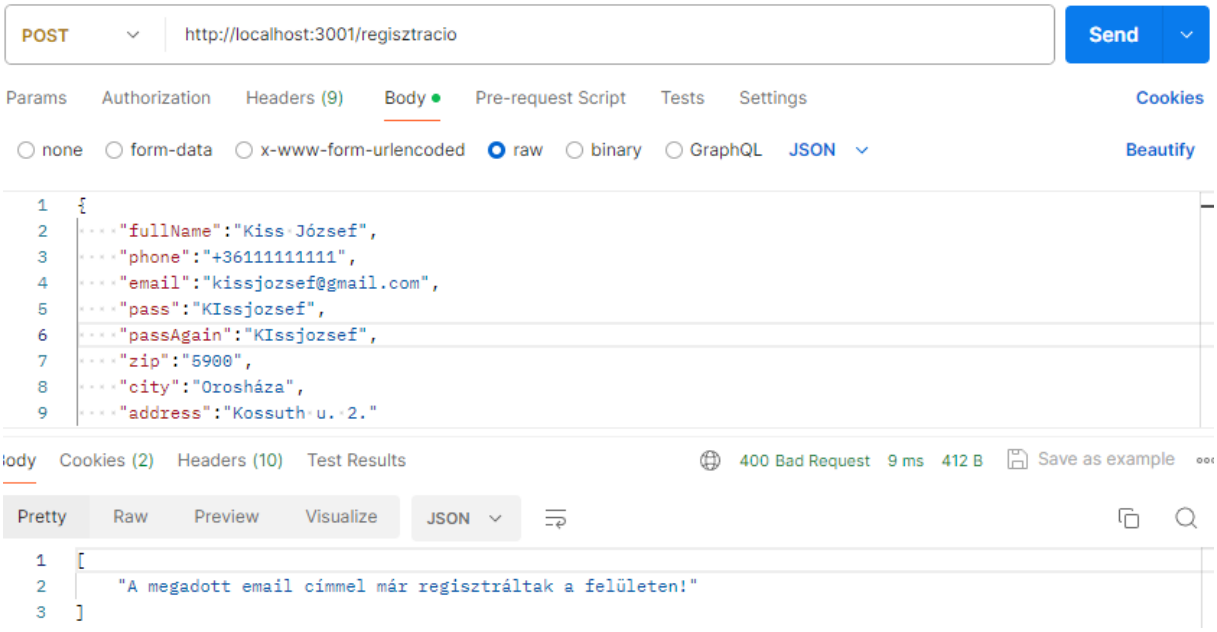
- Első teszteset: Vegyük a legegyszerűbb esetet, amikor a könyvkölcsönző személy sikeresen regisztrál. Postman programban úgy tudjuk szimulálni az adatok szerver felé történő továbbítását (pl. űrlapmezőkben), hogy a kérés **body** részébe kitöltjük az eljuttatni kívánt

információkat (az objektum kulcsainak elnevezése fontos!) Elvárnánk, hogy a visszatérő státuszkód ***200 OK***, „***Sikeres regisztráció!***” üzenettel.



17. ábra Sikeres regisztráció a konyvkolcsonzo adatbázisba

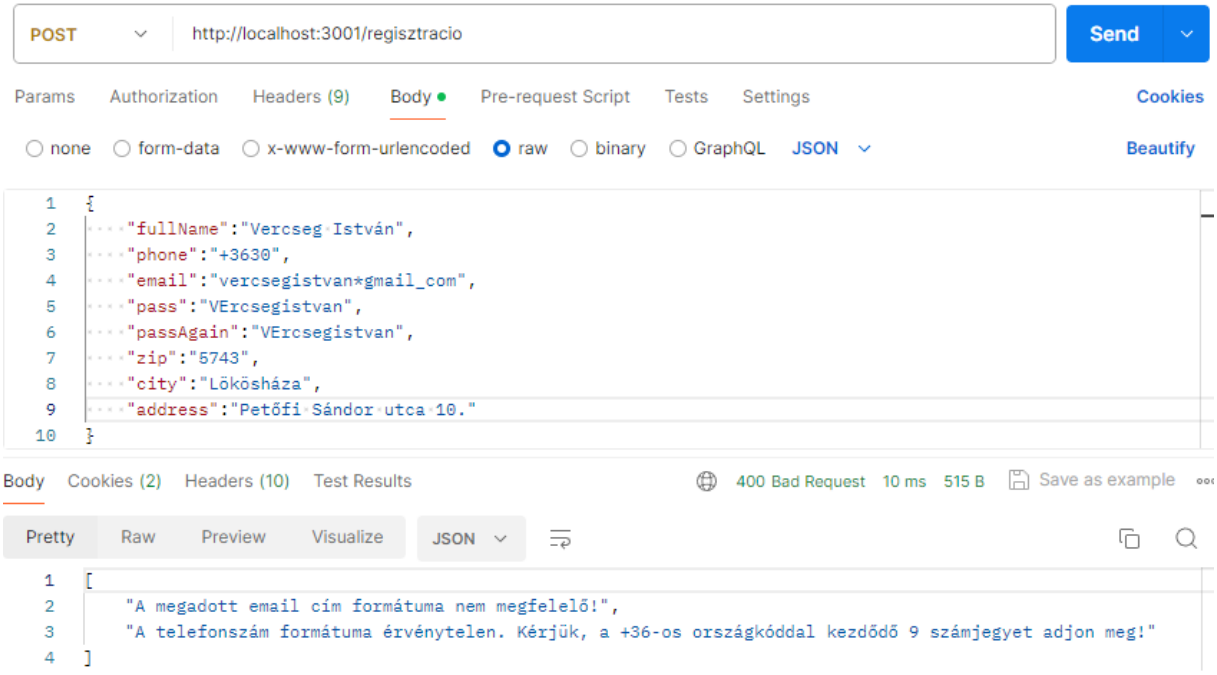
- Második teszteset: A könyvkölcsönző felületre csak olyan email címmel lehet regisztrálni, amivel korábban még nem tették. A users táblában az email mező UNIQUE megkötést kapott, ezáltal csak egyszer szerepelhet az adattáblában. Vegyük szemügyre azt az esetet, ha olyan email címmel kívánunk regisztrálni, amely már szerepel az adatbázisban. A fenti adatokat felhasználva azt várnánk el, hogy a visszatérő üzenet „***A megadott email címmel már regisztráltak a felületen!***” ***400 Bad Request*** státuszkóddal, mivel a kissjozsef@gmail.com címmel az első tesztesetben regisztráltunk.



18. ábra Sikertelen regisztráció adatbázisban szereplő email cím miatt

- Harmadik teszteset: Ennél a tesztelési pontnál megvizsgáljuk azt a lehetőséget, ha a regisztráció meghiúsul a beviteli mezőben megadott adatok formai helytelensége végett. Az applikáció a regisztrációs folyamat során 4 formai követelményt vizsgál, az email cím struktúrája (reguláris kifejezéssel), a telefonszám formátuma (szintén reguláris kifejezésnek való megfelelés), a jelszó hosszúsága(legalább 8 karakteresnek kell lennie), a jelszó és a jelszó megerősítése mezőbe rögzített adat egyezősége (két begépelt jelszó megegyezik-e). A tesztelés során a **telefonszámot helytelenül adtuk meg** (+3630 adatot próbáltunk rögzíteni, kimaradt tovább 7 számjegy), továbbá az **email címet is nem megfelelően** adtuk meg (@

karakter helyett *, illetve . helyett _ karakter) Jelen esetben azt várnánk el, hogy a visszatérő érték ***400 Bad Request*** státuszkód.



19. ábra Sikertelen regisztráció formai okok miatt

Mind a két jogosultsági szinten 3-3 db. tesztesetet vettünk, amelynek során megállapítottuk, hogy backend oldalon az elvárásoknak megfelelően működik a webapplikációnk.

2.4.2. Frontend oldalon végzett tesztesetek

A frontend oldali tesztelést Cypress v13.8.1 alkalmazásban végeztük. A Cypress egy fejlesztők és tesztelők számára kialakított frontend tesztelő keretrendszer, amely lehetővé teszi webalkalmazások tesztelését valós időben futó böngészőkörnyezetben. A tesztelésünk során a felkínált böngésző lehetőségek közül a Microsoft Edge v124-et választottuk. Minden esetben e2e (end-to-end) tesztekkel dolgoztunk annak érdekében, hogy megvizsgáljuk a felhasználói interakciók hogyan hatnak a teljes alkalmazás működésére, ahogy azok a valóságban is történnek. Két főbb részt vizsgáltunk, az egyik a kezdőlap megjelenését, a másik pedig a felhasználó belépését.

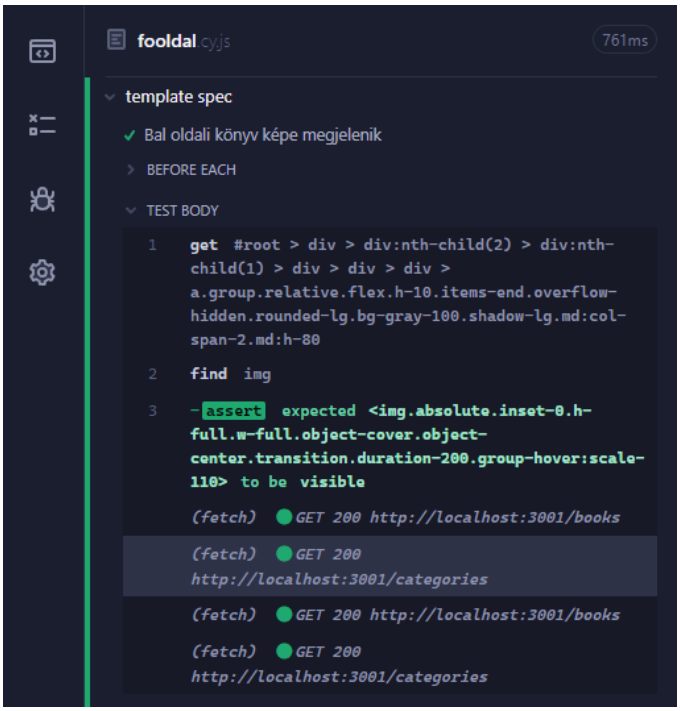
2.4.2.1.A kezdőoldal(főoldal) megjelenésének vizsgálata

- Első teszteset: A könyvkölcsönző webapplikáció kezdő oldalán bal oldalt, **láthatónak kell lennie** egy kinyitott könyvnek. Tulajdonképpen az alábbi kódsor azt fogja vizsgálni, hogy a keresett image státusza látható-e.

```
describe('template spec', () => {
  beforeEach('',()=>{cy.visit('http://localhost:5173')})

  it('Bal oldali könyv képe megjelenik',()=>{
    cy.get('#root > div > div:nth-child(2) ...:col-span-2.md\\:h-80').find('img').should('be.visible
  })



})
```

20. ábra Frontend főoldal teszteset - képmegjelenés

A vizsgálat elvégzése során azt várjuk, hogy a „***Bal oldali könyv képe megjelenik***” nevű teszt sikeresen lefut, a Cypress ***zöld pipát tesz a teszt mellé***, továbbá az ***assert*** (elvárt eredményre utal) is zöld jelzéssel tér vissza.



21. ábra Frontend főoldal teszteset - sikeres képmegjelenés

- Második teszteset: A menüsor a betöltődött weblapon 3 elemet tartalmaz a be nem lépett felhasználók esetében, amelyek az alábbiak:

  -Főoldal

  -Belépés

  -Regisztráció

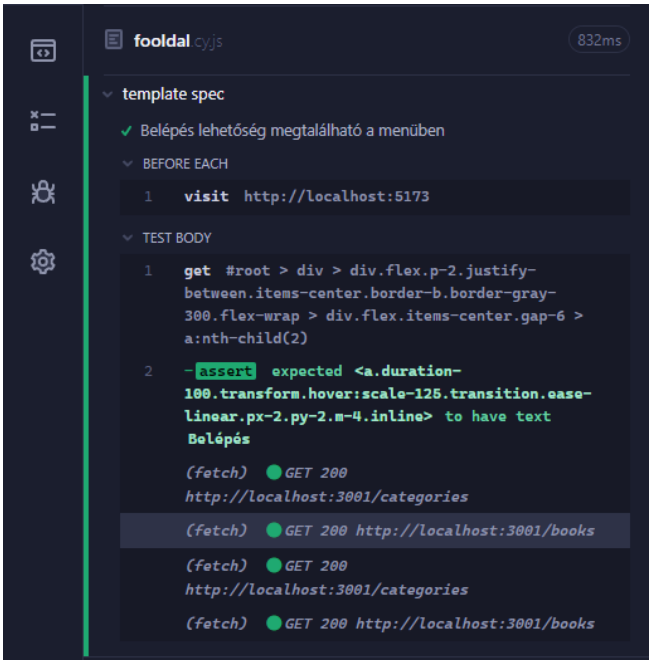
Jelen tesztelés arra irányul, hogy a „navigációs sáv” tartalmazza a Belépés lehetőséget.

A kódsor a következő:

```
describe('template spec', () => {
  beforeEach('',()=>{cy.visit('http://localhost:5173')})

  it('Belépés lehetőség megtalálható a menüben', () => {
    cy.get('#root > div > div.flex.p-...ms-center.gap-6 > a:nth-child(2)').should('have.text','Belépés')
  })

})
```

22. ábra Frontend főoldalteszt - Belépés menüpont

Fentiektől azt várnánk el, hogy a ***’Belépés lehetőség megtalálható a menüben’*** tesztsor eredménye, az előzőhöz hasonlóan „zöld pipával” és zöld ***assert*** értékkel térjen vissza.



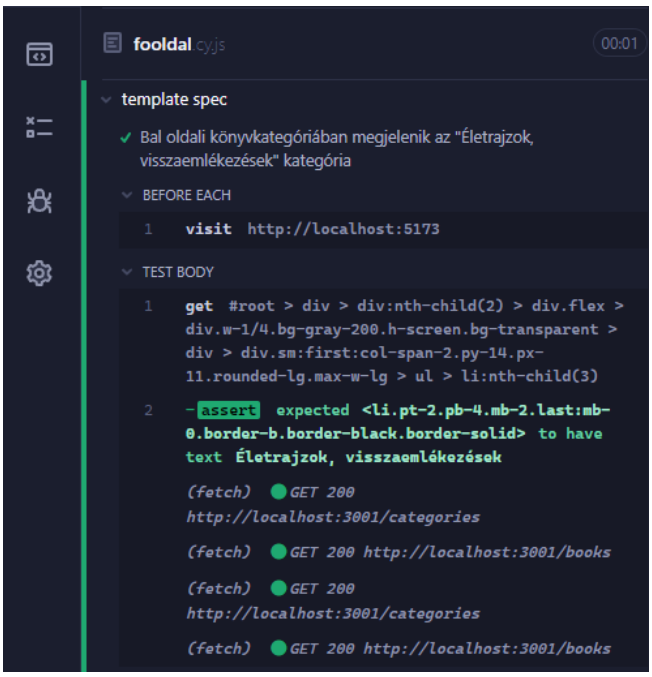
23. ábra Frontend főoldalteszt - Belépés menüpont sikeres

- Harmadik teszteset: A főoldal megjelenésének további tesztelése során megvizsgáljuk, hogy a bal oldali LeftSideBar komponens által renderelt könyvkategóriáknál megjelenik-e az „Életrajzok, visszaemlékezések” kategória.

```
describe('template spec', () => {
  beforeEach('',()=>{cy.visit('http://localhost:5173')})

  it('Bal oldali könyvkategóriában megjelenik az "Életrajzok, visszaemlékezések" kategória', () => {
    cy.get('#root > div >g.max..d(3)').should('have.text','Életrajzok, visszaemlékezések')
})

})
```

24. ábra Frontend főoldal tesztelés - könyvkategória

Az elvárásunk továbbra is az, hogy a „***Bal oldali könyvkategóriában megjelenik az „Életrajzok, visszaemlékezések" kategória***" tesztesetünk sikerrel lefut, zöld pipa jelzéssel lesz ellátva és az ***assert*** érték is zölden jelenjen meg.



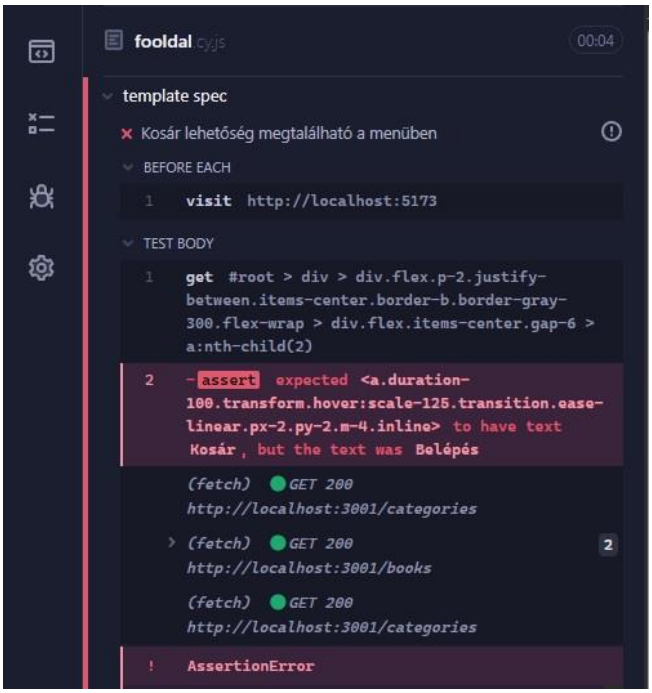
25. ábra Frontend főoldal tesztelés - könyvkategória sikeres

- Negyedik teszteset: Az fenti elemzések során olyan elvárásoknak szerettünk volna megfelelni, amelyek sikerrel végződnek. Vegyünk egy olyan esetet is, amely, akkor számít sikeresnek, ha a Cypress hibát fog jelezni. A bekezdés elején tárgyaltuk, hogy a bejelentkezés nélküli felhasználó a 3 menüpontot láthat. Készítettünk egy olyan tesztkódsort, amely azt feltételezi, hogy a menüszalag tartalmaz *Kosár* menüpontot is.

```
describe('template spec', () => {
  beforeEach('',()=>{cy.visit('http://localhost:5173')})

  it('Belépés lehetőség megtalálható a menüben', () => {
    cy.get('#root > div > ....-center.gap-6 > a:nth-child(2)').should('have.text','Kosár')
  })
})
```

26. ábra Frontend főoldal tesztelés - kosár

Az előző tesztesetektől eltérően azt várnánk el, hogy a „Kosár lehetőség megtalálható a menüben” tesztünk hibát jelezve fusson le, a ***zöld pipa helyett piros x-el*** térjen vissza és az ***assert*** elvárásunk is ***hibás legyen***.



27. ábra Frontend főoldal tesztelés - kosár sikertelen

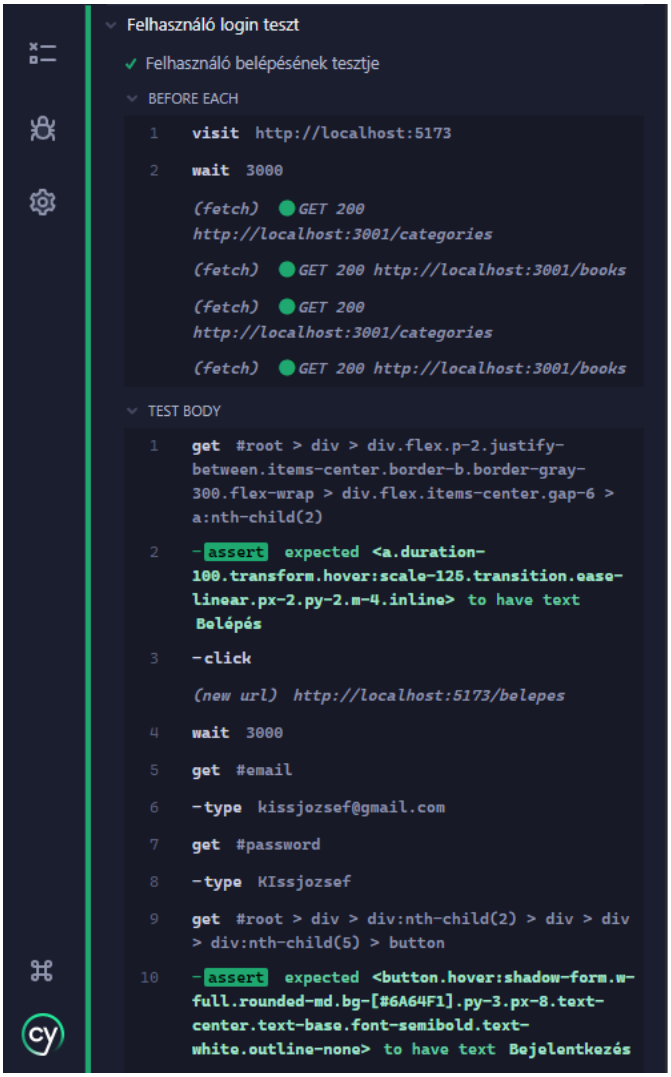
2.4.2.2.Adminisztrációs jogosultsággal rendelkező felhasználó belépésnek tesztelése.

Az fenti esetektől eltérően itt nem a megjelenést vizsgáltuk, hanem az admin felhasználó belépésének folyamatát és némi navigálást, interakciót.

- Első teszteset: A kódsor a következőket szimulálja. Megvizsgálja, hogy a menüpont között szerepel-e a Belépés lehetőség, ha igen rákattint, vár 3 másodpercet (a várakozás azért szükséges, mert amennyiben vizuálisan követjük a tesztet a böngészőben, pillanatok alatt lefutna és nem tudnánk lépésről lépésre nyomon követni), megadja az email címet és jelszót(a kissjozsef@gmail.com cím lett a teszt tárgya), rákattint a bejelentkezés gombra, vár további 3 másodpercet, a navigáció a Könyvek adminisztrálás funkcióhoz vezeti. Az adatbázisban található könyvek kilistázást követően 3 másodperc múlva a felhasználó kilép.
  Az elvárásain szerint a „***Felhasználó belépésnek tesztje***” tesztesetünk ***zöld pipa*** jelzést kap, továbbá az ***assert*** értékünk is ***zölden*** jelenik meg.

```
describe('Felhasználó login teszt', () => {
  beforeEach('',()=>{
    cy.visit('http://localhost:5173')
    cy.wait(3000)
  })

  it('Felhasználó belépésének tesztje',()=>{
    cy.get('#root > divgap-6 > a:nth-child(2)').should('have.text','Belépés').click()
    cy.wait(3000)
    cy.get('#email').type('kissjozsef@gmail.com')
    cy.get('#password').type('KIssjozsef')
    cy.get('#root > div > div:nth:nth-child(5) > button').should('have.text','Bejelentkezés').click()
    cy.wait(3000)
    cy.get('#root > div > div.p-40.button').should('contains.text','Könyvek adminisztrálás').click()
    cy.wait(3000)
    cy.get('#root > di-wrap > div.flex.items-cd(4)').should('contains.text','Kilépés').click()

  })

})
```

28. ábra Frontend belépés teszt - kód

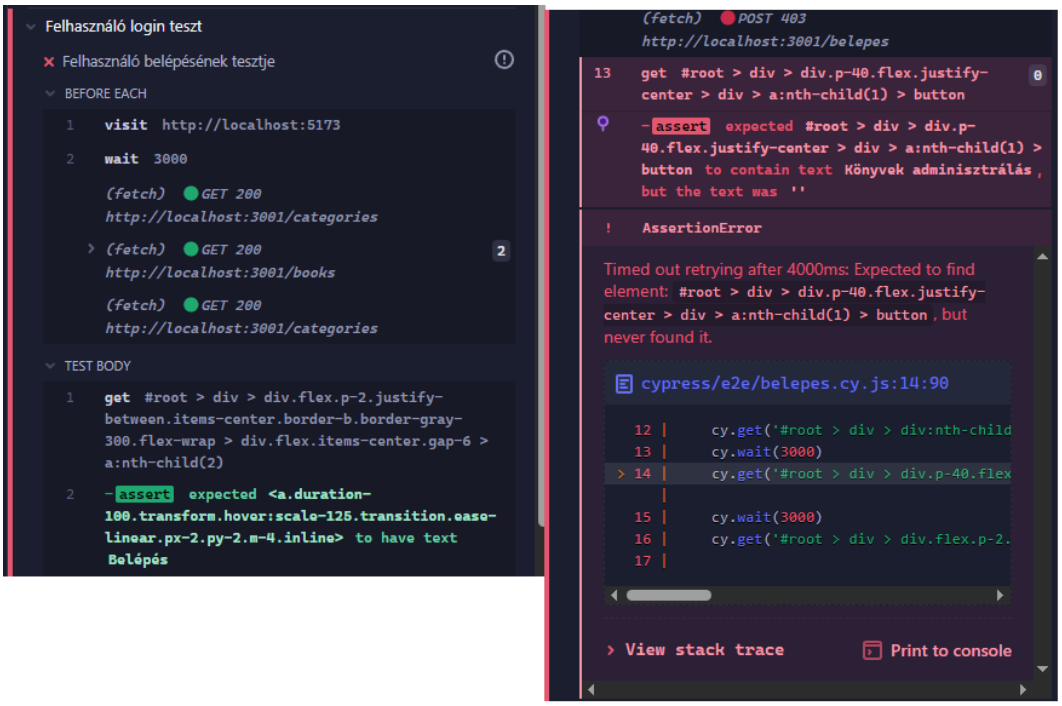


29. ábra Frontend belépés teszt - sikeres

- Második teszteset: A sikeres belépést követően szeretnénk megvizsgálni azt, hogy mi lesz abban az esetben, ha olyan email cím/jelszó párossal szeretnénk belépni, amely nincs regisztrálva a konyvkolcsonzo adatbázisban.

Jelen teszteset kódsora megegyezik az előzőével, azonban módosítottuk a szimuláció során megadott email címet és jelszót. Azt feltételezzük, hogy a sikeres belépéstől eltérően ***piros x jelzést kapunk***, továbbá az ***assert érték is pirosan*** jelenik meg.

A teszt lefutása után a belépési szimuláció végeredménye hibás lett (a bal oldali képen felül látható a piros x jelzés). Az első tesztfeltétel sikeresen lefutott (az ***assert*** elvárás zöld színnel jelenik meg) mivel megtalálja a „Belépés” lehetőséget de a ***további részeken piros jelzést*** ad vissza.



30. ábra Frontend belépés teszt - hiba

Összességében 6 db. tesztesetet futtatunk le, amelyek eredményére támaszkodva bizonyosságot nyertünk, hogy frontend oldalon is a kezdetekben meghatározott elvárásoknak megfelelően működik a programunk.

2.5. Fejlesztési lehetőségek

A webapplikációnk a tervezett alapvető funkciókat ellátja. Azonban az alábbi bővítésekkel „felhasználóbarátabbá” lehetne tenni a weboldalt:

- a kölcsönzés lejárta előtt pár nappal emlékezteti az applikáció a felhasználót (pl. email kiküldésével), a könyv visszavételének végső dátumáról
- a kikölcsönözhető könyvek bővülése esetén (újabb kölcsönözhető könyvek kerülnek az adatbázisba) hírlevél kiküldése a regisztrált felhasználók részére
- a felhasználó érdeklődési körének megfelelően kikölcsönözhető könyvajánlatok megjelenítése az oldalon
- a felhasználó a belépést követően nyomon tudná követni a korábbi rendeléseit
- könyv kosárba tétele anélkül, hogy meg kellene nyitni a könyvről szóló információkat (gyors kölcsönzési gomb kialakítása a felületen)
- eseménynaptár kialakítása a felületen, amely tájékoztatást ad az újonnan megjelenő könyvek időpontjáról
- a felhasználónak lehessen értékelést adnia az általa kikölcsönzött könyvekről, amelyet más felhasználók is láthassanak
- sütik helyett tokenek használata az autentikációs folyamat során
- felhasználói felületen könyvkeresési lehetőség pl. könyv címe alapján
- adminisztrátori felületen a lejáró kölcsönzési határidők feltüntetése (lejárat előtt pár nappal)
- adminisztrátori felületen a kölcsönzés dátuma alapján történő sorba-rendezés
- adminisztrátori felületen a kilistázott felhasználok adatai között szerepeljen a telefonszám is
- a felhasználó fiók résznél lehessen avatar képet feltölteni
- kapcsolattartási felület kialakítása (pl. űrlap kitöltésével információt kérő üzenetet tud küldeni a felhasználó)

3. Felhasználói dokumentáció

A könyvkölcsönző egy könyvek kölcsönzésére szolgáló webes applikáció. A felhasználónak lehetősége van regisztrálni a felületre, belépni, majd a könyvek között keresgélve kiválaszthatja

a számára megfelelő könyvet és a kosárba adhatja azokat (egyszerre akár több könyvet is). A kölcsönzés befejezésével az alkalmazás egy email-t küld a könyvkölcsönzés részleteiről.

3.1. Telepítés/indítás

Az applikáció elindításához egy internetezésre alkalmas asztali számítógépre vagy laptopra van szükség. Különös hardverigényt nem igényel a program, azonban fontos, hogy az alábbi programok telepítve legyenek az eszközön:

1. Visual Studio Code *1.88.1*– telepítéssel kapcsolatos információk : https://code.visualstudio.com/download
2. XAMPP szoftvercsomag 8.2.0 verziója – telepítéssel kapcsolatos információk: https://www.apachefriends.org/download.html
3. Node.js – telepítéssel kapcsolatos információk https://nodejs.org/en/download

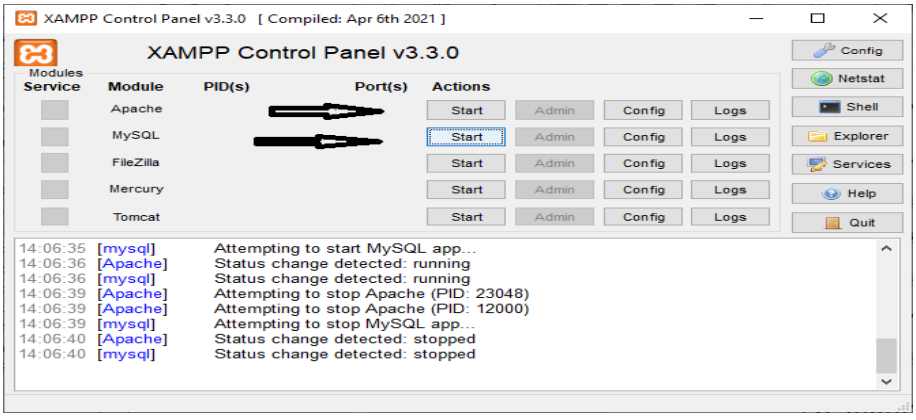
Az alábbi linkről tudja letölteni a frontend és backend oldalt is tartalmazó applikációt:

http://213.157.112.170:3000/szf_esti_2024

Letöltést követően két mappát és egy „konyvkolcsonzo.sql” fájlt találhat.

Az applikáció elindítása.

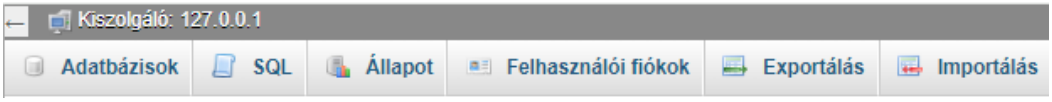
1. Indítsa el a XAMPP Control Panelt
   Majd kattintson az „Apache” és a „MySQL” modul start gombjaira.



29. ábra XAMPP Control Panel

Amennyiben sikeresen elindította a csomagot, a gomb start felirata stop-ra módosul.

2. Ezt követően a MySQL modulnál keresse meg az ***Admin*** gombot és kattintson rá, az alapértelmezett böngészője megnyílik és a http://localhost/phpmyadmin/ URL fog betöltődni.
3. Létre kell hoznia az az adatbázist. Keresse meg a felső menüsorba az „SQL" menüpontot.

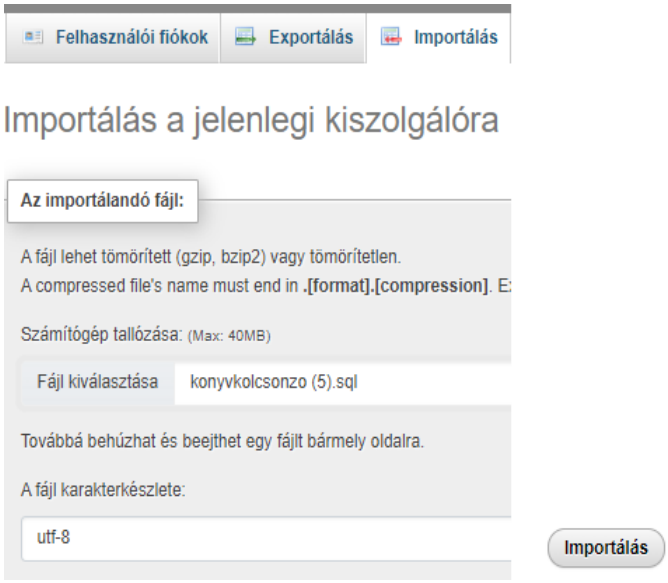


30. ábra SQL menüpont a phpMyAdmin felületén

4. A felnyíló ablakba kérem illessze be a következő kódsort.

   CREATE DATABASE konyvkolcsonzo DEFAULT CHARACTER SET utf8 COLLATE utf8_hunagrian_ci;

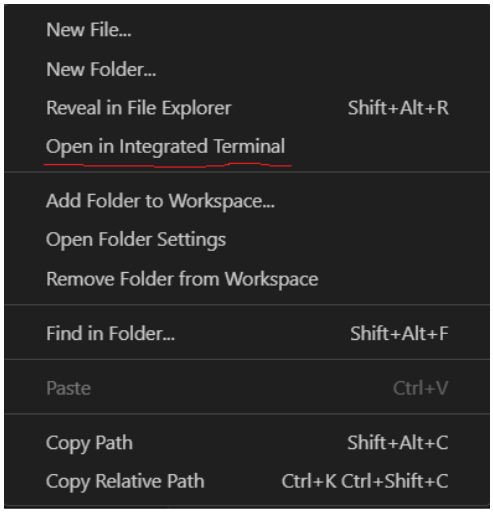
5. Be kell importálja az adatokat az előbb létrehozott adatbázisba.
   Keresse meg a felső menüsorban az Importálás lehetőséget. Kattintson rá, majd a Fájl kiválasztása résznél töltse be a korábban letöltött konyvkolcsonzo.sql fájlt és az oldal alján kattintson az Importálás gombra. A művelet után adatbázisba bekerülnek az adatok.



31. ábra Sql fájl importálása phpMyAdmin felületen

6. Indítsa el a Visaul Studio Code programot.

   Nyissa meg a projektmappát (File,Open Folder…"Konyvkolcsonzo"). Két mappát fog találni. Az egyik a *backend*, a másik a *frontend*. A betöltést követően kattintson jobb egérgombbal a backend mappa nevére és válassza az „Open in Integrated Terminal" lehetőséget. A terminál az ablak alsó részében aktív lesz.

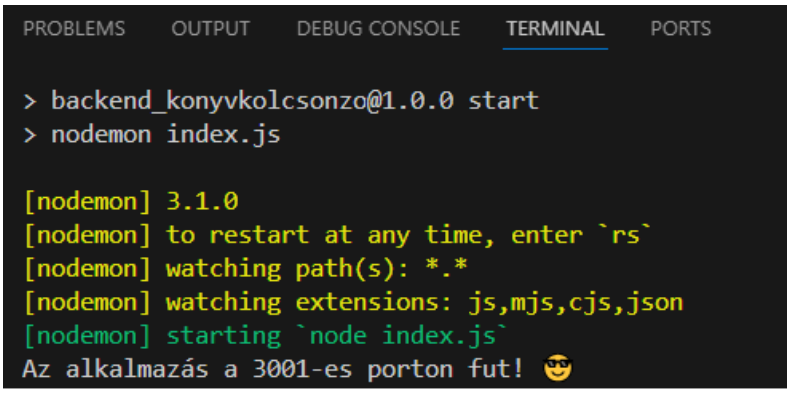


32. ábra Visual Studio Code - Terminál megnyitása

A terminálba írja be a következő parancsot:

npm start

A parancs kiadását követően elindul a backend alkalmazás.

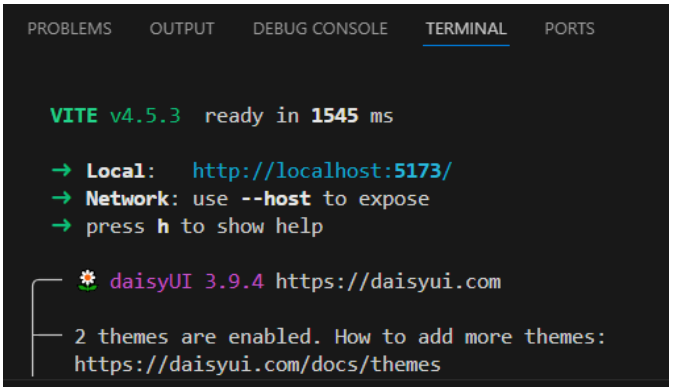


33. ábra Futó backend alkalmazás

Most kattintson jobb egérgombbal a frontend mappára és nyissa azt is meg a Terminálba (jobb egérgomb, „Open in Integrated Terminal"). A terminálba írja be a következő parancsot:

npm run dev

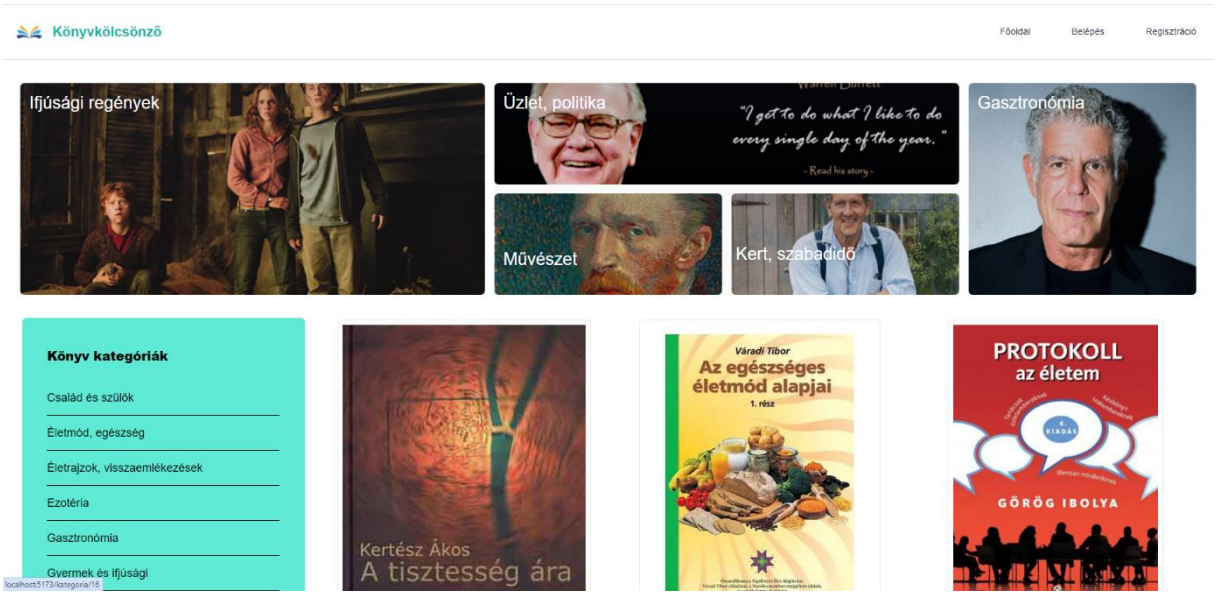
A parancs kiadását követően az alábbiaknak kell megjelennie.



34. ábra Futó frontend

A ctrl billentyű lenyomása mellett kattintson egérrel a http://localhost:5173 linkre.

Az alapértelmezett böngészője elindul és betölt a könyvkölcsönző alkalmazás.



35. ábra Könyvkölcsönző főoldal

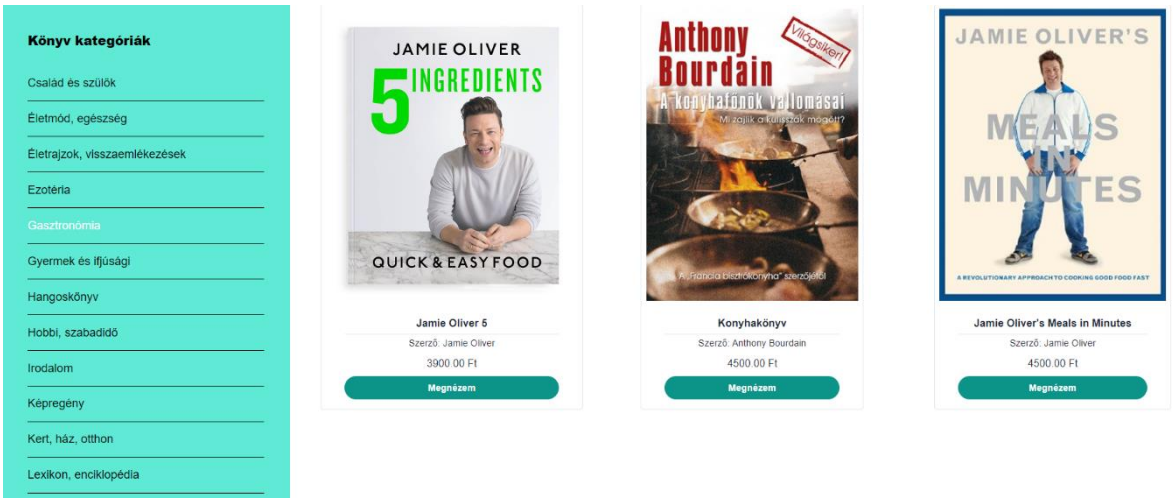
Az alkalmazást sikeresen elindította!

3.2. Az alkalmazás felépítése és funkciói

A könyvkölcsönző webapplikációnak három fő színtere létezik. A be nem lépett/regisztrált felhasználó-, a belépett/regisztrált felhasználó- és az admin jogosultsággal rendelkező felhasználó színtere. Színtérként azt értjük, hogy mit lát a képernyőn a felhasználó, milyen jogosultságokkal rendelkezik.

3.2.1. Nem regisztrált/belépett felhasználó színtere- PUBLIC USER

A magát még nem regisztrált felhasználót (public user) a webapplikáció a főoldalra navigálja (34.ábra). A bal oldali könyvkategória szűrő aktív, azaz amennyiben szeretné kilistáztatni az adatbázisban szereplő könyveket kategória alapján, csupán annyi a dolga, hogy rákattint a megtekinteni kívánt könyvkategóriaára.



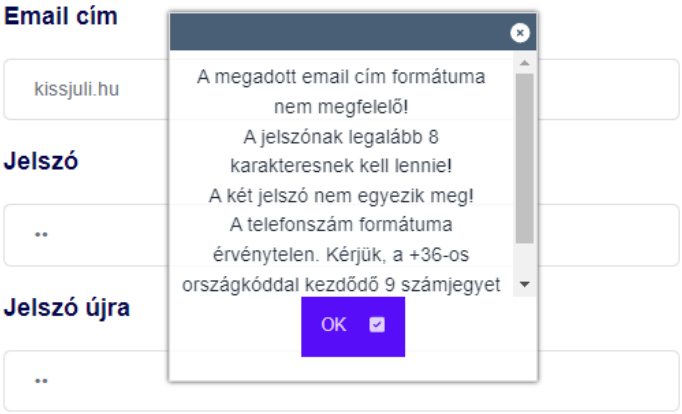
36. ábra Könyvek kilistázása kategória alapján - Gasztronómia kategória

A regisztráció menete.

Az oldal felső részén elhelyezkedő menüsorból válassza ki a regisztráció lehetőséget. Kérjük az űrlapelemek kitöltését végezze el! Figyelem! A webapplikáció az alábbi beviteli mezők formai tartalmát elemzi:

- email cím formátuma: a felhasználónév tartalmazhat ékezetes betűket, számokat pontot, aláhúzást, kötőjelet; domain név tartalmazhat betűket, számokat, pontot, kötőjelet, aláhúzást; @ és . karaktert tartalmaznia kell az email címnek
- a jelszónak legalább 8 karakteresnek kell lennie
- a jelszó és a jelszó újra mezőben megadott értéknek egyeznie kell
- a telefonszám formátuma: +36-os országhívószám és 9 szám kövesse
- továbbá fontos, hogy olyan email címmel kísérelje meg a regisztrációt, amellyel még nem tette

Amennyiben hibás formátumban adja meg az adatokat az alábbi hibaüzenet fogadja:

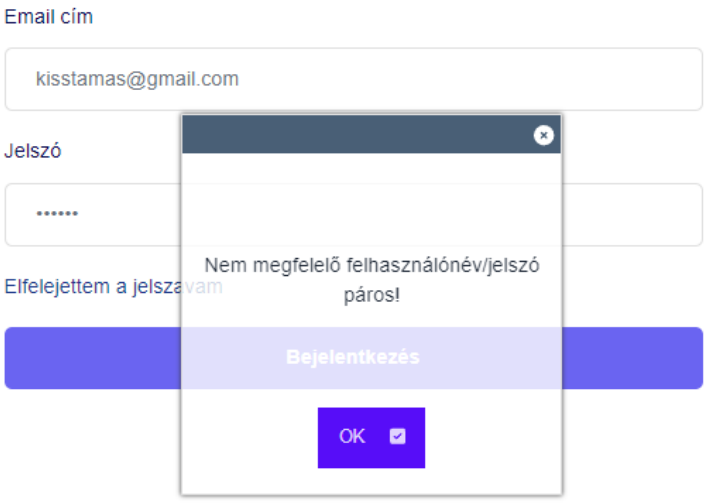


37. ábra Hibaüzenet regisztrációkor

Sikeres regisztrációt követően az alkalmazás a belépési felületre navigálja.

Belépés menete

A regisztrált felhasználóknak lehetőségük van a rendszerbe történő belépésre az email cím/jelszó páros megadásával. Amennyiben hibás jelszó/email címmel próbál bejelentkezni, abban az esetben a rendszer figyelmezteti!



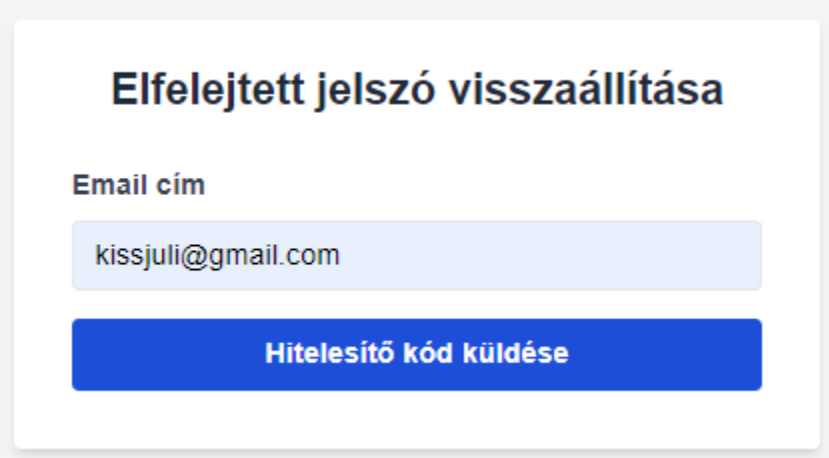
38. ábra Rossz email cím/jelszó páros megadása

Sikeres belépés esetén a felhasználó már több funkcióját is használhatja a könyvkölcsönző oldalnak.

Elfelejtett jelszó. Új jelszó megadása.

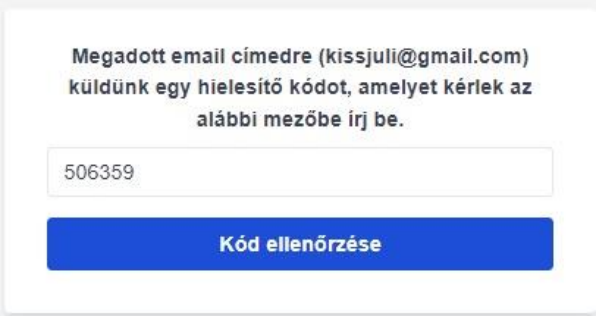
Amennyiben a felhasználó elfelejtette a korábbi regisztráció során megadott jelszavát, az oldal lehetőséget kínál arra, hogy az adatbázisban már meglévő email címéhez újabb jelszót párosítson.

A helytelen jelszó/email cím páros megadása esetén a hibaüzenet OK gombjára kattintva szüntesse meg a felugró hibajelzést, majd keresse meg az „Elfelejtettem a jelszavam” lehetőséget. A megjelenő beviteli mezőbe írja be a korábban már regisztrált email címét!



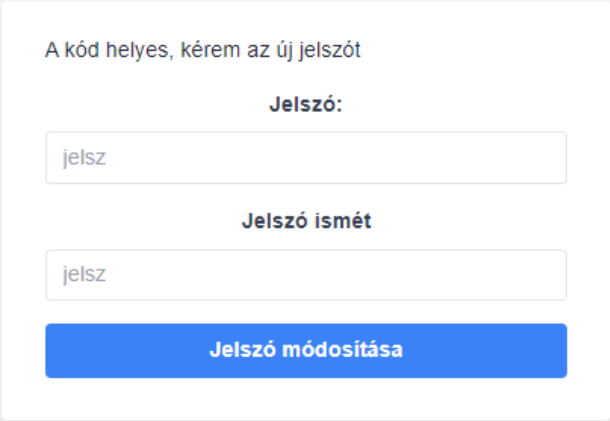
39. ábra Elfelejtett jelszó - Kódkérés

A hitelesítő kódot a rendszer a megadott email címre kiküldi, majd az ott szereplő kódot írja be a felület beviteli mezőjébe és kattintson a „Kód ellenőrzése” lehetőségre.



40. ábra Kód ellenőrzése

Amennyiben a kód helyes, adja meg új jelszavát.

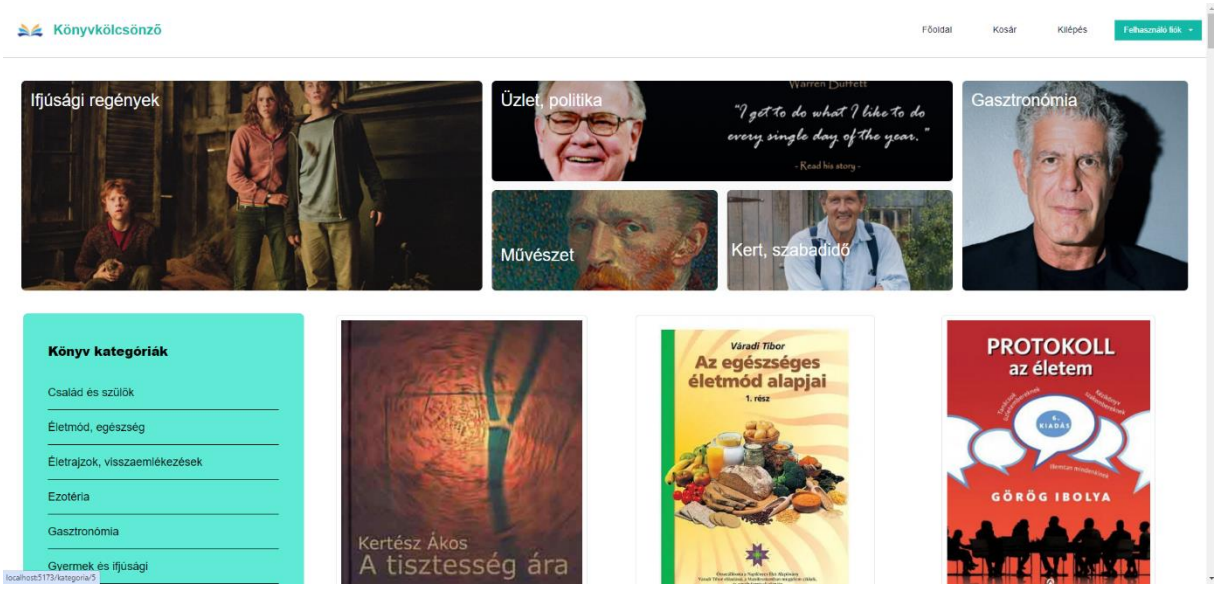


41. ábra Új jelszó megadása

Itt is fontos, hogy a Jelszó és a Jelszó ismét mező tartalma megegyezzen, továbbá a hossza minimum 8 karakter hosszúságú legyen.

3.2.2. Belépett felhasználó színtere – USER

A belépett felhasználó felső menüsorán megjelentek a „*Kilépés*”, „*Kosá*r” és „*Felhasználó fiók*” menüpontok. A Kilépés menüpont kilépteti a felhasználót és elnavigálja a főoldalra (PUBLIC USER).



42. ábra Belépett felhasználó főoldala

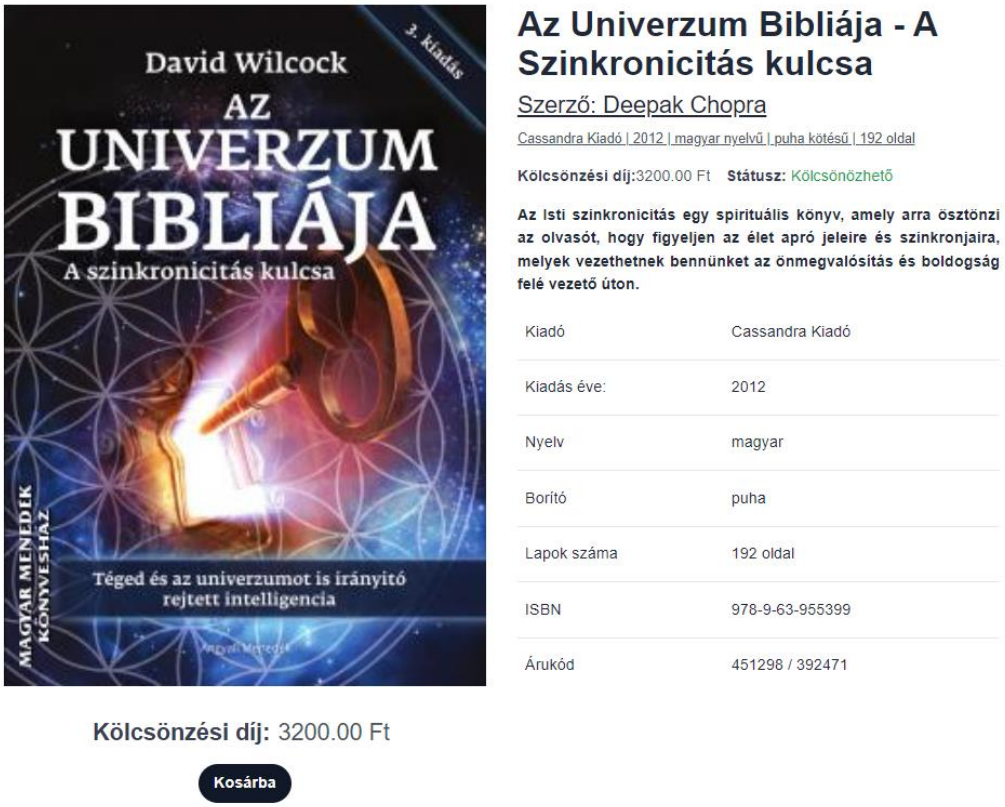
Amennyiben a *Főoldal* lehetőséget választja, bármely pontján legyen az oldalnak, mindig a belépést követően megjelenő kezdőoldalra navigálja a webapplikáció.

A *Felhasználó fiók – Adataim* menüpontja kilistázza a regisztrációkor megadott alapadatait.

A *Kosár* menüpontot a rendelés folyamatának ismertetése során mutatjuk be.

A könyvrendelés menete

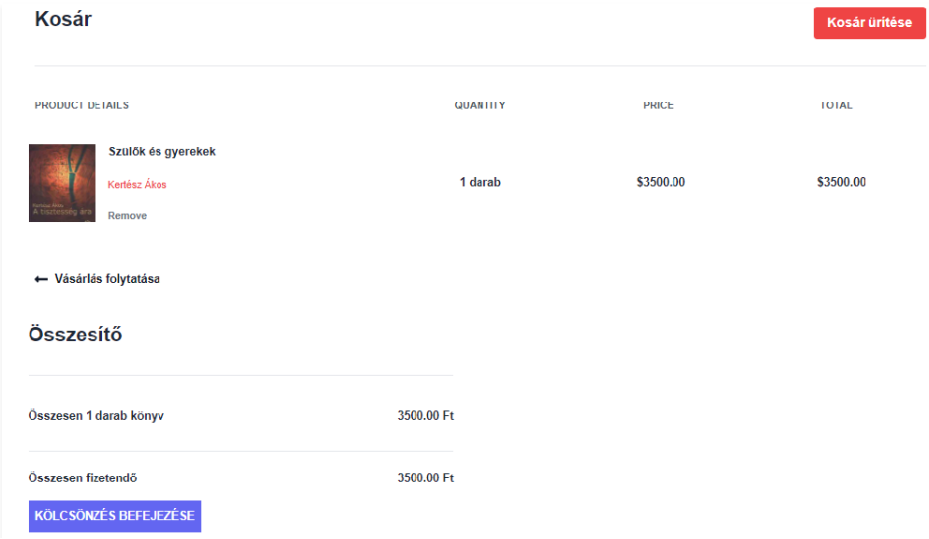
A főoldalon megjelenő könyvek „*Megnézem*” gombjára kattintva a kiválasztott könyvekről információkat nyújt az oldal (pl. Kiadó, Kiadás éve, ISBN, tartalom rövid ismertetése stb.), illetve a kölcsönzés díja is megjelenik. Amennyiben elnyerte tetszését a könyv és ki szeretné kölcsönözni, kattintson a „Kosárba” gombra.



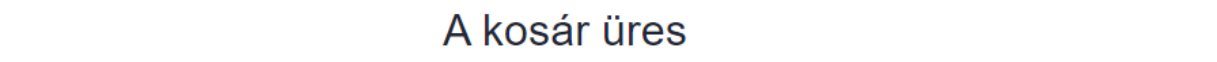
43. ábra Kiválasztott könyv adatai

A kosárba helyezést követően a Kosárhoz navigálja az oldal, ahol az alábbi lehetőségek közül választhat.

- Kosár ürítése – A kosárba rakott megrendelését kitörli, ezáltal a kosara üres lesz.
- Kölcsönzés befejezése – A megrendelést véglegesíti
- A Főoldal menüpontot választva tovább folytathatja a böngészést az oldalon

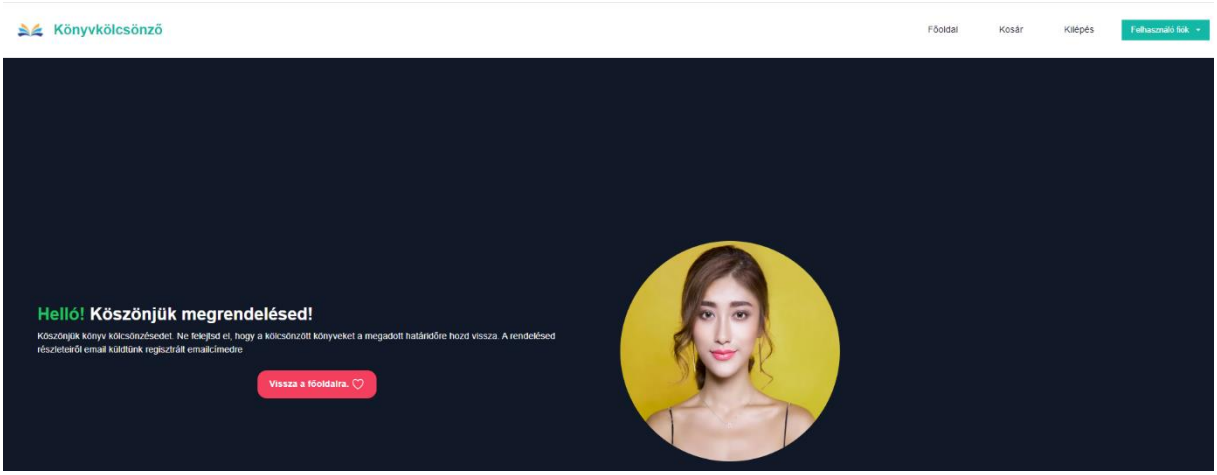


44. ábra Kosár tartalma



45. ábra Kiürített kosár

Amennyiben be szeretné fejezni a böngészést és ki kívánja kölcsönözni a könyvet, válassza a „*Kölcsönzés befejezése*” lehetőséget. A művelet befejeztével át lesz irányítva egy köszönetet tartalmazó oldalra, ahol a *Vissza a főoldalra* lehetőséget választhatja. A művelet befejeztével a rendszert elhagyja és ki lesz léptetve.



46. ábra Sikeres megrendelés - áruház elhagyása

A megrendelés végeztével a webapplikáció emailt küld a regisztrációkor megadott email címére, amely tartalmazza a rendelés részleteit.


Kedves Vásárló!

Köszönjük rendelését könyvkölcsönzőkben!

Tájékoztatjuk, hogy ez a levél azonnali automatikus visszaigazolás az Ön által leadott rendeléséről, i
számított **1-2 munkanapon belül történik.**

A rendelés részletei:

- A rendelés azonosítója: 805169
- Kölcsönzött könyv címe: Szülők és gyerekek: Mennyisége: 1 darab

A kölcsönzés megkezdésének ideje: 2024-05-05

A könyveket a következő időpontig köteles visszajuttatni részünkre: 2024-06-04

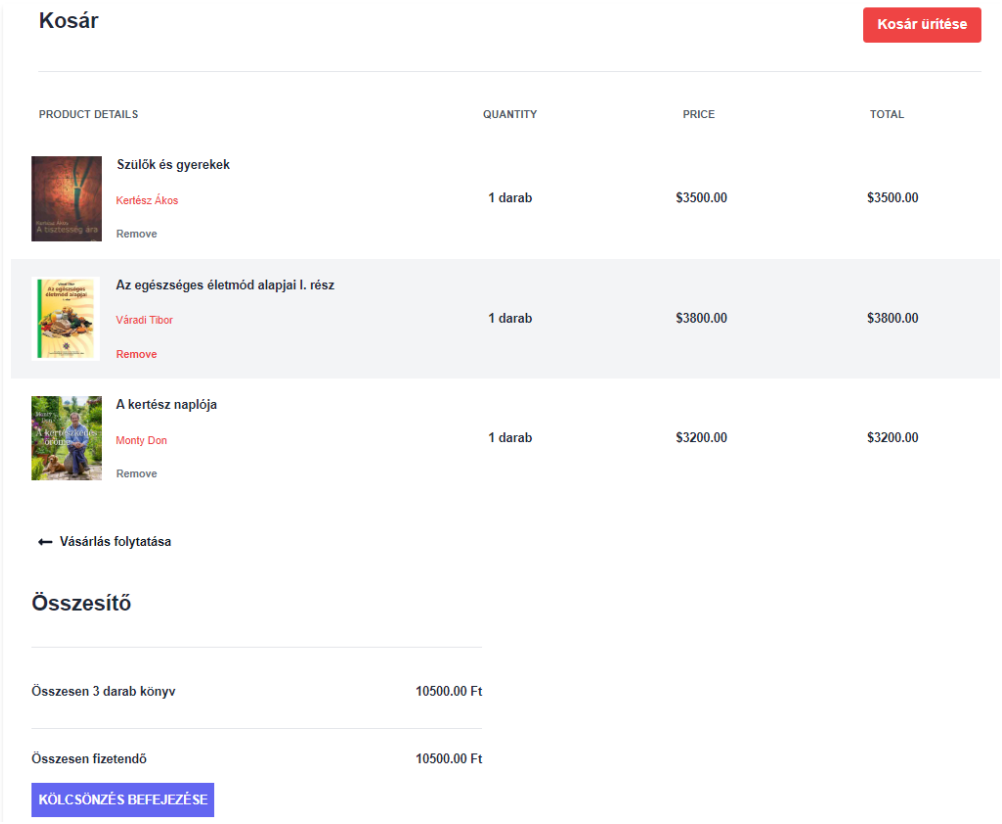
Tovvábbi szép napot kívánunk!

Könykolcsonzo.hu csapata

Email sent via EmailJS.com

47. ábra Sikeres megrendelésről email érkezése

Amennyiben több könyvet is belehelyezett a kosarába és nem szeretné az egész kosár tartalmát kiüríteni, csak egy nem kívánt tételt törölni, kattintson a „*Remove*” lehetőségre. A több tételt is tartalmazó kosár a fizetendő összeget kumuláltan is megjeleníti.



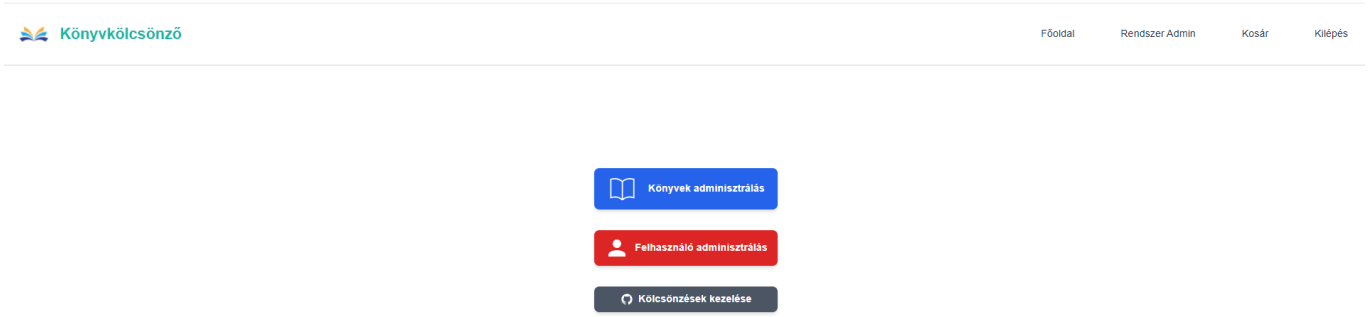
48. ábra Kosár tartalma több tétel esetén

3.2.3. Adminisztrátori jogosultsággal rendelkező felhasználó színtere (ADMIN USER)

Az adminisztrátor az oldalon működő összes funkció használatára jogosult. A sikeres belépést követően a program a felhasználót az adminisztrációs funkciókat tartalmazó oldalra navigálja. A menüszalagon megjelenik a „Rendszer Admin” menüpont is.

3 fő tevékenységi kört tartalmaz az oldal.

1. Könyvek adminisztrálás – könyvekkel kapcsolatos műveletek
2. Felhasználó adminisztrálás – regisztrált felhasználók adminisztrációs tevékenységek
3. Kölcsönzések kezelése – könyvek kölcsönzését karbantartó funkció



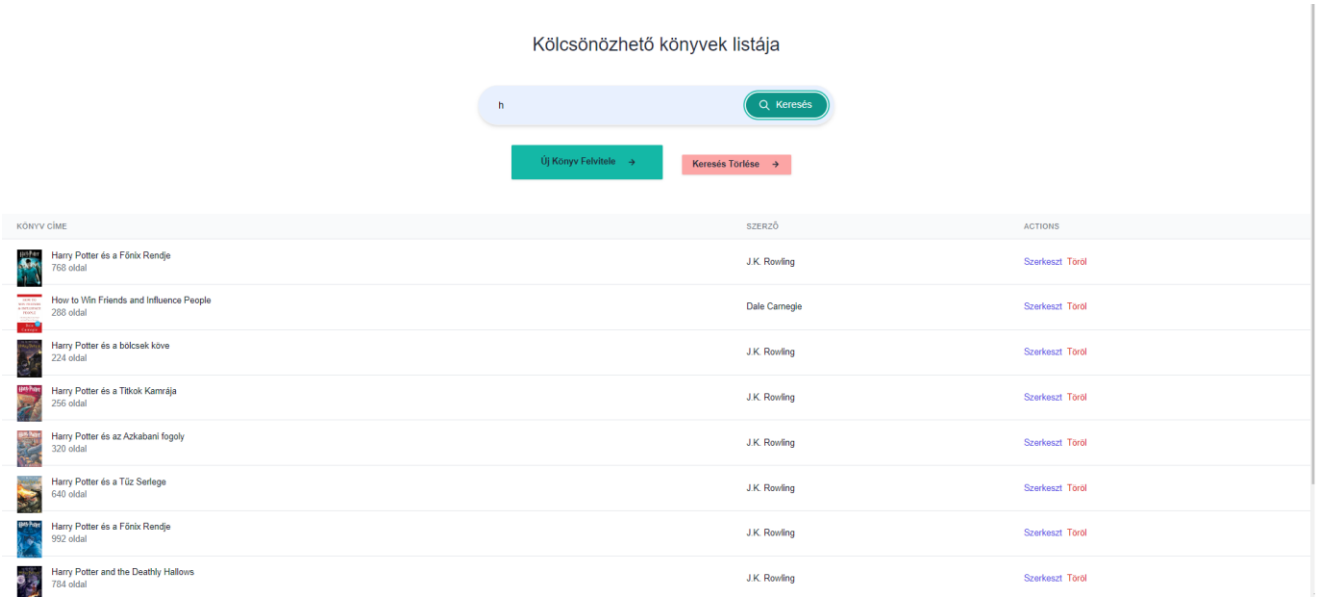
49. ábra Adminisztrátori felület

3.2.3.1. Könyvek adminisztrálás

A felületen az alábbi funkciókat tartalmazza.

Könyv keresése cím alapján.

Kattintson bele a „keresés könyv címe alapján” beviteli mezőbe, írja be a keresendő könyv címének kezdőbetűjét, majd nyomja meg a „Keresés” gombot.

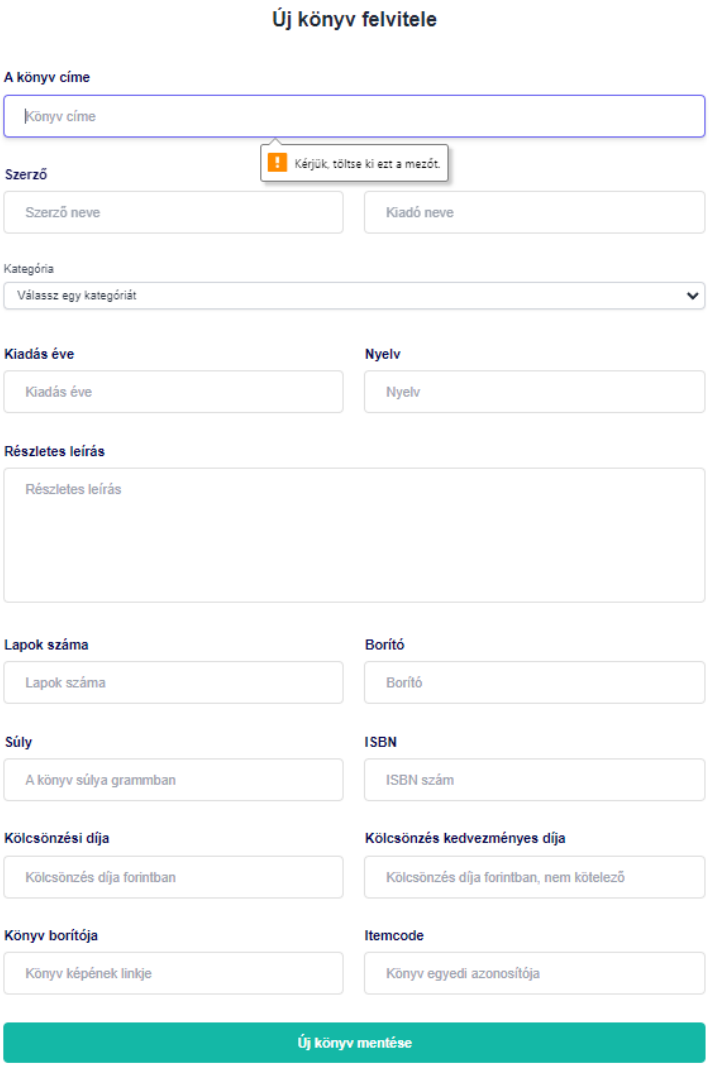


50. ábra „h” – val kezdődő könyvek kilistázása

Amennyiben meg kívánja szűntetni a szűrési feltételt, kattintson a „Keresés Törlése” gombra és újra az adatbázisban szereplő összes könyv kilistázásra kerül.

Új könyv felvitele

A felület lehetőséget biztosít adatbázisban még nem szereplő új könyv felvitelére. Amennyiben rögzíteni kíván egy újat, válassza a „Új Könyv Felvitele” gombot. Az oldalon megjelenő űrlapmezőket értelemszerűen töltse ki. Amennyiben valamelyik mezőt üresen hagyja, a felület narancssárga felkiáltó jellel ellátott „*Kérjük, töltse ki ezt a mezőt.*”, vagy „*Kérjük, válasszon egyet a lista elemei közül.*” hibaüzenettel figyelmezteti. A kedvezményes kölcsönzési díj megadása opcionális.

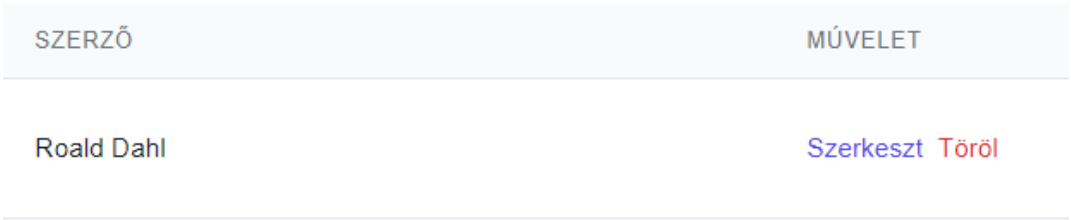


51. ábra Új könyv felvitele

A kitöltést követően kattintson az „Új könyv mentése” lehetőségre.

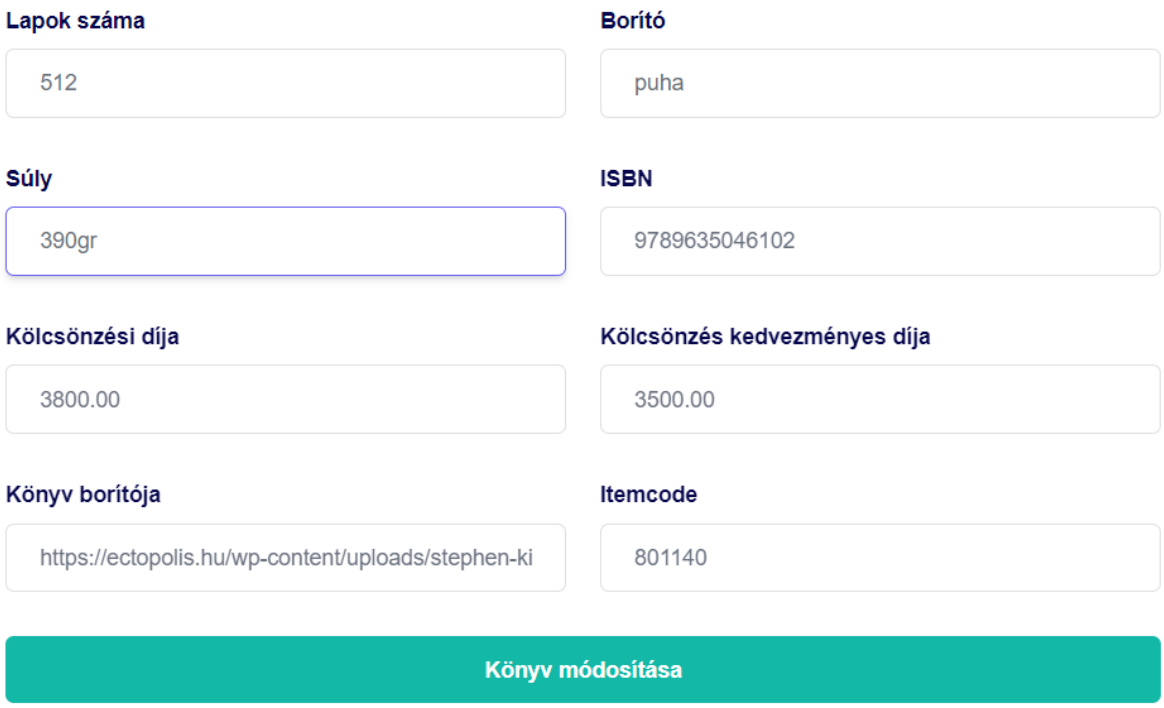
Adatbázisban szereplő könyv módosítása

A megjelenő könyvlistában megtalálható jobb oldalt, a Művelet oszlopban a „*Szerkeszt*” és „*Töröl*” lehetőség. Válassza a „*Szerkeszt*” lehetőséget!



52. ábra Könyv módosítása

Az oldalon megjelennek az adatbázisban szereplő könyvadatok. Kattintson arra a beviteli mezőre, amelynek adatait módosítani kívánja, majd írja azt át. A módosítás végeztével kattintson a „*Könyv módosítása*” gombra.

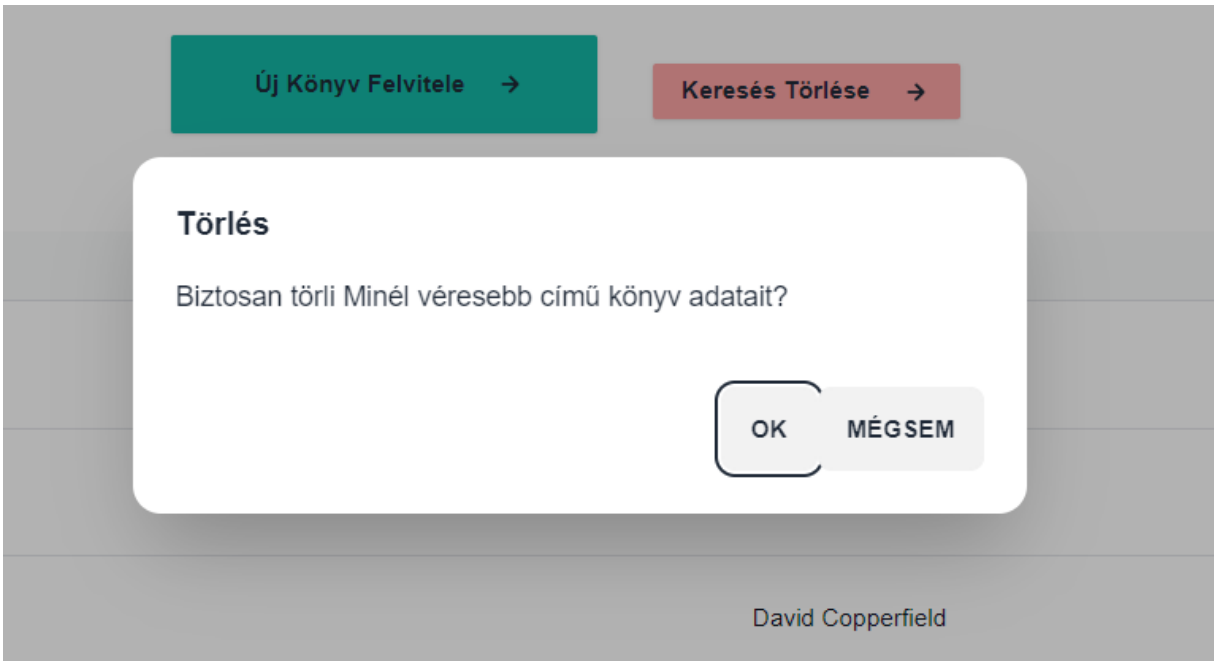


53. ábra Könyv adatainak módosítása

Könyv törlése

Amennyiben szeretné az adatbázisban szereplő könyvet kitörölni a kilistázott könyvek oldalon, a „Műveletek” oszlopban válassza a „*Törlés*” funkciót (lásd 51. ábra).

A rendszer biztonsági okokból megerősítést fog kérni a művelet elvégzéséhez! Az „*Ok*” gomb megnyomásával a könyv kitörlésre kerül az adatbázisból! A „*Mégsem*” válasz esetén visszanavigál a program a kilistázott könyvek oldalára.



54. ábra Könyv törlésének megerősítése

3.2.3.2. Felhasználó adminisztrálás

Az adminisztrátori felületen válassza a „*Felhasználó adminisztrálás*” lehetőséget (lásd 48. ábra). Az oldalon a rendszerbe regisztrált felhasználók adatai fognak megjelenni. amelyek az alábbiak:

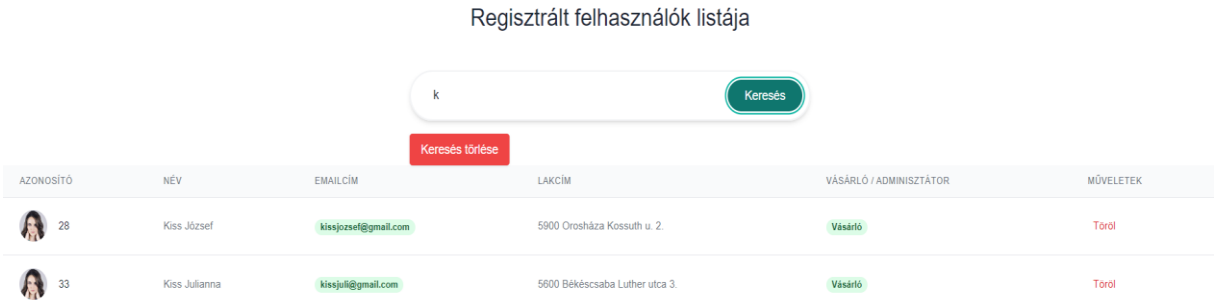
- AZONOSÍTÓ: A felhasználó egyedi azonosítója (regisztrációkor a rendszer automatikusan létrehozza)
- NÉV: A felhasználó neve
- EMAILCÍM: A felhasználó email címe
- LAKCIM: A felhasználó lakcíme

- VÁSÁRLÓ/ADMINISZTRÁTOR: Jogosultsági szintről ad tájékoztatást. *Vásárló* megnevezés estén USER, míg „*Adminisztrátor*” esetén ADMIN felhasználó.
- MŰVELETEK: A felhasználó törlése az adatbázisból

55. ábra Felhasználók listája

Felhasználó keresése név alapján.

Kattintson bele a „Felhasználó kezdőbetűje alapján” beviteli mezőbe, írja be a keresendő felhasználó nevének kezdőbetűjét, majd nyomja meg a „Keresés” gombot.



56. ábra "k" betűvel kezdődő nevű felhasználók

Amennyiben meg kívánja szűntetni a szűrési feltételt, kattintson a „Keresés Törlése” gombra és újra az adatbázisban szereplő összes felhasználó kilistázásra kerül.

Felhasználó törlése

Amennyiben szeretne eltávolítani felhasználót az adatbázisból, válassza a Műveletek oszlopból a „*Töröl”* lehetőséget.

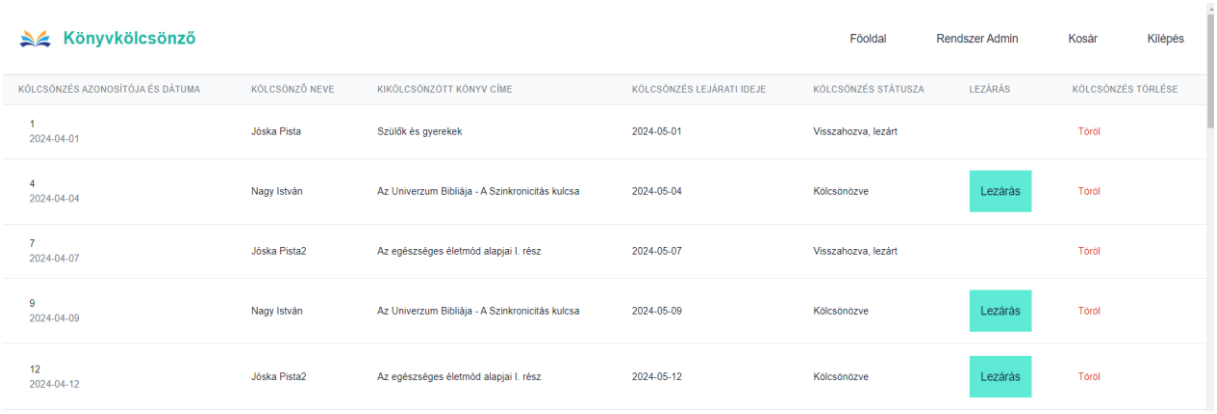
A rendszer biztonsági okokból megerősítést fog kérni a művelet elvégzéséhez! Az „*Ok*” gomb megnyomásával a felhasználó kitörlésre kerül az adatbázisból! A „*Mégsem*” válasz esetén visszanavigál a program a kilistázott felhasználók oldalára.

57. ábra Felhasználó törlésének megerősítése

3.2.3.3. Kölcsönzések kezelése

Az adminisztrátori felületen válassza a „*Kölcsönzések kezelése*” lehetőséget (lásd 48. ábra). Az oldalon a rendszerben szereplő rendelések adatai fognak megjelenni. amelyek az alábbiak:

- KÖLCSÖNZÉS AZONOSÍTÓJA ÉS DÁTUMA: A kölcsönzés egyedi azonosítószáma és a kölcsönzés kezdetének dátuma.
- KÖLCSÖNZŐ NEVE: A könyvet kikölcsönző felhasználó neve.
- KIKÖLCSÖNZÖTT KÖNYV CÍME: A kikölcsönzött könyv címe
- KÖLCSÖNZÉS LEJÁRATI IDEJE: A könyv visszahozásának legkésőbbi dátuma
- KÖLCSÖNZÉS STÁTUSZA: A könyv kölcsönzési állapotáról nyújt tájékoztatást. „*Kölcsönözve*” állapot arra utal, hogy a kölcsönzés nem fejeződött be, a példány a felhasználónál van. „*Visszahozva, lezárt*” státusz arról informál, hogy a könyv visszahozatala megtörtént és a rendszergazda azt rögzítette is. Sikeres visszahozatal.
- LEZÁRÁS: Egy olyan funkció, amely az adminisztrátornak lehetőséget ad arra, hogy beállítsa a sikeres visszahozatalt. Amennyiben rákattint a „*Lezárás*” gombra, a gomb eltűnik és a KÖLCSÖNZÉS STÁTUSZA mezőben szereplő érték „*Visszahozva, lezárt”* státuszra vált.
- KÖLCSÖNZÉS TÖRLÉSE: Ennek a funkciónak a használatával az adminisztrátor ki tudja törölni az adatbázisból a kölcsönzést.



58. ábra Kölcsönzések kezelése oldal

Lezárás

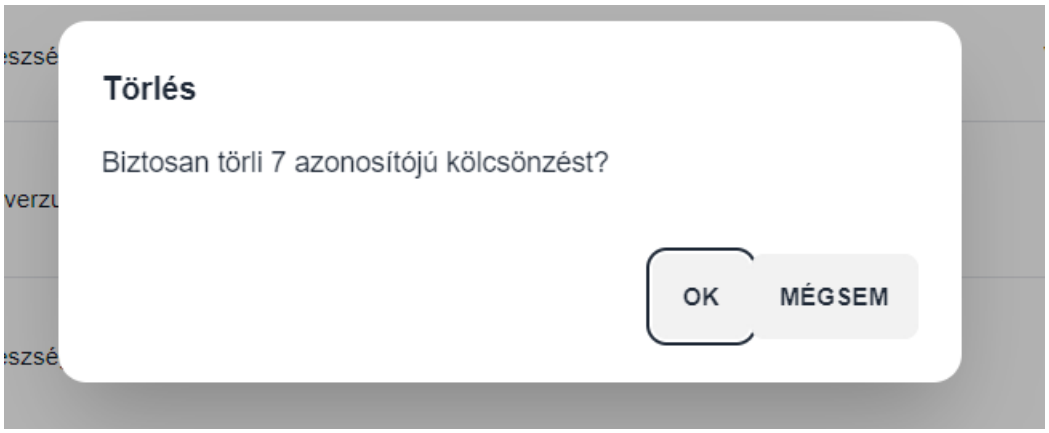
Az admin felhasználói jogosultsággal rendelkező személy be tudja állítani a rendelések kapcsán azt, hogy a kölcsönző a szabályoknak megfelelően, határidőre visszahozta-e a könyvet. A megjelenő rendeléseknél a „Lezárás” oszlopban kattintson a zöld színű „*Lezárás*” gomba. A „Kölcsönzés státusza” oszlop értéke „*Kölcsönözve*” állapotról „*Visszahozva, lezárt*” állapotra módosul.

59. ábra Lezárás menete

Rendelés törlése

A lezárást követően a rendszergazda törölni tudja a rendelést az adatbázisból. A megjelenő rendelési listáknál a „Kölcsönzés törlése” oszlopban talál egy „*Töröl*” feliratot. Amennyiben szeretné törölni a megrendelést kattintson rá.

A rendszer biztonsági okokból megerősítést fog kérni a művelet elvégzéséhez! Az „Ok” gomb megnyomásával a rendelés kitörlésre kerül az adatbázisból! A „Mégsem” válasz esetén visszanavigál a program a kilistázott rendelések oldalára.



60. ábra Rendelés törlésének megerősítő üzenete

3.2.4. Információ kérése

Amennyiben további információra lenne szüksége a program működése kapcsán, esetlegesen szeretné további funkciókkal is gazdagítani a meglévő könyvkölcsönző alkalmazást, kérem, szíveskedjen felvenni a kapcsolatot a program készítéséért felelős személyekkel.

Balog Csaba fejlesztő – Tel. 06 70 382 6597 email. bastelnkft@gmail.com

Hoffer Áron fejlesztő – Tel. 06 30 234 6000 email. hofferaron86@gmail.com

4. Összegzés

Megállapíthatjuk, hogy a könyvkölcsönző webapplikációnk a tervezett kezdeti elvárásainknak megfelelően került elkészítésre. Mind az „extra” hozzáférési jogosultsággal nem rendelkező felhasználó, mind az adminisztrátor képes elvégezni a tevékenységeiket. A webapplikációnk elkészítése során az alábbi tapasztalatokkal gyarapodtunk.

A projektünk kezdeti, tervezési fázisa sokáig tartott. A „szoftverfejlesztő párosok” kijelölése is több időt vett igénybe. Az adatbázis összeállítása, megbeszélése is nehézkesnek mutatkozott a tervezés fázisában, mivel a honlap alapvető funkcionalitását határoztuk meg (rendelés folyamata, továbbá némi adminisztratív tevékenységek ellátása), így az adatbázisunk többszöri módosításon is átesett. A nem megfelelő adattípusok meghatározása, az újabb ötletek bevezetése is nehezítette az implementációs fázist. Meg kellett tapasztaljuk, hogy akár a legkisebb részegység változtatása is a projekt további módosítását vonja maga után több részénél. Mivel a könyvkölcsönző alkalmazásunkat munka mellett készítettük el, eleinte nehézkesnek bizonyultak a közös egyeztetések időpontkijelölései, azonban ezek a projekt előrehaladtával megoldódni látszódtak. A vizsgaremekünk párban történő elkészítése pici betekintést adhatott egy valós szoftverfejlesztői munkakörbe. Átélhettük azokat a kihívásokat, amelyek akkor keletkeznek, ha nem csak egyedül egymaga dolgozik egy fejlesztő egy adott projekten. A „*társas kódolás*” még több egyeztetést, megbeszélést igényel, legalábbis kezdő szinten biztosra mondhatjuk. A fent részletezett kezdeti nehézségek ellenére bátran kijelenthetjük a könyvkölcsönző projektünket határidőre sikeresen befejeztük. Nyilván vannak benne hiányosságok, illetve ahogy egyik tanárunk említette órája alatt:

„Egy program három dolgot biztosan tartalmaz. Egy ciklust, egy elágazást és egy hibát.” 😊

Köszönetet szeretnénk nyilvánítani a tanárainknak, akik mindig készséggel segítettek, ha problémáink akadtak, akik óráikon rávezettek a „fejlesztői gondolkodásmód útjára”, akik bátorítottak, bíztattak bennünket. Továbbá szeretnénk köszönetet mondani családtagjainknak, szeretteinknek, akik készséggel álltak mellettünk és elnézték nekünk, hogy sok esti órát a gép előtt töltöttünk.

A vizsgaremekünk elkészítése során frontend oldalon 25 komponens lett létrehozva (kommenteket is bele értve összesen 2240 sor), backend oldalon 10 scriptfájl (kommentekkel 1428 sor).

5. Irodalomjegyzék

A vizsgaremekünk elkészítése során az alábbi segédanyagokat használtuk fel:

- https://www.emailjs.com/docs/
- https://regex101.com/
- https://www.w3schools.com/MySQL/default.asp
- https://www.geeksforgeeks.org/nodejs/
- https://www.youtube.com/playlist?list=PLC3y8-rFHvwgg3vaYJgHGnModB54rxOk3
- https://www.youtube.com/watch?v=f2EqECiTBL8
- https://tailwindui.com/documentation
- https://tailwindflex.com/
- https://www.geeksforgeeks.org/express-cookie-parser-signed-and-unsigned-cookies/
- https://www.geeksforgeeks.org/express-js/
- https://stackoverflow.com/questions/57206675/how-to-fix-cors-error-with-credentials-include
- https://www.geeksforgeeks.org/how-to-deal-with-cors-error-in-express-node-js-project/
- https://www.npmjs.com/package/react-toastify
-