Dokumentáció

Bevezetés, a téma ismertetése

A könyvtárkezelő alkalmazás egy modern, felhasználóbarát rendszer, amelynek célja a könyvtári állomány átlátható nyilvántartásának biztosítása, valamint a kölcsönzési és visszavételi folyamatok egyszerűsítése. Az alkalmazás Laravel alapú backenddel, Angular frontenddel és XAMPP adatbázissal működik, így stabil és hatékony megoldást nyújt mind a diákok, mind az adminisztrátorok számára. A rendszer intuitív kezelőfelületet biztosít, amely lehetővé teszi a gyors és hatékony adatkezelést, csökkentve ezzel az adminisztrációs terheket és az esetleges hibalehetőségeket. A projekt elsődleges célja egy olyan webalapú könyvtárkezelő rendszer létrehozása volt, amely lehetővé teszi a könyvek egyszerű nyilvántartását, a kölcsönzések kezelését és a visszavételi kérelmek gyors feldolgozását. Napjainkban számos oktatási intézmény még mindig hagyományos, papíralapú vagy elavult szoftverekkel működő nyilvántartási rendszereket használ, amelyek lassúvá és kevésbé hatékonnyá teszik a könyvtári ügyintézést. Ez nemcsak az adminisztrátorok munkáját nehezíti meg, hanem a diákok számára is kevésbé kényelmes és átlátható könyvkölcsönzési folyamatot eredményez. Az általunk fejlesztett rendszer ezt a problémát kívánja orvosolni egy korszerű, digitális megoldással, amely automatizálja a kölcsönzési folyamatokat, valamint lehetővé teszi a valós idejű adatkezelést és keresést.

A téma választásának indoklása

Az alkalmazás fejlesztésének egyik fő motivációja az volt, hogy iskolánkban jelenleg nincs olyan digitális könyvtárkezelő rendszer, amely megfelelően támogatná a könyvkölcsönzések és a visszavételek nyomon követését. A meglévő nyilvántartási módszerek gyakran manuálisak és időigényesek, ami nemcsak a könyvtárosok számára jelent többletmunkát, hanem a diákok számára is hosszadalmas várakozási időt eredményezhet. Az iskolai könyvtárnak szüksége volt egy olyan rendszerre, amely megkönnyíti a diákok számára a könyvek elérhetőségének ellenőrzését, valamint az egyszerű és gyors kölcsönzési és visszavételi folyamatokat. Ezért döntöttünk úgy, hogy létrehozunk egy saját fejlesztésű könyvtárkezelő rendszert, amely nemcsak az iskolánk igényeire szabható, hanem hosszú távon is fenntartható és bővíthető. Az alkalmazás olyan funkciókkal rendelkezik, amelyek lehetővé teszik a könyvek gyors keresését, az aktuális kölcsönzési állapotok nyomon követését, valamint az egyszerű visszavételi kérések benyújtását. Ezen kívül a könyvtári dolgozók számára biztosított egy adminisztrációs felület, amely lehetővé teszi a könyvállomány kezelését és a kölcsönzések egyszerű nyomon követését. A projekt megvalósítása lehetőséget adott arra, hogy modern webfejlesztési technológiákkal dolgozzunk, és olyan szakmai tapasztalatokat szerezzünk, amelyek a jövőben is hasznosak lesznek számunkra. A Laravel keretrendszer segítségével egy biztonságos és jól strukturált backend készült, amely biztosítja a rendszer megbízható működését és skálázhatóságát. Az Angular frontend dinamikus és reszponzív felhasználói élményt nyújt, amely lehetővé teszi a diákok és az adminisztrátorok számára a zökkenőmentes és intuitív használatot. Az XAMPP környezet biztosította az adatbázis kezelését, lehetővé téve a gyors és hatékony adattárolást, miközben garantálja az adatintegritást és a keresési funkciók optimalizálását. Összességében a projekt célja egy hatékony, skálázható és felhasználóbarát könyvtárkezelő rendszer létrehozása volt, amely az oktatási intézmények számára egy digitális, korszerű megoldást kínál a könyvtári adminisztrációs folyamatok egyszerűsítésére és optimalizálására.

Felhasznált Technológiák

* **Frontend:** Angular 18, TypeScript, Bootstrap 5, Html 5, CSS
* **Backend:** Laravel, PHP 8
* **Adatbázis:** MySQL
* **Autentikáció:** Token alapú (JWT)
* **Fordítás:** ngx-translate
* **Egyéb:** Dia, phpMyAdmin, Képmetsző, VSCode, Insomnia, Github

Fejlesztőkörnyezet

**Hardver**

* **Fejlesztői gépek:**
  + **Frontend:**
    - **Processzor (CPU):** Intel® Core™ i7-11800H nyolcmagos processzor, alapórajele 2.30 GHz, turbó frekvenciája elérheti a 4.60 GHz-et.
    - **Memória (RAM):** 16 GB DDR4 SDRAM.
    - **Tárolókapacitás:** 1 TB PCIe NVMe SSD, 1 TB HDD.
    - **Videokártya (GPU):** NVIDIA® GeForce RTX™ 3070 dedikált grafikus kártya 8 GB GDDR6 memóriával.
    - **Kijelző:** 15,6 hüvelykes Full HD (1920 x 1080) IPS panel, 144 Hz-es frissítési rátával.
    - **Operációs rendszer:** Windows 11 Home 64-bit.
    - **Operációs rendszer:** Windows 11
    - **Választás indoklása:** Nagy teljesítőképességű laptop, amely megfelel a modern webalkalmazás-fejlesztés követelményeinek.
  + **Backend:**
    - **Processzor (CPU):** Intel® Core™ i5-6400, alapórajele 2.70 GHz, 2.71 GHz
    - **Memória (RAM):** 16 GB.
    - **Operációs rendszer:** Windows 10 Enterprise.
    - **Operációs rendszer:** Windows 10

**Szoftverek**

* **Fejlesztői eszközök:** VS Code, Insomnia, Xampp
* **Verziókezelés:** Git, GitHub
* **Választás indoklása:** Ezek a technológiák biztosítják a hatékony fejlesztést és együttműködést.

Kialakított Adatszerkezet

A képen szöveg, diagram, Párhuzamos, Tervrajz látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

Fejlesztési Lehetőségek

A rendszer további fejlesztési lehetőségei:

* **API Rate Limiting:** Biztonsági fejlesztések
* **Mobilalkalmazás:** Natív Android és iOS támogatás
* **QR-kódos könyvkölcsönzés:** Minden könyv kapna egy QR-kódot, amelyet a diákok beolvashatnak a gyorsabb kölcsönzés érdekében.
* **Mobilalkalmazás fejlesztése:** Egy natív Android/iOS alkalmazás fejlesztése, amely megkönnyíti a kölcsönzést, értesítéseket küld, és offline módban is elérhetővé teszi a könyvlistát.
* **Automatikus emlékeztetők és értesítések:** Emailben vagy push értesítésben figyelmeztetné a diákokat a kölcsönzési határidő közeledtére vagy a késedelmi díjakra.
* **Megerősítő email:** regisztrációt követően megerősítő emailt küldeni a felhasználók részére.
* **Könyvajánló rendszer:** Egy mesterséges intelligencia alapú ajánlórendszer, amely a kölcsönzési előzmények és a könyvek értékelése alapján javasol új könyveket a felhasználóknak.
* **Vonalkódos könyvkölcsönzés:** Minden könyv kapna egy vonal-kódot, és minden diákigazolvány rendelkezi eggyel, így a könyvkiadási folyamat is gyorsulna..
* **Személyre szabott olvasási statisztikák:** A diákok és a könyvtárosok részletes statisztikákat láthatnának az elolvasott könyvekről, a legnépszerűbb műfajokról és az olvasási szokásaikról.
* **API integráció más rendszerekkel:** Az iskola egyéb rendszereivel való összekapcsolás (pl. e-learning platform, tanulmányi rendszer), hogy a diákok könyvtári adatai és az oktatási adatok összekapcsolhatók legyenek.
* **Többnyelvű támogatás:** A felület több nyelven is elérhető lenne, így külföldi diákok is könnyen használhatnák a rendszert.
* **Chatbot alapú ügyfélszolgálat:** Egy mesterséges intelligenciával működő chatbot, amely gyorsan válaszol a gyakran ismételt kérdésekre, például könyvkölcsönzéssel vagy határidőkkel kapcsolatban.
* **Könyvkölcsönzés előzményeinek exportálása:** A felhasználók lementhetik vagy exportálhatják PDF/Excel formátumban a saját kölcsönzési előzményeiket, ami segíthet az olvasási napló vezetésében.
* **Könyvek értékelése és véleményezése:** A diákok értékelhetik és kommentálhatják a kölcsönzött könyveket, ezzel segítve másokat a választásban, valamint közösséget építve a könyvtár köré.

Frontend fejlesztői dokumentáció

***Üdvözlő oldal***

Az üdvözlő oldal a home komponens „home.component.html” -ben található és egy általános üdvözlő oldalként funkcionál, amely minden látogató számára elérhető.   
A html részben tárolunk egy rövid bemutatkozó szöveget a BZSH Külkereskedelmi Technikumról, amin keresztül megismerhetjük az iskolát.

A képen szöveg, képernyőkép, Betűtípus, tervezés látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

***Bejelentkező oldal***

A bejelentkezési funkció lehetővé teszi a felhasználók számára, hogy hitelesítsék magukat az alkalmazásban. A belépést követően a rendszer ellenőrzi a felhasználó szerepkörét, és ennek megfelelően navigálja őket a megfelelő oldalra.

Bejelentkezési Folyamat

1. A felhasználó megnyitja a bejelentkezési oldalt.
2. Kitölti az e-mail és jelszó mezőket.
3. Ha az adatok érvényesek, a rendszer a login() függvényen keresztül hitelesíti a felhasználót.
4. A sikeres bejelentkezés után a felhasználó szerepkörét (role) az alkalmazás ellenőrzi, és az alábbi oldalak egyikére navigálja:
   * libraryUser (Ha a szerepkör: student)
   * libraryAdmin (Ha a szerepkör: admin)
5. Sikertelen bejelentkezés esetén egy hibaüzenet jelenik meg.

A képen szöveg, képernyőkép, Betűtípus, Márka látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

Elérhetőség

A regisztrációs oldal minden látogató számára elérhető.

* Sikeres hitelesítés után a felhasználók a szerepkörük alapján más oldalakra kerülnek átirányításra:
  + Diákok (student): libraryUser
  + Adminisztrátorok (admin): libraryAdmin
* Ha a felhasználó már be van jelentkezve, a rendszer automatikusan a megfelelő oldalra irányítja.

HTML Felépítés

* E-mail mező:
  + Az [(ngModel)]="email" adatkötést használja.
  + Validáció: kötelező (required) és e-mail formátum ellenőrzés (pattern).
  + Hibaüzenetet küld a rendszer, ha a mező üres vagy a formátum nem megfelelő.
* Jelszó mező:
  + Az [(ngModel)]="password" adatkötést használja.
  + Validáció: kötelező (required) és erős jelszó ellenőrzés (pattern).
  + Hibaüzenetet küld a rendszer, ha a mező üres vagy a formátum nem megfelelő.
* Gombok:
  + A bejelentkezési gomb le van tiltva (disabled), ha a mezők hibásak vagy üresek.
  + A regisztrációs link az /register oldalra navigál.

TypeScript Működés

* login() függvény:
  + Meghívja az auth.login(email, password) metódust, amely a AuthenticationService osztályon keresztül történik.
  + Ha a bejelentkezés sikeres, ellenőrzi a localStorage.getItem('role') értékét:
    - Ha student, akkor átirányítja a libraryUser oldalra.
    - Ha admin, akkor átirányítja a libraryAdmin oldalra.
  + Ha hiba történik, egy alert üzenetet jelenít meg.
* Adatok kezelése:
  + Az email és password változók az ngModel segítségével kötődnek a beviteli mezőkhöz.
  + Az emailControl és passwordControl változók az űrlap validációját segítik.

Függvények Részletezése

* login():
  + Meghívja az auth.login() metódust az AuthenticationService szolgáltatásból.
  + A felhasználó által megadott email és password paramétereket továbbítja a hitelesítéshez.
  + Sikeres bejelentkezés esetén a felhasználó szerepkörét (role) a localStorage-ból olvassa ki.
  + Az adott szerepkör alapján a megfelelő oldalra irányítja a felhasználót.
  + Ha hiba történik, egy alert üzenetet jelenít meg.

Navigáció és Szerepkör Ellenőrzés

A bejelentkezés után a felhasználó jogosultságát a localStorage.getItem('role') érték alapján határozza meg, és az adminokat, illetve diákokat a megfelelő könyvtár oldalra irányítja.

Függőségek

* AuthenticationService: A bejelentkezési hitelesítést végzi.
* Router: A megfelelő oldalra navigálja a felhasználót.

Összegzés

Ez a bejelentkezési funkció egy egyszerű, de hatékony hitelesítési megoldást biztosít az alkalmazás számára. A validációs szabályok és az automatikus átirányítás biztosítják a felhasználói élmény gördülékenységét és biztonságát.

***Regisztrációs oldal***

A regisztrációs funkció lehetővé teszi az új felhasználók számára, hogy létrehozzanak egy fiókot az alkalmazásban. A felhasználók megadják a szükséges adatokat, és sikeres regisztráció után a rendszer automatikusan átirányítja őket a bejelentkezési oldalra.

**Regisztrációs Folyamat**

1. A felhasználó megnyitja a regisztrációs oldalt.
2. Kitölti az alábbi mezőket:
   * Felhasználónév
   * Email cím
   * Oktatási azonosító szám
   * Jelszó
   * Jelszó megerősítés
3. Az adatok kitöltése után a rendszer validálja azokat.
4. Ha minden adat helyes, a register() függvény elküldi az adatokat az AuthenticationService-nek.
5. Sikeres regisztráció esetén a felhasználó átirányításra kerül a bejelentkezési oldalra.
6. Sikertelen regisztráció esetén egy hibaüzenet jelenik meg.

**A képen szöveg, képernyőkép, Betűtípus, szoftver látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.**

**Elérhetőség**

* A regisztrációs oldal minden látogató számára elérhető.

**HTML Felépítés**

* **Felhasználónév mező:**
  + Az [(ngModel)]="username" adatkötést és a formControlName="usernameControl" használja.
  + Validáció: kötelező (required).
  + **Hibaüzenetet küld a rendszer, ha a mező üres vagy a formátum nem megfelelő.**
* **Email mező:**
  + Az [(ngModel)]="email" adatkötést és a formControlName="emailControl" használja.
  + Validáció: kötelező (required) és e-mail formátum ellenőrzés (pattern).
  + **Hibaüzenetet küld a rendszer, ha a mező üres vagy a formátum nem megfelelő.**
* **Oktatási azonosító mező:**
  + Az [(ngModel)]="studentId" adatkötést és a formControlName="studentIdControl" használja.
  + Validáció: kötelező (required) és egyedi formátum (pattern), amely 11 számjegyből áll és 7-es számmal kezdődik.
  + **Hibaüzenetet küld a rendszer, ha a mező üres vagy a formátum nem megfelelő.**
* **Jelszó mező:**
  + Az [(ngModel)]="password" adatkötést és a formControlName="passwordControl" használja.
  + Validáció: kötelező (required) és erős jelszó ellenőrzés (pattern), amely tartalmaz kisbetűt, nagybetűt és számot, legalább 8 karakter hosszú.
  + **Hibaüzenetet küld a rendszer, ha a mező üres vagy a formátum nem megfelelő.**
* **Jelszó megerősítés mező:**
  + Az [(ngModel)]="passwordConfirm" adatkötést és a formControlName="passwordConfirmControl" használja.
  + Validáció: kötelező (required) és egyezés ellenőrzés a password mezővel.
  + **Hibaüzenetet küld a rendszer, ha a mező üres vagy a formátum nem megfelelő.**

**TypeScript Működés**

* **register() függvény:**
  + Meghívja az auth.register() metódust az AuthenticationService szolgáltatáson keresztül.
  + Átadja a következő adatokat:
    - username
    - email
    - studentId
    - password
    - passwordConfirm
  + Sikeres regisztráció esetén átirányítja a felhasználót a bejelentkezési oldalra.
  + Ha hiba történik, egy megfelelő hibaüzenetet jeleníti meg.
* **Form Validáció:**
  + Az Angular Reactive Forms segítségével biztosított.
  + Az űrlapot a registerForm objektum kezeli.
  + Egyedi validátor (passwordMismatchValidator) ellenőrzi, hogy a két jelszó egyezik-e.

**Függőségek**

* AuthenticationService: A regisztrációs folyamat kezelésére szolgál.
* Router: A megfelelő oldalra navigálja a felhasználót.
* FormGroup és FormControl: Az űrlap mezőinek validálásához szükséges.

**Összegzés**

A regisztrációs funkció biztosítja az új felhasználók számára a fiók létrehozását. A validációs szabályok biztosítják az adatok helyességét, míg az automatikus átirányítás segíti a felhasználók gördülékeny belépését az alkalmazásba.

***Névjegy oldal***

Az névjegy oldal az about komponens „about.component.html” -ben található és egy általános üdvözlő oldalként funkcionál, amely minden látogató számára elérhető  
A html részben található meg a BZSH Külkereskedelmi Technikum közismereti információi, amin keresztül megtudhatjuk az iskola általános helyszínét, adatait és elérhetőségeit. Illetve egy rövid leírást, hogy kik és miért készítették a weboldalt.

A képen szöveg, képernyőkép, Betűtípus, tervezés látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

***Navigációs sáv***

A navigációs sáv lehetővé teszi a felhasználók számára, hogy könnyedén navigáljanak az alkalmazás különböző részeire. A sáv dinamikusan változik a felhasználó bejelentkezésének állapota és szerepköre alapján. Emellett a felhasználók válthatnak nyelvet is a kínált lehetőségek között.

**Elérhetőség**

A navigációs sáv minden felhasználó számára elérhető az alkalmazás bármely oldalán, és az alábbiakat tartalmazza:

* Kezdőlap
* Bejelentkezés (csak be nem jelentkezett felhasználóknak)
* Kikölcsönzött könyvek (csak diákoknak)
* Könyvek, Diákok, Felhasználók, Könyvtár (csak adminoknak)
* Névjegy
* Kijelentkezés (csak bejelentkezett felhasználóknak)
* Nyelvváltás (magyar és angol között)

**HTML Felépítés**

* **Hamburger Menü:**
  + A képen szöveg, képernyőkép, Betűtípus látható

    Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.A hamburger-menu gombra kattintva a menü nyílik vagy záródik be. Ez nagyobb méretű nézetek (pl.: desktop) esetében nem jelenik meg, leginkább kisebb nézetekben van jelen (pl.: mobil nézet).
  + A menü elemei dinamikusan jelennek meg vagy tűnnek el attól függően, hogy a felhasználó be van-e jelentkezve és milyen szerepköre van (diák vagy admin).
* **Nyelvváltó Gombok:**
  + Lehetővé teszi a felhasználók számára, hogy a weboldalt magyar és angol nyelv között válthassák.
  + A switchLanguage() függvény segítségével a felhasználó nyelve dinamikusan frissül.
  + Funkcionalitás szempontjából a gombok úgy működnek, hogy ha az oldal magyar nyelven van, a gomb EN felirattal fogja jelezni a felhasználók számára, hogy angol nyelvre tudnak váltani és ugyanez fordítva is működik a HU feliratú gombbal.





* **Linkek:**
  + **Kezdőlap:** A home oldalra navigál.
  + **Bejelentkezés:** Csak akkor jelenik meg, ha a felhasználó nincs bejelentkezve, és az /login oldalra vezet.
  + **Kikölcsönzött könyvek:** Csak diákok számára érhető el, és az /libraryUser oldalra navigál.
  + **Könyvek, Diákok, Felhasználók, Könyvtár:** Csak adminok számára elérhetők, és ezek a megfelelő adminisztrátori oldalra vezetnek.
  + **Névjegy:** Az /about oldalra navigál.
  + **Kijelentkezés:** A bejelentkezett felhasználók számára elérhető, és a logOut() függvény segítségével kijelentkezik.

**TypeScript Működés**

* **toggleMenu() függvény:**
  + Változtatja a menuOpen állapotát, amely a hamburger menü nyitott/zárt állapotát szabályozza.
* **closeMenu() függvény:**
  + Zárja a hamburger menüt, ha az meg van nyitva.
* **switchLanguage() függvény:**
  + Frissíti az alkalmazás nyelvét a configService.switchLanguage() hívásával, és változtatja a currentLang változót.
  + A két nyelv közötti váltás magyar (hu) és angol (en) között történik.
* **logOut() függvény:**
  + Meghívja az auth.logout() metódust, amely a felhasználót kijelentkezteti, és a navigációs sávon megjelenő linkek frissülnek.
* **ngOnInit() függvény:**
  + Az alkalmazás indításakor ellenőrzi, hogy a felhasználó be van-e jelentkezve az auth.isLoggedIn() segítségével.

**Navigáció és Szerepkör Ellenőrzés**

A navigációs sáv dinamikusan változik a felhasználó bejelentkezési állapotától és szerepkörétől függően:

* Ha a felhasználó **nem jelentkezett be**, a "Bejelentkezés" link lesz elérhető.
* Ha a felhasználó **diák**, akkor a "Kikölcsönzött könyvek" link lesz elérhető.
* Ha a felhasználó **admin**, akkor az adminisztrátori linkek (Könyvek, Diákok, Felhasználók, Könyvtár) válnak elérhetővé.

**Függőségek**

* AuthenticationService: A felhasználó bejelentkezésének állapotát ellenőrzi.
* ConfigService: A nyelv váltásáért felelős szolgáltatás.

**Összegzés**

Ez a navigációs sáv egyszerű és intuitív módot biztosít a felhasználóknak az alkalmazás különböző oldalaira történő navigálásra, a felhasználói szerepkörök figyelembevételével, és lehetővé teszi a nyelvváltást is.

***Diák felhasználói oldal***

A libraryListUser komponens az alkalmazás felhasználói oldalát kezeli, amely lehetővé teszi a felhasználók számára a kikölcsönzött könyvek megtekintését és visszaadását.

**HTML részletezése**

1. **Felhasználó Neve**:
   * Az {{ userName }} a felhasználó nevét jeleníti meg. A név az localStorage-ban tárolódik és a ngOnInit() során töltődik be.
   * A title="{{ userName }}" attribútum segít a felhasználó számára információt adni a névről a tooltip segítségével.
2. **Táblázat a Könyvek Adatainak Megjelenítésére**:
   * A táblázat a felhasználó által kikölcsönzött könyvek adatait jeleníti meg, beleértve a szerzőt, címet, raktári számot, árat, státuszt és a visszaadás gombot.
   * Az \*ngFor direktíva segítségével iterálunk a loanArray tömbön, amely tartalmazza a felhasználó által kikölcsönzött könyveket.
3. **Könyvek Visszaadása**:
   * A returnBook() függvény meghívása akkor történik, amikor a felhasználó a "Visszaadás" gombra kattint. A gomb csak akkor érhető el, ha a könyv státusza nem "függőben" vagy "requested\_return".

**TypeScript Működés**

**Konstruktor és Kezdő Adatok**

* **Felhasználó Neve**: A userName változó a localStorage-ból van beállítva, és a felhasználó nevét tárolja.
* **Könyvek és Könyvkölcsönzések Betöltése**: A konstruktorban két HTTP kérés történik:
  + A db.getStudentBook() lekéri a felhasználó könyveit (bookArray), és betölti őket a komponensbe.
  + A db.getMyLoans() lekéri a felhasználó aktuális kölcsönzéseit (loanArray).

**Funkciók**

1. **translateStatus(status: string)**:
   * Ez a függvény a könyv kölcsönzési státusza alapján lefordítja a státuszt egy nyelvi kulcs segítségével, és visszaadja a megfelelő fordítást.
2. **filterLoans()**:
   * A loanArray tömböt szűri, hogy csak azok a könyvek jelenjenek meg, amelyek státusza nem "returned" vagy "visszaadott". Ez biztosítja, hogy csak az aktuálisan kikölcsönzött könyvek jelenjenek meg.
3. **getBookByInventoryNumber(inventory\_number: string)**:
   * A könyvek raktári számát használva megkeresi a megfelelő könyvet a bookArray-ban, és visszaadja a könyv típusát (vagy null, ha nem található).
4. **returnBook(id: string)**:
   * Ez a függvény kezeli a könyv visszaadását. A db.returnBook(id) HTTP hívás meghívása után frissíti a képernyőt (újratöltés történik), ha a visszaadás sikeres.

**Összegzés**

A komponens a felhasználói könyvtári felületet biztosítja, amely lehetővé teszi a felhasználók számára a kikölcsönzött könyvek megtekintését, a státuszuk ellenőrzését és a könyvek visszaadását.

Az Angular ngFor direktívája segítségével dinamikusan jeleníti meg a kölcsönzött könyvek listáját.

Az alkalmazás a TranslateService-et használja a többnyelvű támogatáshoz, így a könyvek státusza és egyéb információk lefordíthatók különböző nyelvekre.

***Könyv kölcsönzési oldal***

A libraryListAdmin komponens az adminisztrátori felületet kezeli a könyvtári alkalmazásban, ahol az adminisztrátorok kezelhetik a könyvek kiadását, elfogadását, elutasítását, valamint kereshetnek a kölcsönzött könyvek között.

**HTML részletezése**

1. **Keresési mező**:
   * A felhasználók szűrhetik a kölcsönzéseket a keresőmező segítségével. Az ngModel a searchText változóra van kötve, és az input esemény során a filterLoans() függvény hívódik meg.
   * A placeholder szöveg a TranslateService segítségével van lefordítva, és az adminisztrátorok számára mutatja a keresési utasítást.
2. **Könyvek táblázata**:
   * A táblázat minden kölcsönzött könyvet megjelenít, amely tartalmazza az oktatási azonosítót, a raktári számot és a könyv státuszát. Ezen kívül az adminisztrátorok képesek kiadni, elfogadni vagy elutasítani egy könyvet.
   * A táblázat minden sorában a következő gombok találhatók:
     + **Kiadás**: Egy modal ablakot nyit, ahol az adminisztrátor beírhatja a felhasználó oktatási azonosítóját és a könyv raktári számát.
     + **Elfogadás**: A kölcsönzött könyvet jóváhagyja, amennyiben nem "visszaadott" állapotú.
     + **Elutasítás**: A könyvet elutasítja, ha még nem lett visszaadva és nincs másik kölcsönzés alatt.
3. **Modal ablak**:
   * A modal ablak a könyv kiadására szolgál, ahol az adminisztrátor megadhatja az oktatási azonosítót és a raktári számot. A createLoan() függvény hívódik meg a könyv kiadása során.

**TypeScript Működés**

**Konstruktor és Kezdő Adatok**

* **Könyv Kölcsönzések Betöltése**:
  + A konstruktorban a db.getLoans() HTTP kérés hívódik meg, amely lekéri az összes kölcsönzést (loanArray). A filteredLoans tömb üres, és a keresési eredmények itt kerülnek tárolásra.

**Funkciók**

1. **translateStatus(status: string)**:
   * Ez a függvény a könyv kölcsönzési státusza alapján lefordítja a státuszt egy nyelvi kulcs segítségével, és visszaadja a megfelelő fordítást.
2. **filterLoans()**:
   * A filterLoans() függvény a loanArray tömböt szűri a searchText alapján. Ha a keresési szöveg üres, a filteredLoans üres marad, egyébként a kölcsönzéseket szűri a keresett szövegre.
3. **createLoan()**:
   * Az adminisztrátor a modal ablakban megadhatja a kölcsönzéshez szükséges adatokat. A createLoan() függvény hívásával a könyvet kiadja, és az adatokat a backendnek továbbítja.
   * A window.location.reload() frissíti az oldalt a kölcsönzés létrehozása után.
4. **approveLoan(id: string)**:
   * A approveLoan() függvény egy könyvet jóváhagy, ha a státusza megfelelő. A sikeres művelet után az oldal frissül.
5. **forceApproveLoan(id: string)**:
   * A forceApproveLoan() függvény kényszerített jóváhagyást végez, ha a könyv valamilyen különleges körülmények között jóváhagyható. A sikeres művelet után az oldal frissül.
6. **rejectLoan(id: string)**:
   * A rejectLoan() függvény elutasítja a könyv kölcsönzését, és az oldal frissül a művelet sikeres végrehajtása után.

**Összegzés**

Ez az adminisztrátori könyvtári felület a könyvek kiadását, elfogadását, elutasítását, valamint a kölcsönzések keresését kezeli. A komponens a következőket biztosítja:

* **Keresési funkció** a kölcsönzések szűrésére.
* **Kiadás** gomb, amely egy modal ablakban lehetővé teszi új kölcsönzési rekordok létrehozását.
* **Elfogadás** és **Elutasítás** gombok, amelyek segítik a kölcsönzések kezelését.

**Státuszok fordítása** a TranslateService segítségével többnyelvű támogatással.

***Felhasználói táblázat***

A users komponens az adminisztrátori felületet kezeli a felhasználók kezelésére. Az adminisztrátorok listázhatják, szerkeszthetik, létrehozhatják és inaktiválhatják a felhasználókat.

**HTML részletezése**

**1. Keresési mező**

* Az adminisztrátorok szűrhetik a felhasználókat a keresőmező segítségével.
* Az ngModel a searchText változóra van kötve, és az input esemény során a filterUsers() függvény hívódik meg.
* A placeholder szöveg a TranslateService segítségével van lefordítva.

**2. Felhasználók táblázata**

* A táblázat minden regisztrált felhasználót megjelenít, beleértve a nevüket, e-mail címüket, jogosultságukat és állapotukat.
* A ngFor segítségével jelennek meg az adatok a userArray vagy a filteredUsers tömb alapján.
* A következő gombok találhatók a táblázat soraiban:
  + **Szerkesztés**: Egy modal ablakot nyit meg, ahol az adminisztrátor módosíthatja a felhasználó adatait.
  + **Állapot módosítása**: A statusUser() függvény segítségével aktiválja vagy inaktiválja a felhasználót.
  + **Hozzáadás**: Egy új modal ablakot nyit, ahol az adminisztrátor új felhasználót rögzíthet.

**3. Modal ablakok**

* **Új felhasználó létrehozása**:
  + A felhasználók megadására szolgál, ahol az adminisztrátor beírhatja a nevet, az oktatási azonosítót és az e-mail címet.
  + A createUser() függvény hívódik meg a mentéskor.
* **Felhasználó szerkesztése**:
  + A szerkesztő modal ablakba a kiválasztott felhasználó adatai töltődnek be.
  + A modifyUser() függvény módosítja az adatokat.

**TypeScript Működés**

**Konstruktor és Kezdő Adatok**

* Az adatok betöltése a db.getUser() függvény segítségével történik, amely lekéri a felhasználók listáját és eltárolja a userArray változóban.
* Az editUsers állapotváltozó segítségével az adminisztrátor átválthat szerkesztési módba.

**Funkciók**

1. **translateStatus(status: string)**
   * A felhasználói státusz (aktív/inaktív) lefordítására szolgál.
   * A TranslateService segítségével nyelvi kulcs alapján adja vissza az adott státusz fordítását.
2. **translateRole(role: string)**
   * A felhasználó szerepkörét fordítja le (admin vagy student).
3. **filterUsers()**
   * A keresési mezőben beírt szöveg alapján szűri a felhasználókat.
   * Ha nincs keresési szöveg, az eredeti lista marad érvényben.
4. **createUser()**
   * A db.addUser() hívásával új felhasználót ad hozzá.
   * Sikeres hozzáadás után frissíti az oldalt (window.location.reload()).
5. **setSelectedUser(user: any)**
   * Az adott felhasználó adatait átmásolja a selectedUser változóba a szerkesztő modal ablak megnyitásakor.
6. **modifyUser(id: string, name: string, email: string, role: string)**
   * A db.updateUser() segítségével módosítja az adott felhasználó adatait.
   * A módosítás után frissíti az oldalt (window.location.reload()).
7. **statusUser(id: string, currentStatus: string)**
   * A felhasználó státuszának módosítására szolgál (active ↔ inactive).
   * A db.statusSwitchUser() végzi el az állapot módosítását.
   * A változtatás után az oldal frissül (window.location.reload()).

**Összegzés**

Ez az adminisztrátori felhasználókezelő felület biztosítja:

* A felhasználók listázását és keresését.
* Új felhasználók létrehozását egy modal ablak segítségével.
* Felhasználói adatok szerkesztését egy modal ablakon belül.
* A felhasználók aktív/inaktív állapotának módosítását.
* Többnyelvű támogatást a TranslateService segítségével.

***Könyvek táblázat***

A books komponens a könyvtári alkalmazás adminisztrátori felületét kezeli, ahol az adminisztrátorok kezelhetik a könyvek adatbázisát. Lehetőség van könyvek hozzáadására, szerkesztésére, törlésére, valamint keresésre az adatbázisban.

**HTML részletezése**

1. **Keresési mező:**
   * Az adminisztrátorok szűrhetik a könyveket a keresőmező segítségével.
   * Az ngModel a searchText változóra van kötve, és az input eseménynél a filterBooks() függvény hívódik meg.
   * A placeholder szöveg a TranslateService segítségével többnyelvű támogatást biztosít.
2. **Könyvek táblázata:**
   * A táblázat minden könyvet megjelenít a raktári számmal, címmel, szerzővel, árral és példányszámmal.
   * Az adminisztrátorok a következő műveleteket végezhetik el:
     + **Hozzáadás**: Egy modal ablakot nyit meg, ahol az adminisztrátor megadhatja az új könyv adatait.
     + **Szerkesztés**: Egy modal ablakban lehetőség van a könyv adatait módosítani.
     + **Törlés**: Egy gombbal az adminisztrátorok eltávolíthatják a könyvet az adatbázisból.
3. **Modal ablakok:**
   * Az egyik modal ablak **új könyv** hozzáadására szolgál.
   * A másik modal ablak **könyv szerkesztésére** használható.
   * Mindkét modal az ngModel segítségével köti össze az adatokat a TypeScript kóddal.

**TypeScript Működés**

1. **Adatok kezelése:**
   * A komponens több változót tartalmaz a könyvek adatainak tárolására.
   * A bookArray tartalmazza az összes lekért könyvet.
   * A filteredBooks egy ideiglenes tömb a keresési eredményekhez.
   * A selectedBook változó a szerkesztésre kiválasztott könyvet tárolja.
2. **Könyvek betöltése:**
   * Az adatok betöltése a db.getBook() HTTP-kérés segítségével történik.
   * A kapott adatokat a bookArray tömbben tárolja.
3. **Funkciók:**
   * **filterBooks()**
     + A keresőmezőben megadott szöveg alapján szűri a bookArray tartalmát.
     + Ha a keresési szöveg üres, az filteredBooks tömb is üres marad.
   * **createBook()**
     + Az új könyv adatait a backendnek küldi egy HTTP-kérésen keresztül.
     + A sikeres művelet után az oldal frissül.
   * **setSelectedBook(book: any)**
     + Az adott könyvet kijelöli szerkesztésre, és egy másolatot hoz létre a modal ablakhoz.
   * **modifyBook()**
     + A szerkesztett adatokat elküldi a backendnek frissítésre.
     + A sikeres frissítés után az oldal frissül.
   * **deleteBook()**
     + A könyv ID alapján törli azt az adatbázisból.
     + A sikeres törlés után az oldal frissül.

**Összegzés**

A books komponens az adminisztrátori felület része, amely lehetővé teszi a könyvek teljes körű kezelését:

* **Keresési funkció** a könyvek gyors megtalálására.
* **Hozzáadás** modal ablakban új könyvek felvételére.
* **Szerkesztés** lehetőség a meglévő könyvek adatainak módosítására.
* **Törlés** gomb a felesleges könyvek eltávolítására.
* **Többnyelvű támogatás** a TranslateService segítségével.

Ez a komponens biztosítja az adminisztrátorok számára a hatékony könyvkezelést egy modern, Angular alapú könyvtári rendszeren belül.

***Diákok táblázat***

A students komponens az adminisztrátori felületet kezeli a diákok adatainak kezeléséhez.

**HTML részletezése**

**1. Keresési mező:**

* Az **input** mező [(ngModel)]="searchText" kötése lehetővé teszi a felhasználók számára, hogy szűrjenek a diákok között.
* Az **(input)="filterStudents()"** eseményfigyelő automatikusan meghívja a filterStudents() függvényt minden beírt karakter után.
* A placeholder szöveg a TranslateService segítségével van lefordítva, hogy többnyelvű támogatást biztosítson.

**2. Diákok táblázata:**

* A **table** minden diák adatait megjeleníti, beleértve a **nevet, oktatási azonosítót és az állapotot**.
* A táblázat soraiban az alábbi gombok találhatóak:
  + **Hozzáadás**: Egy modal ablakot nyit meg, amelyben az adminisztrátor új diákot rögzíthet.
  + **Szerkesztés**: Egy másik modal ablakot nyit meg a kiválasztott diák adatainak módosításához. A (click)="setSelectedStudent(student)" metódus segítségével beállítja az éppen szerkesztett diák adatait.
  + **Aktiválás/Inaktiválás**: Egy gomb, amely a diák státuszát változtatja meg. A gomb felirata dinamikusan változik az állapotnak megfelelően, a [ngClass] segítségével különböző kinézetet kap.

**3. Modal ablakok:**

* **Új diák hozzáadása:**
  + Egy **input** mezőn keresztül az adminisztrátor megadja a diák nevét és oktatási azonosítóját.
  + A "students.Mentés" | translate gomb meghívja a createStudent() függvényt az új diák létrehozásához.
* **Diák szerkesztése:**
  + Az adminisztrátor módosíthatja a diák nevét.
  + A "students.Mentés" | translate gomb meghívja a modifyStudent() függvényt, amely frissíti a diák adatait a szerveren.

**TypeScript Működés**

**1. Konstruktor és Kezdő Adatok**

* Az HttpService segítségével az alkalmazás betölti a diákok listáját a db.getStudent() metóduson keresztül.
* Az studentArray tömb tartalmazza az összes diákot.
* Az filteredStudents tömb kezeli a keresési eredményeket.

**2. translateStatus(status: string)**

* A függvény az ngx-translate segítségével dinamikusan lefordítja a diák állapotát ("aktív" vagy "inaktív") a megfelelő nyelvre.

**3. filterStudents()**

* Ha a keresőmező üres, akkor az eredménylista (filteredStudents) is kiürül.
* Egyébként az studentArray tömbön belül minden mezőt ellenőriz, és csak azokat a diákokat tartalmazza, amelyek illeszkednek a keresett szövegre.

**4. createStudent()**

* A db.addStudent(this.name, this.edu\_id).subscribe(...) meghívásával az adminisztrátor új diákot adhat hozzá.
* A window.location.reload() frissíti az oldalt a sikeres létrehozás után.

**5. setSelectedStudent(student: any)**

* Ez a függvény az adott diák adatait beállítja egy változóba (selectedStudent), így azok megjelennek a szerkesztési modalban.

**6. modifyStudent(id: string, name: string)**

* Az adminisztrátor frissítheti a diák nevét a db.updateStudent(id, name).subscribe(...) API-hívás segítségével.
* Az oldal frissül a módosítás után.

**7. statusStudent(id: string, currentStatus: string)**

* Az active és inactive státusz közötti váltás történik meg.
* A db.statusSwitchStudent(id, newStatus).subscribe(...) segítségével a változás rögzítésre kerül az adatbázisban.
* Az oldal frissül, hogy a módosítások azonnal láthatók legyenek.

**Összegzés**

Ez a **diákkezelő adminisztrátori felület** a következőket biztosítja:

* **Keresési lehetőség** a diákok között.
* **Új diák hozzáadása** egy modal ablakon keresztül.
* **Diákok szerkesztése** és módosítása egy külön modal ablak segítségével.
* **Státusz módosítás** aktív/inaktív állapotok között.
* **Többnyelvű támogatás** az ngx-translate segítségével.

Ez a komponens biztosítja az adminisztrátorok számára a diákok adatainak hatékony kezelését.

***Authentication szervíz***

Ez az **AuthenticationService** egy Angular szolgáltatás, amely a felhasználók bejelentkezéséért, regisztrációjáért, kijelentkezéséért és az aktuális állapotuk ellenőrzéséért felelős. A szolgáltatás a **HttpService** segítségével kommunikál a backenddel.

**Főbb funkciók és működés**

**1. Inicializálás**

* **initfelhasznalo**: Egy alapértelmezett User objektum, amely üres értékekkel inicializálódik.
* **felhasznalo**: Ez az aktuális bejelentkezett felhasználó adatait tárolja.
* **isAuthenticated**: Egy boolean változó, amely jelzi, hogy a felhasználó be van-e jelentkezve.

A **konstruktorban** az alábbi műveletek történnek:

* Az HttpService példányosítása (http: HttpService).
* Az initfelhasznalo értékének másolása felhasznalo változóba.
* **Token és szerepkör betöltése a localStorage-ból**:
  + A felhasznalo.token a localStorage-ból olvasódik be.
  + A felhasznalo.user.role szintén a localStorage-ból kerül kiolvasásra.
* Az **isAuthenticated** értékének beállítása attól függően, hogy a localStorage tartalmazza-e az isLoggedIn kulcsot.

**2. Bejelentkezés (login függvény)**

* A login(email, passWord) függvény a backendhez küldi a bejelentkezési adatokat a this.http.login() metóduson keresztül.
* A szerver válaszát (egy User objektumot) elmenti:
  + **Token** mentése a localStorage-ba.
  + **Felhasználói név és szerepkör** mentése.
  + Az **isLoggedIn** értékét true-ra állítja.
* Sikeres bejelentkezés esetén a függvény egy **ígéretet (Promise)** ad vissza, amely tartalmazza a szerver válaszát.
* Hibás bejelentkezés esetén a reject(error.message) metódussal a hiba visszaadódik.

**3. Regisztráció (register függvény)**

* A register(userName, email, studentId, passWord, passWordConfirm) függvény a regisztrációért felel.
* Az adatok továbbítása a backend felé a this.http.registration() metódus segítségével történik.
* **Sikeres regisztráció esetén**: Egy alert üzenetben megjelenik a válaszüzenet.
* **Hibás regisztráció esetén**:
  + A backend által visszaadott validációs hibák össze vannak gyűjtve és kiírásra kerülnek egy alert mezőben.
  + Ha nincs specifikus hibaüzenet, egy általános "Ismeretlen hiba történt!" üzenet jelenik meg.

**4. Kijelentkezés (logout függvény)**

* Az **isAuthenticated** értékét false-ra állítja.
* **A felhasználó adatai törlődnek**, az initfelhasznalo értékre állítja vissza.
* **A localStorage teljes tartalma törlődik**.
* Az isLoggedIn értéke false lesz a localStorage-ban.

**5. Bejelentkezési állapot ellenőrzése (isLoggedIn függvény)**

* Az **isAuthenticated** értékét az isLoggedIn kulcs alapján határozza meg.
* Ha isLoggedIn === 'true', akkor az **isAuthenticated értéke true**.
* Ellenkező esetben az **isAuthenticated értéke false**.
* A felhasznalo.token és a felhasznalo.user.role mindig frissül a localStorage-ból.

**6. Szerepkör ellenőrzése (isAdmin és isStudent függvények)**

* **isAdmin()**: Igazat ad vissza, ha a felhasználó szerepköre "admin".
* **isStudent()**: Igazat ad vissza, ha a felhasználó szerepköre "student".
* Ezek a metódusok fontosak lehetnek az alkalmazás jogosultsági rendszerében, hogy az adminisztrátori és diák jogok megfelelően legyenek kezelve.

**Összegzés**

Ez az **AuthenticationService** az **Angular alkalmazás hitelesítési és jogosultságkezelési rétege**, amely a következőket biztosítja:

* **Felhasználók bejelentkezése és kijelentkezése.**
* **Regisztrációs folyamat kezelése, hibakezeléssel.**
* **Felhasználói jogosultságok ellenőrzése (admin vagy student).**
* **Felhasználói adatok és tokenek mentése a localStorage-ba.**
* **Promise-alapú bejelentkezés, amely lehetővé teszi az aszinkron hívások kezelését.**

Ez a szolgáltatás lehetővé teszi, hogy az Angular alkalmazás megfelelően kezelje a felhasználók autentikációját, és biztosítsa a különböző szerepkörök szerinti jogosultságokat.

***Config szervíz***

Ez a **ConfigService** egy Angular szolgáltatás, amely a **nyelvkezelésért** felelős az ngx-translate csomag segítségével.

**Főbb Funkciók és Működés**

**1. Inicializálás (constructor)**

* A **TranslateService** injektálásra kerül, amely biztosítja a nyelvi fordításokat az alkalmazásban.
* Az alapértelmezett nyelv **magyar (hu)**, amit a setDefaultLang('hu') állít be.
* A használt nyelv a localStorage-ból kerül betöltésre, ha nincs elmentett nyelv, akkor az alapértelmezett hu lesz.

**Ez biztosítja, hogy ha a felhasználó korábban más nyelvet választott, akkor az alkalmazás azt használja újraindítás után is.**

**2. Nyelvváltás (switchLanguage függvény)**

* A switchLanguage(language: string) függvény segítségével a nyelv dinamikusan megváltoztatható.
* A **use(language)** metódus beállítja a fordítási szolgáltatás aktuális nyelvét.
* A kiválasztott nyelv **elmentésre kerül a localStorage-ban**, hogy az újratöltés után is megmaradjon.

**Összegzés**

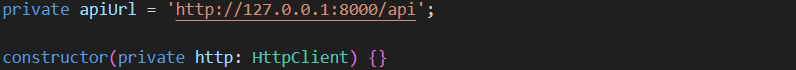
Ez a **ConfigService** felel a nyelvkezelésért és biztosítja a következőket:

* **Alapértelmezett nyelv beállítása (hu).**
* **Nyelv megőrzése localStorage-ban.**
* **Dinamikus nyelvváltás switchLanguage metódussal.**

Ezáltal az alkalmazás **felhasználóbarátabb** és **többnyelvű támogatást biztosít**!

***Http szervíz***

A **HttpService** egy Angular szolgáltatás, amely HTTP-kéréseket kezel, és az alkalmazás különböző API végpontjaival kommunikál.

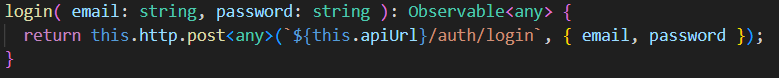


**1. Bejelentkezés és Regisztráció:**

A **login()** és **registration()** metódusok a felhasználói autentikációt és regisztrációt végzik.

* **Login**:
  + A **login()** metódus POST kérést küld a backendhez a felhasználó email és jelszó paramétereivel.
  + A sikeres válasz esetén a kapott felhasználói adatokat (token, szerepkör, név) a **localStorage**-ban tároljuk, így azok a felhasználó böngészőjében megőrződnek és később is felhasználhatóak.
  + **Backend elvárás:** JWT token visszaküldése sikeres hitelesítés esetén
  + **Adatszerkezet:** { email: string, password: string }
  + **Body tartalom:** email és jelszó

typescript



* **Regisztráció**:
  + A **registration()** metódus POST kérést küld, amely a felhasználó regisztrációs adatainak (név, email, oktatási azonosító, jelszó) feldolgozását végzi.
  + Hibák esetén a válasz tartalmazza az error üzeneteket, és azokat alertben jelenítjük meg.
  + **Backend elvárás:** Új felhasználó létrehozása és validáció
  + **Adatszerkezet:** { name: string, email: string, edu\_id: string, password: string, password\_confirmation: string }
  + **Body tartalom:** minden felhasználói adat + jelszó megerősítés

typescript

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

**2. Könyvek Kezelése:**

A könyvekkel kapcsolatos műveletek a következő metódusokban találhatóak:

* **Könyvek lekérése**:
  + A **getBook()** metódus GET kérést küld a backendnek, hogy lekérje az összes könyvtípust.
  + **Backend elvárás:** Az összes könyvtípus listázása
  + **Adatszerkezet:** Könyvtípusok tömbje
  + **Body tartalom:** nincs (GET kérés)

typescript

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

* **Könyv hozzáadása**:
  + Az **addBook()** metódus egy új könyv hozzáadását végzi a backendhez POST kéréssel. A könyv adatainak, mint raktári szám, cím, szerző, ár, példányok számának megadásával.
  + **Backend elvárás:** Új könyvtípus létrehozása
  + **Adatszerkezet:** { inventory\_number\_base: string, title: string, author: string, price: number, copies: number }
  + **Body tartalom:** könyv alapadatai

typescript

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

* **Könyv frissítése:**
  + Az **updateBook()** metódus egy meglévő könyv adatainak módosítását végzi a backendben ID alapján. A könyv új adatainak, mint raktári szám, cím, szerző, ár és példányok számának megadásával.
  + **Backend elvárás:** Meglévő könyvtípus módosítása ID alapján
  + **Adatszerkezet:** { inventory\_number\_base: string, title: string, author: string, price: number, copies: number }
  + **Body tartalom:** frissített könyvadatok

typescript

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

* **Könyv törlése:**
  + Az **deleteBook()** metódus egy könyv törlését végzi a backendben ID alapján.
  + **Backend elvárás:** Könyvtípus törlése ID alapján
  + **Adatszerkezet:** nincs
  + **Body tartalom:** nincs (DELETE kérés)

typescript

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

**3. Kölcsönzés Kezelése:**

A kölcsönzéshez kapcsolódó metódusok a következő műveleteket támogatják:

* **Kölcsönzések lekérése:**
  + A **getLoans()** metódus GET kérést küld, hogy lekérje az összes kölcsönzést.
  + **Backend elvárás:** Összes kölcsönzés listázása
  + **Adatszerkezet:** Kölcsönzések tömbje
  + **Body tartalom:** nincs (GET kérés)

typescript

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

* **Saját kölcsönzések:**
  + Az **getMyLoans()** metódus a bejelentkezett felhasználó kölcsönzéseit listázza.
  + **Backend** **elvárás:** Bejelentkezett felhasználó kölcsönzései
  + **Adatszerkezet:** Kölcsönzések tömbje
  + **Body tartalom:** nincs (GET kérés)

typescript

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

* **Új kölcsönzés:**
  + A **addLoan()** metódus POST kérést küld a backendhez, ahol a felhasználó oktatási azonosítója és a kölcsönzendő könyv raktári száma kerül elküldésre.
  + **Backend elvárás:** Új kölcsönzés létrehozása
  + **Adatszerkezet:** { user\_edu\_id: string, inventory\_number: string }
  + **Body tartalom:** oktatási azonosító és könyv raktári szám

typescript

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

* **Kölcsönzés jóváhagyás:**
  + Az **approveLoan()** metódus egy kölcsönzés státuszát "approved"-ra módosítja.
  + **Backend elvárás:** Kölcsönzés státuszának módosítása "approved"-ra
  + **Adatszerkezet:** nincs
  + **Body tartalom:** üres (POST kérés)

typescript

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

* **Kölcsönzés elutasítás:**
  + Az **rejectLoan()** metódus egy kölcsönzés státuszát "rejected"-re módosítja.
  + **Backend elvárás:** Kölcsönzés státuszának módosítása "rejected"-re
  + **Adatszerkezet:** nincs
  + **Body tartalom:** üres (POST kérés)

typescript

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

* **Könyv visszavétel:**
  + Az **returnBook()** metódus egy kölcsönzés lezárását és a könyv visszavételét végzi.
  + **Backend elvárás:** Kölcsönzés lezárása és könyv visszavétele
  + **Adatszerkezet:** nincs
  + **Body tartalom:** üres (POST kérés)

typescript

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

**4. Felhasználók Kezelése:**

A felhasználókkal kapcsolatos műveletek kezelésére az alábbi metódusok állnak rendelkezésre:

* **Felhasználók lekérése**:
  + A **getUser()** metódus GET kérést küld, hogy lekérje az összes felhasználót.
  + **Backend elvárás:** Összes regisztrált felhasználó listázása
  + **Adatszerkezet:** Felhasználók tömbje
  + **Body tartalom:** nincs (GET kérés)

typescript

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

* **Felhasználó hozzáadása**:
  + Az **addUser()** metódus POST kérést küld, hogy új felhasználót adjon hozzá a backendhez.
  + **Backend elvárás:** Új felhasználó létrehozása admin által
  + **Adatszerkezet:** { name: string, email: string, edu\_id: string }
  + **Body tartalom:** felhasználó alapadatai

typescript

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

* **Felhasználó frissítése:**
  + Az **updateUser()** metódus egy meglévő felhasználó adatainak módosítását végzi.
  + **Backend elvárás:** Felhasználó adatainak módosítása
  + **Adatszerkezet:** { name: string, email: string, role: string }
  + **Body tartalom:** frissített felhasználói adatok

typescript

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

* **Státusz váltás:**
  + Az **statusSwitchUser()** metódus egy felhasználó státuszának aktiválását vagy deaktiválását végzi.
  + **Backend** **elvárás:** Felhasználó aktiválása/deaktiválása
  + **Adatszerkezet:** { status: string }
  + **Body** **tartalom:** új státusz érték

typescript

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

**5. Diákok Kezelése:**

A diákok adatkezelés hasonlóan történik:

* **Diákok lekérése**:
  + A **getStudent()** metódus GET kérést küld a backendhez a diákok adatainak lekérésére.
  + **Backend elvárás:** Összes diák listázása
  + **Adatszerkezet:** Diákok tömbje
  + **Body tartalom:** nincs (GET kérés)

typescript

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

* **Diák hozzáadása**:
  + Az **addStudent()** metódus POST kérést küld a backendhez új diák hozzáadására.
  + **Backend elvárás:** Új diák létrehozása
  + **Adatszerkezet:** { name: string, edu\_id: string }
  + **Body tartalom:** diák alapadatai

typescript

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

* **Diák frissítése:**
  + Az **updateStudent()** metódus egy meglévő diák nevének módosítását végzi.
  + **Backend elvárás:** Diák nevének módosítása
  + **Adatszerkezet:** { name: string }
  + **Body tartalom:** új név

typescript

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

* **Diák státusz váltás:**
  + **Az statusSwitchStudent()** metódus egy diák státuszának aktiválását vagy deaktiválását végzi.
  + **Backend elvárás:** Diák aktiválása/deaktiválása
  + **Adatszerkezet:** { status: string }
  + **Body tartalom:** új státusz érték

typescript

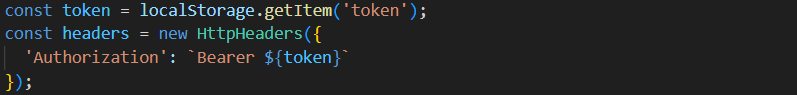
A képen szöveg, képernyőkép, Betűtípus, szoftver látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

**6. Hitelesítés és Fejlécek:**

A legtöbb metódus, amely védett API végpontokkal kommunikál, az **Authorization** fejlécet használ, amely tartalmazza a **Bearer token**-t. Ez a token a **localStorage**-ból kerül beolvasásra.

typescript



Ez biztosítja, hogy az adminisztrátor/felhasználó hitelesítve legyen, amikor érzékeny műveleteket végez, például könyveket ad hozzá, módosítja a felhasználókat, vagy kölcsönzéseket kezel. A **localStorage**-ban tárolt **token** a felhasználó aktuális munkamenetéhez tartozik, így a backend képes ellenőrizni, hogy az adott kérés jogosult-e.

**Összegzés**

Ez a **HttpService** Angular szolgáltatás egy központi pontot biztosít az alkalmazás adatkezeléséhez. Különböző API végpontokhoz küld HTTP kéréseket a felhasználói és könyves műveletek, valamint kölcsönzések kezelésére. Az **Authorization** fejléceken keresztül végzett hitelesítés és a **localStorage** használata biztosítja a felhasználó biztonságos hozzáférését a védett API végpontokhoz.

***Interfaces/User.ts***

Ez a **User** interfész egy felhasználó adatainak strukturált leírását tartalmazza, amelyet az alkalmazásban használtunk, például a bejelentkezési vagy regisztrációs válaszok kezelésére. Az interfész az alábbi mezőket tartalmazza:

**1. message: string**

* **Leírás**: Ez egy üzenet, amely általában a válasz státuszát vagy egyéb információkat tartalmazza (például sikeres bejelentkezés üzenete vagy hibaüzenet).

**2. token: string**

* A képen szöveg, képernyőkép, Betűtípus látható

  Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.**Leírás**: A felhasználó autentikációs tokenje, amelyet a frontend alkalmazásban tárolni kell (pl. a **localStorage**-ban), hogy később a felhasználói kérésekhez hozzáadhassuk az **Authorization** fejlécet.

**3. user: object**

* **Leírás**: Az objektum, amely a felhasználó alapvető adatait tartalmazza. Ez az objektum a következő mezőket tartalmazza:

**3.1 id: string**

* **Leírás**: A felhasználó egyedi azonosítója.

**3.2 name: string**

* **Leírás**: A felhasználó neve.

**3.3 email: string**

* **Leírás**: A felhasználó email címe.

**3.4 edu\_id: string**

* **Leírás**: A felhasználó oktatási azonosítója, amely egyedi azonosítóként használható az oktatási intézmény rendszerében.

**3.5 role: string**

* **Leírás**: A felhasználó szerepköre (pl. admin, user), amely meghatározza, hogy milyen jogosultságokkal rendelkezik az alkalmazásban.

**3.6 created\_at: string**

* **Leírás**: A felhasználó létrehozásának időpontja (timestamp formátumban).

**3.7 updated\_at: string**

* **Leírás**: A felhasználó utolsó frissítésének időpontja (timestamp formátumban).

Ez az interfész tipikusan a bejelentkezés után kapott válasz strukturálásához használható, hogy a backend által küldött adatokat könnyen felhasználhasd a frontendben. A **token** és a **user** objektum kulcsfontosságúak, mivel a **token**-t az autentikációs folyamatban tárolni kell, míg a **user** objektum tartalmazza a felhasználó alapvető információit, amelyeket felhasználhatsz a felhasználói profil megjelenítéséhez vagy jogosultságok kezeléséhez.

***Shared/password-mismatch.directive.ts***

Ez a typescript egy **Angular form validátor** függvényt tartalmaz, amely a felhasználói jelszó és jelszó megerősítése mezők közötti egyezőséget ellenőrzi. Ha a két mező értéke nem egyezik, a validátor hibát jelez, ha viszont egyeznek, akkor nincs hiba.

**Működés**

1. **passwordControl és passwordConfirmControl**: Az validátor két form mezőt keres a formcsoportban: az egyiket a jelszó, a másikat pedig a jelszó megerősítése számára.
2. **Hibaellenőrzés**: A validátor ellenőrzi, hogy a két mező értéke megegyezik-e. Ha nem, akkor visszaad egy hibát (passwordMismatch: true). Ha a két jelszó megegyezik, akkor null-t ad vissza, jelezve, hogy nincs hiba.

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

**Használat**

Ez a validátor egy Angular formhoz adható hozzá, és biztosítja, hogy a felhasználó által megadott jelszavak megegyezzenek. Ha a jelszavak nem egyeznek, akkor a validátor hibát ad, amit az alkalmazás kezelhet.

***Assets/hu.json, en.json***

Ez egy JSON struktúra, amely különböző szövegeket tartalmaz az alkalmazásban, magyar és angol nyelven. Az adatok különböző részekre vannak bontva: **"about"**, **"books"**, **"home"**, **"libraryadmin"**, **"libraryuser"**, **"login"**, **"navbar"**, **"registration"**, **"students"**, és **"users"**. Minden rész egy-egy szöveges tartalmat ad meg, amit a rendszer különböző felületein használhatunk.

**Elemzés**

1. **"about"**: Az iskola bemutatkozó információit tartalmazza, beleértve az iskolai címet, ügyfélszolgálatot, titkársági elérhetőségeket, és jogi adatokat (pl. alapítási év, adószám).
2. **"books"**: A könyvekhez kapcsolódó szövegek, mint a könyvek keresése, hozzáadása, szerkesztése, törlése, valamint a könyvekhez tartozó információk (raktári szám, szerző, cím, ár, példányok száma).
3. **"home"**: Az iskola bemutatkozó szövege, amely a képzési programokat, az iskola célját és szoros kapcsolatait említi a vállalatokkal.
4. **"libraryadmin"**: A könyvtár adminisztrátoroknak szóló szövegek, mint a könyvek kiadása, kölcsönzött állapotok kezelése és könyvvel kapcsolatos műveletek (elfogadás, elutasítás).
5. **"libraryuser"**: A könyvtár felhasználói számára készült szövegek, amelyek a könyvek állapotát, például kikölcsönzött, visszaadott, függőben álló státuszt tartalmazzák.
6. **"login"**: Bejelentkezéshez szükséges szövegek, például a hibák kezelése, email és jelszó mezők.
7. **"navbar"**: A navigációs sáv szövegei, mint a kezdőlap, bejelentkezés, különböző menüpontok (kikölcsönzött könyvek, könyvek, diákok, felhasználók, könyvtár, névjegy, kijelentkezés).
8. **"registration"**: A regisztrációs folyamathoz szükséges szövegek, például felhasználónév, email, oktatási azonosító, jelszó és jelszó megerősítés. Tartalmazza a hibákat is, mint például érvénytelen email cím, jelszó hiba, stb.
9. **"students"**: A diákok kezelése, beleértve a keresést, hozzáadást, szerkesztést, aktiválást és inaktiválást.
10. **"users"**: A felhasználók kezelésére vonatkozó szövegek, hasonlóan a diákokhoz. Tartalmazza a felhasználó neveit, oktatási azonosítókat, email címeket és jogosultságokat, valamint a státusz kezelését (aktív/inaktív).

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen. A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

**Használat**

Ez a JSON fájl az alkalmazás lokalizációs fájlja, amely különböző felületeken és funkciókban biztosítja a szövegek megjelenítését a felhasználóknak. A különböző szövegek segítenek az alkalmazás felhasználóbarát kezelésében, és lehetővé teszik a könnyű nyelvi beállítást.

A kulcs-érték párok bármelyik részét felhasználhatjuk az alkalmazásban, hogy dinamikusan jelenítsük meg az információkat a felhasználói felületen.

***App-routing-module.ts***

Az programban található App routing modul konfigurációja az alkalmazás navigációját biztosítja, amely különböző komponensekhez irányítja a felhasználókat. Az AppRoutingModule modulban található Routes konfiguráció az útvonalakat tartalmazza, és mindegyikhez hozzárendel egy komponenst, amit akkor renderel a rendszer, amikor a felhasználó el szeretné érni azt az útvonalat.

**Részletes elemzés:**

1. **routes tömb**: Ez az alkalmazás útvonalainak listáját tartalmazza, ahol minden egyes objektum az útvonal definícióját tartalmazza.
   * **path**: Az útvonal URL-je, amit a felhasználó böngészőjében látni fog. Például a path: 'home' URL a http://localhost:4200/home-ra irányít.
   * **component**: A komponens, amely a megadott útvonal elérésénél renderelődik. Például, amikor a felhasználó a /home útvonalra navigál, a HomeComponent fog megjelenni.
2. **{ path: '', component: HomeComponent }**: Ez az alapértelmezett (root) útvonal. Amikor a felhasználó az alap URL-t (pl. http://localhost:4200/) látogatja meg, a HomeComponent fog megjelenni.
3. **Részletes útvonalak**:
   * **Bejelentkezés (login)**: A felhasználó a /login útvonalra navigálva a LogInComponent-et látja.
   * **Regisztráció (register)**: A /register útvonal a RegistrationComponent-et jeleníti meg.
   * **Információs oldal (about)**: A /about útvonalon az AboutComponent jelenik meg.
   * **Könyvek (books)**: A /books útvonal a BooksComponent-et jeleníti meg.
   * **Diákok (students)**: A /students útvonal a StudentsComponent-et hozza elő.
   * **Felhasználók (users)**: A /users útvonal a UsersComponent-et jeleníti meg.
   * **Könyvtár listája - Felhasználói (libraryUser)**: A /libraryUser útvonalon a LibraryListUserComponent jelenik meg.
   * **Könyvtár listája - Adminisztrátori (libraryAdmin)**: A /libraryAdmin útvonalon az LibraryListAdminComponent látszik.

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

**NgModule részletezése:**

* **imports**: Az AppRoutingModule a RouterModule-t importálja, és az forRoot(routes)-t használja, hogy beállítsa az útvonalakat. Az forRoot() metódust általában az alkalmazás fő routing konfigurációjában hívjuk meg, ami egyszeri beállítást igényel.
* **exports**: Az exports kulcs alatt a RouterModule exportálása biztosítja, hogy a routing modul elérhető legyen az alkalmazás más részein is.

**A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.**

**Használat**

Ez a konfiguráció egy jól strukturált navigációs rendszert biztosít az Angular alkalmazásban, ahol a különböző URL-ekhez különböző komponensek vannak rendelve. Az alkalmazásban, amikor a felhasználó elér egy adott URL-t, az adott komponens lesz betöltve a megfelelő tartalommal.

***App.component.html***

Ez az HTML kód az Angular alkalmazás fő HTML fájlja, amely az alkalmazás navigációját és tartalmát rendezi el.

**A számunkra fontos rész a <body> elem tartalma.**

* Az <body> elem tartalmazza az oldal tartalmát, amelyet a felhasználók látnak és kölcsönhatásba lépnek vele.
* **<app-nav-bar></app-nav-bar>**: Ez az Angular komponens, amely a navigációs sávot jeleníti meg. Az app-nav-bar egy egyéni Angular komponens, amely tartalmazza az alkalmazás fő navigációs linkjeit (például "Home", "Login", "Books" stb.).
* **<router-outlet></router-outlet>**: Ez az Angular router direktívája, amely helyet biztosít a különböző komponensek számára az URL útvonalak szerint. Amikor a felhasználó különböző oldalakra navigál (például /home, /login stb.), az Angular itt fogja megjeleníteni az adott komponens tartalmát.

**Összegzés:**

Ez az alap HTML struktúra egy Angular alkalmazás számára, amely tartalmaz egy navigációs sávot és egy helyet a különböző komponensek (oldalak) renderelésére a router-outlet segítségével. Az alkalmazás dinamikusan töltődik be a navigációs menü és az útvonalak szerint.

***App.component.ts***

Ez az Angular komponens kódja, amely az AppComponent osztályt definiálja és tartalmazza az alap felépítését, mely létre jön a projekt generélésakor.

**1. @Component Dekorátor**

* A @Component dekorátor az Angular számára jelzi, hogy az alatta lévő osztály egy komponens, és tartalmazza annak konfigurációs beállításait.
* **selector: 'app-root'**: Ez a komponens az app-root nevű HTML címkében lesz használva, tehát a app-root tag az alkalmazás gyökérkomponense.
* **templateUrl: './app.component.html'**: Ez a HTML fájl, amely az AppComponent komponens sablonját tartalmazza.
* **styleUrls: ['./app.component.css']**: Ez a CSS fájl tartalmazza az alkalmazás gyökérkomponensének stílusait.

**2. AppComponent Osztály**

* Az osztály tartalmazza az alkalmazás logikáját, amelyet az Angular az adott komponenshez kapcsolódóan futtat.
* **title = 'Library';**: Ez a komponens egy title nevű változót deklarál, amely az alkalmazás címét tárolja. Ez később a sablonban felhasználható, ha dinamikusan szeretnénk megjeleníteni a cím értékét.

**3. constructor(private translate: TranslateService)**

* Az AppComponent konstruktorában egy függőségi injekcióval (Dependency Injection) injektálja a TranslateService szolgáltatást, amely lehetővé teszi a nyelvi fordítások kezelését az alkalmazásban.
* **TranslateService** az @ngx-translate/core könyvtár része, és a nyelvek közötti váltást, valamint a fordításokat kezeli.

**Miért fontos ez?**

* Az AppComponent szolgál az alkalmazás fő komponensének, így minden más komponens ezen keresztül épül fel. A nyelvkezelés miatt a TranslateService fontos szerepet játszik, mivel biztosítja, hogy az alkalmazás több nyelvet támogasson, és a felhasználói felület dinamikusan változhasson a felhasználó által választott nyelvnek megfelelően.

**Összegzés**

Ez a komponens inicializálja az alkalmazás fő nyelvkezelési mechanizmusát és meghatározza az alkalmazás címét. Az AppComponent ezen kívül a teljes alkalmazás alapvető konténere, amely az összes többi komponenst tartalmazza.

***App.module.ts***

Ez az Angular alkalmazás modulja, amely az összes szükséges beállítást és függőséget tartalmazza az alkalmazás működéséhez.

**1. @NgModule Dekorátor**

* Az @NgModule dekorátor az Angular modulját definiálja. Az Angular alkalmazás alapja a modulok rendszere, és az AppModule az alapértelmezett (root) modul, amely az egész alkalmazást körbeveszi.

**2. declarations**

* **AppComponent, HomeComponent, LogInComponent, RegistrationComponent, NavBarComponent, AboutComponent, LibraryListUserComponent, LibraryListAdminComponent, BooksComponent, StudentsComponent, UsersComponent**:
  + Ezek azok a komponensek, amelyeket az Angular ebben a modulban nyilvántart. Mivel mindegyik komponens az alkalmazás része, azokat itt kell deklarálni.

**3. imports**

* **BrowserModule**: Az Angular alapvető modulja, amely a böngésző alkalmazásokhoz szükséges függőségeket tartalmazza. Mivel ez az alapértelmezett (root) modul, itt kötelező importálni.
* **AppRoutingModule**: Az alkalmazás útvonalait (routes) kezelő modul, amely lehetővé teszi, hogy az alkalmazás különböző URL-ekre navigáljon.
* **NgbModule**: Az @ng-bootstrap modul, amely a Bootstrap elemeit integrálja az Angular alkalmazásba. Az ng-bootstrap segítségével használhatunk például modális ablakokat, popupokat, dátumválasztókat stb.
* **FormsModule, ReactiveFormsModule**: Az Angular form kezelési moduljai. A FormsModule a template-driven formokat, míg a ReactiveFormsModule a reactive formokat kezeli.
* **InputTextModule, PasswordModule**: Az InputTextModule és a PasswordModule a PrimeNG könyvtárból származik. Az InputText egyszerű input mezőt biztosít, míg a Password egy speciális mezőt biztosít jelszavak kezelésére, például titkosított karakterekkel.
* **AngularFireDatabaseModule**: Az AngularFire modul része, amely lehetővé teszi az Angular alkalmazások számára, hogy kommunikáljanak a Firebase Realtime Database-szel, ha esetleg a későbbiekben szeretnénk ezt is felhasználni a projektben.
* **TranslateModule.forRoot(...)**: Az ngx-translate könyvtár konfigurációja, amely lehetővé teszi a többnyelvű alkalmazások készítését. A TranslateHttpLoader segítségével a fordításokat JSON fájlokból töltjük be, amelyek az /assets/i18n/ mappában találhatóak.

**4. providers**

* **provideHttpClient()**: Ez a HttpClient szolgáltatás beállítása, amely lehetővé teszi HTTP kérések küldését az alkalmazásból. Az Angular HttpClientModule-ját használja az API-k hívásához, például a fordítások betöltéséhez.

**5. bootstrap**

* **AppComponent**: Az alkalmazás gyökérkomponense, amelyet először indít el az Angular. Ez fogja tartalmazni a többi komponenst és az alkalmazás felépítését.

**6. HttpLoaderFactory**

* A HttpLoaderFactory egy függvény, amely a TranslateHttpLoader példányát hozza létre. A TranslateHttpLoader felelős a fordítások betöltéséért a megfelelő URL-ről (itt: /assets/i18n/ mappa).
* A HttpClient szolgáltatást injektálja, hogy az HTTP kéréseket végrehajtsa a fordítások betöltésére.

A képen szöveg, képernyőkép, szoftver látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

**Összegzés**

Ez az Angular alkalmazás AppModule-ja, amely az alkalmazás összes szükséges függőségét és beállítását kezeli. Az importált modulok és komponensek biztosítják az alkalmazás funkcióit, beleértve a routingot, a többnyelvűséget, a formokat, a Firebase adatbázist, és a felhasználói felület elemeit. Az ngx-translate és a ng-bootstrap könyvtárak lehetővé teszik a dinamikus nyelvkezelést és az egyedi Bootstrap elemek használatát.

Tesztelés

A tesztelésen komponensenként fogok végig menni és bemutatok minden lehetséges hibát, illetve a helyes megoldást/használatot is.

Komponensek amikben nincs hibalehetőség: HomeComponent, AboutComponent, és a NavBarComponent.

A gombok esetében nincs helytelen használat, mivel a hibákat kizártuk azzal, hogy a helytelen használatkor disabled a gomb: pl:. visszaadott állapotú könyv esetén értelmetlenné válik az elfogadás és elutasítás gomb használata, illetve a kikölcsönzött állapotú könyv elutasítása, mivel az még nem lett benyújtásra kérelmezve. Pl:.  
A képen szöveg, képernyőkép, szoftver, Számítógépes ikon látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

LogInComponent:

* Hibás használatok:
  + Email helytelen formátuma: amely nem tartalmazza a @ karaktert vagy a megfelelő végződést (pl:. kulker@gimihu, kulkergimi.hu).
  + Jelszó: regisztrációkor megadott jelszó ami legfeljebb 8 karakter és nem tartalmaz kis- és vagy nagy betűket, vagy számot.
* Helyes használat:
  + Email helyes formátuma: tartalmazza egy megfelelő email cím közismert adottságait (pl:.kulker@gimi.hu).
  + Jelszó: regisztrációkor megadott jelszó ami legalább 8 karakter és tartalmaz kis- és nagy betűket, illetve számot. (pl:. Kulkergimi1).

A képen szöveg, képernyőkép, Betűtípus, Operációs rendszer látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

RegistrationComponent:

* Hibás használatok:
  + Felhasználónév: ha üresen van hagyva.
  + Email helytelen formátuma: amely nem tartalmazza a @ karaktert vagy a megfelelő végződést (pl:. kulker@gimihu, kulkergimi.hu).
  + Oktatási azonosító: ha nem 7-essel kezdődik és nem pontosan 11 szám
  + Jelszó: regisztrációkor megadott jelszó ami legfeljebb 8 karakter és nem tartalmaz kis- és vagy nagy betűket, vagy számot.
  + Jelszó megerősítése: ha nem egyezik meg a jelszóval.
* Helyes használat:
  + Felhasználónév: ha nincs üresen van hagyva.
  + Email helyes formátuma: tartalmazza egy megfelelő email cím közismert adottságait (pl:.kulker@gimi.hu)
  + Oktatási azonosító: ha 7-essel kezdődik és pontosan 11 szám
  + Jelszó: regisztrációkor megadott jelszó ami legalább 8 karakter és tartalmaz kis- és nagy betűket, illetve számot. (pl:. Kulkergimi1)
  + Jelszó megerősítése: ha megegyezik a jelszóval.

A képen szöveg, képernyőkép, szoftver, Operációs rendszer látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

LibraryListUserComponent:

* Hibás használatok: nincs
* Helyes használat: A felhasználó megnyomja a kikölcsönzött könyv melletti gombot, ha vissza szeretné adni a kikölcsönzött könyvét, ami ezután függő állapotba kerül

A képen szöveg, képernyőkép, szoftver, Számítógépes ikon látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

LibraryListAdminComponent:

* Hibás használatok:
  + Az adatbázisban nem létező oktatási azonosító vagy raktári szám alapján való kiadási próbálkozás.

A képen szöveg, képernyőkép, szoftver, Multimédiás szoftver látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.A képen szöveg, képernyőkép, szoftver, Multimédiás szoftver látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

* Helyes használat:
  + Az adatbázisban létező oktatási azonosító vagy raktári szám alapján való kiadási próbálkozás.

A képen szöveg, szoftver, Multimédiás szoftver, Számítógépes ikon látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

BooksComponent:

* Hibás használatok:
  + Új könyv hozzáadása esetén egy teljesen üres vagy csak 1 adat hiányában hibát dob a program.

A képen szöveg, képernyőkép, szoftver, Multimédiás szoftver látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

A képen szöveg, képernyőkép, szoftver, Multimédiás szoftver látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

* + Szerkesztés esetén ha az ár és példányszám betűt tartalmaz, az alábbi hibát dobja a program.

A képen szöveg, képernyőkép, szoftver, Multimédiás szoftver látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

* Helyes használat: Szerkesztés és hozzáadás esetén is a megfelelő adatokat adni a rendszernek.

StudentsComponent:

* Hibás használatok:
  + Új diák hozzáadása esetén egy teljesen üres vagy csak 1 adat hiányában hibát dob a program.

A képen szöveg, képernyőkép, szoftver, Multimédiás szoftver látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

* + Szerkesztés esetén nem fog hibát dobni a program.

A képen szöveg, képernyőkép, szoftver, Multimédiás szoftver látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

* Helyes használat: Szerkesztés és hozzáadás esetén is a megfelelő adatokat adni a rendszernek.

UsersComponent:

* Hibás használatok:
  + Új felhasználó hozzáadása esetén egy teljesen üres vagy csak 1 adat hiányában hibát dob a program.

A képen szöveg, szoftver, Multimédiás szoftver, Számítógépes ikon látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

* Helyes használat: Szerkesztés és hozzáadás esetén is a megfelelő adatokat adni a rendszernek.

Felhasznált parancsok a program létrehozásánál:

* ng n Library --standalone=false
* css-enter
* Server-Side Rendering (SSR)-no
* cd Library
* ng add @ng-bootstrap/ng-bootstrap
* npm install firebase @angular/fire@18
* npm i @ngx-translate/core
* npm install @angular/cdk@18
* npm install primeng@18 @primeng/themes@18
* ng g c components/home
* ng g c components/logIn
* ng g c components/registration
* ng g c components/about
* ng g c components/navBar
* ng g c components/libraryListUser
* ng g c components/libraryListAdmin
* ng g c components/users
* ng g c components/books
* ng g c components/students ----components
* ng g s services/authentication
* ng g s services/config
* ng g s services/http ------service
* ng serve -o

**Backend fejlesztői dokumentáció**

**Api.php**

**Nyelv beállítása**

* *GET:* /lang/{locale}
* Leírás: A támogatott nyelvek: hu (magyar) és en (angol).

**Végpontok**

**Nem regisztrált felhasználó által elérhető funkciók**

* **AuthController**
  + *POST:* /auth/register
  + *POST:* /auth/login

**Regisztrált felhasználó által elérhető funkciók**

* **BookController**
  + *GET:* /books
  + *GET:* /books/{id}
* **LoanController**
  + *GET:* /myLoans
  + *POST:* /books/return/{loan\_id}

**Adminisztrátor által elérhető funkciók**

1. **UserController**
   * *GET:* /users
   * *POST:* /users
   * *GET:* /users/{id}
   * *PUT:* /users/{id}
   * *PUT:* /users/{id}/status
2. **BookController**
   * *POST:* /books
   * *PUT:* /books/{book}
   * *DELETE:* /books/{book}
3. **BookTypeController**
   * *GET:* /book-types
   * *GET:* /book-types/{id}
   * *POST*: /book-types
   * *PUT:* /book-types/{id}
   * *DELETE:* /book-types/{id}
4. **LoanContoller**
   * *POST:* /books/borrow
5. **StudentContoller**
   * *GET:* /students
   * *POST:* /students
   * *GET:* /student/{id}
   * *PUT:* /students/{id}
   * *PUT:* /student/{id}/status
6. **Adminisztrátor által kezelt kölcsönzések**
   * *GET:* /loans
   * *POST:* /loans/approve/{loan}
   * *POST:* /loans/reject/{loan}
   * *POST:* /loans/{loan}/force

**Controllerek**

**AdminController**

* **listLoans /** *GET:* /loans
  + **Leírás:**
    - Lekérdezi az összes kölcsönzést, beleértve a felhasználókat és a könyveket a hozzájuk tartozó típusokkal.
  + **Funkciók:**
    - Az BorrowedBook modellel lekérdezi a kölcsönzéseket, és with() segítségével betölti a kapcsolódó user és book.bookType adatokat.
    - A válasz JSON formátumban tartalmazza a kölcsönzés adatokat.
  + **Válaszok:**
    - **Sikeres**
      * A kölcsönzés listája JSON formátumban, amely tartalmazza a felhasználót, könyvet és annak típusát.
    - **Hibás:**
      * Nincs hiba, ha a kölcsönzések lekérdezése sikeres.

*Sikeres lekérdezés:*

*json*

*[{*

*"id": 1,*

*"user\_edu\_id": 71235567811,*

*"inventory\_number": "SI-NPI1AMF-001",*

*"status": "returned",*

*"created\_at": "2025-03-10T15:08:58.000000Z",*

*"updated\_at": "2025-03-17T13:25:12.000000Z",*

*"user": null,*

*"book": {*

*"id": 1,*

*"book\_type\_id": 1,*

*"inventory\_number": "SI-NPI1AMF-001",*

*"created\_at": null,*

*"updated\_at": null,*

*"book\_type": {*

*"id": 1,*

*"inventory\_number\_base": "SI-NPI1AMF",*

*"title": "Nuovissimo Progetto italiano 1A – Quaderno degli esercizi con DVD",*

*"author": "T. Marin",*

*"price": 2800,*

*"copies": 99,*

*"created\_at": null,*

*"updated\_at": "2025-03-19T13:09:08.000000Z"*

*}*

*}*

*}]*

* **approveReturn /** *POST:* /loans/approve/{loan}
  + **Leírás:**
    - Jóváhagyja a kölcsönzött könyv visszaadását.
  + **Funkciók:**
    - Ellenőrzi, hogy a kölcsönzés állapota *requested\_return*, ha nem, hibát ad vissza.
    - Frissíti a könyv típusának példányait, növelve azok számát, majd a kölcsönzés állapotát *returned*-re állítja.
  + **Paraméter:**
    - loan: A kölcsönzés id-ját tartalmazza, amelyet a route paramétereként kap.
  + **Válaszok:**
    - **Sikeres**
      * Ha a művelet sikeres, egy JSON válasz, amely megerősíti a visszaadás jóváhagyását.
    - **Hibás:**
      * Ha a kölcsönzés állapota nem *requested\_return*, hibát ad vissza a következő üzenettel: *loan\_not\_pending\_return*.

*Sikeres jóváhagyás (angolul):*

*json*

*{*

*"message": "Return approved."*

*}*

*Sikertelen jóváhagyás (angolul):*

*json*

*{*

*"message": "This loan is not pending return approval"*

*}*

* **rejectReturn /** *POST:* /loans/reject/{loan}
  + **Leírás:**
    - Elutasítja a könyv visszaadását.
  + **Funkciók:**
    - Ellenőrzi, hogy a kölcsönzés állapota *requested\_return* , ha nem, hibát ad vissza.
    - A kölcsönzés állapotát *borrowed*-ra frissíti, jelezve, hogy a könyvet nem adták vissza.
  + **Paraméter:**
    - loan: A kölcsönzés id-ját tartalmazza, amelyet a route paramétereként kap.
  + **Válaszok:**
    - **Sikeres**
      * Ha a művelet sikeres, egy JSON válasz, amely megerősíti a visszaadás elutasítását.
    - **Hibás:**
      * Ha a kölcsönzés állapota nem requested\_return, hibát ad vissza a következő üzenettel: *loan\_not\_pending\_return.*

*Sikeres elutasítás (angolul):*

*json*

*{*

*"message": "Return request rejected."*

*}*

*Sikertelen elutasítás (angolul):*

*json*

*{*

*"message": "This loan is not pending return approval"*

*}*

* **forceApproveReturn /** *POST:* /loans/{loan}/force
  + **Leírás:**
    - Könyv visszaadása, akkor is, ha a kölcsönzés még nem *requested\_return* állapotú.
  + **Funkciók:**
    - Ha a kölcsönzés állapota már *returned*, hibát ad vissza.
    - Frissíti a könyv típusának példányait, növelve azok számát, majd a kölcsönzés állapotát *returned*-re állítja.
  + **Paraméter:**
    - loan: A kölcsönzés id-ját tartalmazza, amelyet a route paramétereként kap.
  + **Válaszok:**
    - **Sikeres**
      * Ha a művelet sikeres, egy JSON válasz, amely megerősíti a kényszerített visszaadás jóváhagyását.
    - **Hiba:**
      * Ha a kölcsönzés már vissza lett adva, hibát ad vissza a következő üzenettel: *loan\_already\_returned.*

*Sikeres jóváhagyás (angolul):*

*json*

*{*

*"message": "Force return approved."*

*}*

*Sikertelen jóváhagyás (angolul):*

*json*

*{*

*"message": "* *This loan has already been returned or does not exist"*

*}*

**AuthController**

* **register /** *POST*: /auth/register
  + **Leírás**:
    - A felhasználó regisztrációját végzi. A regisztráció során validálja a bemeneteket és új felhasználót hoz létre a rendszerben.
  + **Funkciók**:
    - A metódus először validálja a kérésben érkező adatokat:
      * **name**: Kötelező, maximum 255 karakter hosszú, csak szöveget tartalmazhat.
      * **email**: Kötelező, egyedi email cím, maximum 255 karakter hosszú.
      * **edu\_id**: Kötelező, egyedi, csak számokat tartalmazó karakterlánc, amely 11 számjegyű és 7-tel kezdődik (például 71234567890).
      * **password**: Kötelező, legalább 8 karakter hosszú, kis- és nagybetűt, valamint számot is tartalmazó jelszó.
    - Ha a validálás nem sikerül, a hibák JSON válaszként kerülnek visszaküldésre.
    - A sikeres validálás után létrejön a felhasználó az *User::create()* metódus segítségével.
    - A jelszót biztonságosan hasheljük az *Hash::make()* segítségével.
* **Válaszok**:
  + **Sikeres**:
    - Ha minden rendben van, a válasz egy üzenet, hogy a felhasználó sikeresen regisztrálva lett.
  + **Hibás:**
    - **Validációs hiba**:
      * Ha valamelyik mező nem felel meg az elvárásoknak, akkor a válasz tartalmazza a hibákat.

*Sikeres regisztráció (angolul):*

*json*

*{*

*"message": "User successfully registered."*

*}*

*Sikertelen regisztráció (angolul):*

json

{

"errors": {

"email": ["The email has already been taken."]

}

}

* **login** / *POST*: /auth/login
  + **Leírás**:
    - A felhasználó bejelentkezését kezeli, és ha sikeres, egy **API tokent** generál számára.
  + **Funkciók**:
    - A metódus validálja a bejelentkezési adatokat:
    - **email**: Kötelező, email formátumban.
    - **password**: Kötelező, legalább 8 karakter hosszú.
    - Ha a validálás nem sikerül, a hibák JSON válaszként kerülnek visszaküldésre.
    - A felhasználó email címét és jelszavát ellenőrzi. Ha a felhasználó nem létezik vagy a jelszó helytelen, hibát ad vissza.
    - Ha a felhasználó státusza inactive, az azt jelenti, hogy az account inaktív, és az API erről tájékoztatja a felhasználót.
    - **API tokent**, a felhasználó a további API hívásokhoz használhat.
  + **Válaszok**:

*Sikeres bejelentkezés (angolul):*

*json*

*{*

*"message": "Login successful.",*

*"token": "your\_generated\_token\_here",*

*"user": {*

*"id": 1,*

*"name": "Teszt User",*

*"email": "tesztuser@example.com",*

*"edu\_id": "71234567890",*

*"status": "active"*

*}*

*Érvénytelen hitelesítő adatok (angolul:)*

*json*

*{*

*"message": "Invalid credentials."*

*}*

*Inaktív fiók(angolul):*

*json*

*{*

*"message": "Your account is inactive."*

*}*

*Validációs hiba(angolul):*

*json*

*{*

*"errors": {*

*"email": ["The email field is required."]*

*}*

*}*

**BookController**

* **index /** *GET:* /books
  + **Leírás:**
    - Az összes könyv lekérdezése, a könyv típusokkal együtt.
  + **Funkciók:**
    - Az index metódus a *Book::with('bookType')->get()* használatával lekéri az összes könyvet, és betölti a hozzájuk tartozó könyv típusokat (bookType).
    - A válasz egy JSON tömb, amely tartalmazza az összes könyvet és azok típusait.
  + **Válasz:**
    - JSON formátumban, amely tartalmazza az összes könyvet és a hozzájuk tartozó típusokat.

*Sikeres lekérdezés:*

*json*

*[{*

*"id": 1,*

*"book\_type\_id": 1,*

*"inventory\_number": "12345",*

*"bookType": {*

*"id": 1,*

*"name": "Fiction",*

*"copies": 5*

*}*

*}]*

* **show /** *GET:* /books/{id}
  + **Leírás:**
    - Egy adott könyv megjelenítése a könyv típusával együtt.
  + **Funkciók**:
    - A metódus *a Book::with('bookType')->find($id)* segítségével megkeresi a könyvet a megadott id alapján.
    - Ha a könyv nem található, 404-es hibát ad vissza a *book\_not\_found* üzenettel.
    - Ha a könyv létezik, akkor a válasz tartalmazza a könyv adatait és annak típusát.
  + **Paraméter:**
    - id: A könyv id-ját tartalmazza, amelyet a route paramétereként kap.
  + **Válaszok**:

*Sikeres lekérdezés:*

*json*

*{*

*"id": 1,*

*"book\_type\_id": 1,*

*"inventory\_number": "12345",*

*"bookType": {*

*"id": 1,*

*"name": "Fiction",*

*"copies": 5*

*}*

*}*

*Sikertelen lekérdezés (angolul):*

*json*

*{*

*"message": "Book not found"*

*}*

* **store /** *POST:* /books
  + **Leírás:**
    - Új könyv hozzáadása a rendszerhez.
  + **Funkciók**:
    - A bemeneti adatokat validálja:
    - **book\_type\_id**: Kötelező és léteznie kell a book\_types táblában.
    - **inventory\_number**: Kötelező és egyedinek kell lennie a könyvek között.
    - Ha a validálás sikeres, létrehozza az új könyvet a *Book::create()* segítségével.
    - A könyv létrehozása után frissíti a könyv típusának példányszámát (*updateTotalQuantity*).
  + **Válasz**:

*Sikeres hozzáadás (angolul)*

*json*

*{*

*"message": "Book added successfully",*

*"book": {*

*"id": 1,*

*"book\_type\_id": 1,*

*"inventory\_number": "12345"*

*}*

* **update** / *PUT:* /books/{book}
  + **Leírás:**
    - A könyv frissítése a megadott *id* alapján.
  + **Funkciók**:
    - A metódus ellenőrzi, hogy létezik-e a könyv a megadott *id* alapján.
    - A bemeneti adatokat validálja:
    - **inventory\_number**: Ha módosítva van, akkor biztosítani kell, hogy az új inventory\_number egyedülálló legyen.
  + **Paraméter:**
    - book: A könyv id-ját tartalmazza, amelyet a route paramétereként kap.
  + **Válaszok**:

*Sikeres* frissítés *(angolul)*

*json*

*{*

*"message": "Book updated successfully",*

*"book": {*

*"id": 1,*

*"book\_type\_id": 1,*

*"inventory\_number": "12345"*

*}*

*Sikertelen* frissítés *(angolul)*

*json*

*{*

*"message": "Book not found"*

*}*

* **destroy** / *DELETE:* /books/{book}
  + **Leírás**:
    - A könyv törlése a megadott id alapján.
  + **Funkciók**:
    - A metódus ellenőrzi, hogy létezik-e a könyv a megadott *id* alapján.
    - Ha a könyv létezik, törli azt, és frissíti a könyv típusának példányszámát (*updateTotalQuantity*).
  + **Válaszok**:

*Sikeres törlés(angolul)*

*json*

*{*

*"message": "Book deleted successfully"*

*}*

*Sikertelen törlés(angolul)*

*json*

*{*

*"message": "Book not found"*

*}*

* **updateTotalQuantity (Privát metódus)**
  + **Leírás:**
    - Ez a metódus frissíti minden könyv típusának copies mezőjét. A *book\_types* táblában a *copies* mező értékét a kapcsolódó könyvek számával frissíti.
  + **Funkció:**
    - A metódus egy SQL lekérdezést hajt végre, amely minden könyv típusához frissíti a példányszámot.

**BookTypeController**

* **index /** *GET*: /book-types
  + **Leírás:**
    - Az összes könyv típus lekérdezése.
  + **Funkció**:
    - Az index metódus egyszerűen visszaadja az összes könyv típust JSON formátumban, a *BookType::all()* lekérdezéssel.
  + **Válasz**:

*Sikeres lekérdezés*

*json*

*[{*

*"id": 1,*

*"inventory\_number\_base": "ABC",*

*"title": "The Great Book",*

*"author": "Teszt User",*

*"price": 200,*

*"copies": 5*

*}]*

* **store** / *POST*: /book-types
  + **Leírás:**
    - Új könyv típus hozzáadása.
  + **Funkciók**:
    - A metódus validálja a bemeneti adatokat:
    - **inventory\_number\_base**: Kötelező és karakterlánc típusú.
    - **title**: Kötelező, és egyedinek kell lennie a book\_types táblában.
    - **author**: Kötelező, karakterlánc típusú.
    - **price**: Kötelező, numerikus érték.
    - **copies**: Kötelező, egész szám, minimum érték 0.
    - A könyv típus létrehozása után frissíti a CSV fájlt a *updateCsvFile()* privát metódussal.
    - A *BookSeeder* segítségével legenerálja az új könyveket és az SQL táblába helyezi.
  + **Válasz**:

*Sikeres hozzáadás (angolul)*

*json*

*{*

*"message": "Book type added successfully",*

*"book\_type": {*

*"id": 1,*

*"inventory\_number\_base": "ABC",*

*"title": "The Great Book",*

*"author": "Teszt User",*

*"price": 19.99,*

*"copies": 5*

*}*

*}*

* **show /** *GET*: /book-types/{id}
  + **Leírás:**
    - Egy adott könyv típusának megjelenítése az *id* alapján.
  + **Funkciók**:
    - A metódus a *BookType::find($id)* segítségével keres egy könyv típust a megadott *id* alapján.
    - Ha nem található a könyv típus, 404-es hibát ad vissza a *book\_type\_not\_found* üzenettel.
  + **Paraméter:**
    - id: A könyv típus id-ját tartalmazza, amelyet a route paramétereként kap.
  + **Válaszok**:

*Sikeres lekérdezés*

*json*

*{*

*"id": 1,*

*"inventory\_number\_base": "ABC",*

*"title": "The Great Book",*

*"author": "Teszt User",*

*"price": 20,*

*"copies": 5*

*}*

*Sikertelen lekérdezés(angolul)*

*json*

*{*

*"message": "Book type not found"*

*}*

* **update** / *PUT*: /book-types/{id}
  + **Leírás:**
    - Egy meglévő könyv típus frissítése.
  + **Funkciók**:
    - A metódus először ellenőrzi, hogy létezik-e a könyv típus az *id* alapján.
    - Ha létezik, a bemeneti adatokat validálja:
    - A validálás során a title esetén figyelembe kell venni, hogy a könyv típus már létezik-e másik könyv típusban.
    - A frissített könyv típus adatai után frissíti a CSV fájlt.
  + **Paraméter:**
    - id: A könyv típus id-ját tartalmazza, amelyet a route paramétereként kap.
  + **Válaszok**:

*Sikeres frissítés(angolul)*

*json*

*{*

*"message": "Book type updated successfully",*

*"book\_type": {*

*"id": 1,*

*"inventory\_number\_base": "ABC",*

*"title": "The Great Book - Updated",*

*"author": "Teszt User",*

*"price": 20.99,*

*"copies": 10*

*}*

*}*

*Sikertelen frissítés(angolul)*

*json*

*{*

*"message": "Book type not found"*

*}*

* **destroy** / *DELETE*: /book-types/{id}
  + **Leírás:**
    - A könyv típus törlése.
  + **Funkciók**:
    - A metódus először ellenőrzi, hogy létezik-e a könyv típus az *id* alapján.
    - Ha létezik, törli azt, és frissíti a CSV fájlt.
  + **Paraméter:**
    - id: A könyv típus id-ját tartalmazza, amelyet a route paramétereként kap.
  + **Válasz**:
    - A törlés után visszakapjuk az értesítést, hogy a könyv típus sikeresen törlődött.

Sikeres törlés (angolul)

json

{

"message": "Book type deleted successfully"

}

Sikertelen törlés (angolul)

json

{

"message": "Book type not found"

}

* **updateCsvFile (Privát metódus)**
  + **Leírás:**
    - Ez a metódus frissíti a könyv típusok CSV fájlt, amely az összes könyv típusát tartalmazza.
  + **Funkciók:**
    - A metódus a könyv típusokat lekérdezi az adatbázisból és CSV formátumban eltárolja.
    - A CSV fájl a *public/booktypes.csv* helyen található, és az összes könyv típus mezőit tartalmazza, beleértve az *id, inventory\_number\_base, title, author, price,* és *copies* mezőket.
    - A fájl frissítése után a *db:seed* parancsot hívja meg, hogy a frissített adatokat betöltse az adatbázisba.

**LoanController**

* **borrow /** *POST*: /books/borrow
  + **Leírás**:
    - Az adminisztrátor könyvet ad ki a diák számára.
  + **Funkciók**:
    - A metódus validálja a bemeneti adatokat:
    - **user\_edu\_id**: Kötelező és a users táblában létező egyedi azonosító.
    - **inventory\_number**: Kötelező és a books táblában létező egyedi könyv azonosító.
    - Ellenőrzi, hogy a felhasználónak van-e már aktív kölcsönzése a kért könyvből (olyan kölcsönzés, amely státusza "borrowed" vagy "requested\_return").
    - Ha van aktív kölcsönzés, visszautasítja a kérést.
    - Ellenőrzi, hogy van-e elérhető példány a könyvből, és ha van, létrehozza a kölcsönzést a BorrowedBook modellben, majd csökkenti a könyv példányainak számát.
  + **Válaszok**:
    - **Sikeres:**
      * A könyv kölcsönzésének sikeres végrehajtása után visszakapjuk a sikeres üzenetet.
    - **Hibás:**
      * Ha a könyv már kölcsönözve van.

*Sikeres kiadás(angolul)*

*json*

*{*

*"message": "Book borrowed successfully"*

*}*

*Sikertelen kiadás(angolul)*

*json*

*{*

*"message": "Book already borrowed"*

*}*

* **requestReturn /** *POST*: /books/return/{loan\_id}
  + **Leírás:**
    - A felhasználó visszavételi kérelmet nyújt be egy általa kölcsönzött könyvre.
  + **Funkciók**:
    - Ellenőrzi, hogy létezik-e a kölcsönzés a megadott loan\_id alapján.
    - Ellenőrzi, hogy a felhasználó jogosult-e a visszavételi kérés benyújtására
    - Ha a kölcsönzés státusza nem "*borrowed*", akkor nem engedélyezi a kérés benyújtását.
    - Ha minden feltétel teljesül, a kölcsönzés státuszát "*requested\_return*"-ra frissíti.
  + **Paraméter:**
    - loan\_id: A kölcsönzés id-ját tartalmazza, amelyet a route paramétereként kap.
  + **Válaszok**:
    - **Sikeres:**
      * A visszavételi kérés sikeres benyújtása után az alábbi üzenet jelenik meg.
    - **Hibás:**
      * Ha a kölcsönzés nem található, vagy nem a megfelelő felhasználó próbálja benyújtani a kérést.

*Sikeres kérés (Angolul)*

*json*

*{*

*"message": "Return request submitted successfully"*

*}*

*Sikertelen kérés (Angolul)*

*json*

*{*

*"message": "Loan not found"*

*}*

* **myLoans** / *GET*: /myLoans
  + **Leírás:**
    - A felhasználó összes kölcsönzését lekérdezi.
  + **Funkció**:
    - A metódus lekérdezi a felhasználó összes kölcsönzését a *BorrowedBook* modellel, és visszaadja azokat JSON formátumban.
  + **Válasz**:
    - A felhasználó összes kölcsönzését tartalmazó JSON válasz.

*Sikeres lekérdezés*

*json*

*{*

*"loans":*

*[{*

*"id": 1,*

*"user\_edu\_id": "7234567890",*

*"inventory\_number": "BOOK-123",*

*"status": "borrowed",*

*"book": {*

*"id": 1,*

*"title": "The Great Book",*

*"author": "Teszt User"*

*}*

*}]*

*}*

**StudentController**

* **index** / *GET*: /students
  + **Leírás:**
    - Visszaadja az összes diák adatát.
  + **Funkciók**:
    - Lekéri az összes diákot a Student modelből.
    - Az adatokat JSON formátumban adja vissza.
  + **Válasz**:

*Sikeres lekérdezés*

*json*

*[*

*{*

*"id": 1,*

*"name": "Teszt User",*

*"edu\_id": "712345678901",*

*"status": "active"*

*},*

*...*

*]*

* **store** / *POST*: /students
  + **Leírás:**
    - Új diákot hoz létre a rendszerben.
  + **Funkciók**:
    - A metódus validálja a bemeneti adatokat:
    - **name**: Kötelező és string típusú, maximum 255 karakter hosszú.
    - **edu\_id**: Kötelező és egyedi, amely megfelel a regex mintának, ami a formátumot ellenőrzi (például: "7" kezdettel 11 karakter hosszú).
    - Ha a validáció sikeres, akkor létrehoz egy új diákot a Student modell segítségével, és az adatokat elmenti az adatbázisba.
  + **Válaszok**:
    - **Sikeres:**
      * A sikeres diák létrehozása után az alábbi üzenet jelenik meg:
    - **Hibás:**
      * Ha a validáció nem sikerült, a hibák egyike jelennek meg:

*Sikeres hozzáadás(angolul)*

*json*

*{*

*"message": "Student created successfully",*

*"student": {*

*"id": 1,*

*"name": "Teszt User",*

*"edu\_id": "712345678901",*

*"status": "active"*

*}*

*}*

*Sikertelen hozzáadás(angolul)*

*json*

*{*

*"errors": {*

*"edu\_id": ["The edu id format is invalid."]*

*}*

*}*

* **show** / *GET*: /student/{id}
  + **Leírás:**
    - Egy diák adatának lekérdezése az azonosító alapján.
  + **Funkciók**:
    - A metódus megkeresi a diákot az adatbázisban a megadott *id* alapján.
    - Ha a diák nem található, 404-es hibát ad vissza, ha pedig megtalálható, a diák adatait visszaadja.
  + **Paraméter:**
    - id: A diák id-ját tartalmazza, amelyet a route paramétereként kap
  + **Válaszok**:

*Sikeres lekérdezés*

*json*

*{*

*"id": 1,*

*"name": "Teszt User",*

*"edu\_id": "712345678901",*

*"status": "active"*

*}*

*Sikertelen lekérdezés(angolul)*

*json*

*{*

*"message": "Student not found"*

*}*

* **update** / *PUT*: /students/{id}
  + **Leírás:** Egy diák adatának módosítása.
  + **Funkciók**:
    - A metódus megkeresi a diákot az adatbázisban a megadott *id* alapján.
    - Ha a diák létezik, frissíti az adatokat a kérésben szereplő értékekkel.
    - Ha a **edu\_id** változik, akkor annak is egyedinek kell lennie, és megfelelő formátumban kell lennie.
  + **Paraméter:**
    - id: A diák id-ját tartalmazza, amelyet a route paramétereként kap
  + **Válaszok**:

*Sikeres változtatás*

*json*

*{*

*"id": 1,*

*"name": "Teszt User",*

*"edu\_id": "712345678901",*

*"status": "active"*

*}*

*Sikertelen változtatás (angolul)*

*json*

*{*

*"message": "Student not found"*

*}*

* **updateStatus** /PUT: /student/{id}/status
  + **Leírás:**
    - A diák státuszának frissítése, például aktív vagy inaktív státusz.
  + **Funkciók**:
    - A metódus megkeresi a diákot az adatbázisban a megadott id alapján.
    - Frissíti a státuszt a kérésben szereplő értékre (csak "active" vagy "inactive" lehet).
    - A státusz frissítése után visszaadja a frissített adatokat.
  + **Paraméter:**
    - id: A diák id-ját tartalmazza, amelyet a route paramétereként kap
  + **Válaszok**:
    - **Sikeres**
      * A státusz frissítése után az új státusszal rendelkező diák adatai.
    - **Hibás:**
      * Ha a státusz inaktív, vagy a diák nem található

*Sikeres frissítés (angolul)*

*json*

*{*

*"message": "Student status updated",*

*"student": {*

*"id": 1,*

*"name": "Teszt User",*

*"edu\_id": "712345678901",*

*"status": "inactive"*

*}*

*}*

*Sikertelen frissítés (angolul)*

*json*

*{*

*"message": "Student not found"*

*}*

**UserController**

* **index** / *GET*: /users
  + **Leírás:**
    - Visszaadja az összes felhasználó adatát.
  + **Funkciók**:
    - A metódus lekéri az összes felhasználót a User modelből.
    - Az adatokat JSON formátumban adja vissza.
  + **Válasz**:

*Sikeres lekérdezés*

*json*

*[*

*{*

*"id": 1,*

*"name": "Teszt User",*

*"email": "tesztuser@example.com",*

*"edu\_id": "712345678901",*

*"status": "active",*

*"role": "admin"*

*},*

*...*

*]*

* **store** / *POST*: /users
  + **Leírás:**
    - Új felhasználót hoz létre a rendszerben.
  + **Funkciók**:
    - A metódus validálja a bemeneti adatokat:
      * **name**: Kötelező és string típusú, maximum 255 karakter hosszú.
      * **email**: Kötelező, email formátumú, egyedi, maximum 255 karakter hosszú.
      * **edu\_id**: Kötelező és egyedi, megfelel a regex mintának, ami a formátumot ellenőrzi (például: "7" kezdettel 11 karakter hosszú).
      * **password**: Kötelező, minimum 8 karakter, tartalmazzon kis- és nagybetűket, számokat.
    - Ha a validáció sikeres, akkor létrehoz egy új felhasználót a User modell segítségével, és az adatokat elmenti az adatbázisba.
  + **Válaszok**:
    - **Sikeres:**
      * A sikeres felhasználó létrehozása után az alábbi üzenet jelenik meg:
    - **Hibás:**
      * Ha a validáció nem sikerült, az alábbi hibák jelennek meg:

*Sikeres hozzáadás (angolul)*

*json*

*{*

*"message": "User created successfully",*

*"user": {*

*"id": 1,*

*"name": "Teszt User",*

*"email": "tesztuser@example.com",*

*"edu\_id": "712345678901",*

*"status": "active",*

*"role": "admin"*

*}*

*}*

*Sikertelen hozzáadás (angolul)*

*json*

*{*

*"errors": {*

*"edu\_id": ["The edu id format is invalid."],*

*"email": ["The email format is invalid."],*

*"password": ["The password must include at least one uppercase letter, one lowercase letter, and one number."]*

*}*

*}*

* **show / *GET*: /users/{id}**
  + **Leírás:**
    - Egy felhasználó adatainak lekérdezése az azonosító alapján.
  + **Funkciók**:
    - A metódus megkeresi a felhasználót az adatbázisban a megadott *id* alapján.
    - Ha a felhasználó nem található, 404-es hibát ad vissza, ha pedig megtalálható, a felhasználó adatait visszaadja.
  + **Paraméter:**
    - id: A user id-ját tartalmazza, amelyet a route paramétereként kap
  + **Válaszok**:
    - **Sikeres:**
      * A felhasználó adatainak JSON formátumban történő visszaadása.
    - **Hibás:**
      * Ha a felhasználó nem található:

*Sikeres lekérdezés*

*json*

*{*

*"id": 1,*

*"name": "Teszt User",*

*"email": "tesztuser@example.com",*

*"edu\_id": "712345678901",*

*"status": "active",*

*"role": "admin"*

*}*

*Sikertelen lekérdezés (angolul)*

*json*

*{*

*"message": "User not found"*

*}*

* **update** / *PUT*: /users/{id}
  + **Leírás:**
    - Egy felhasználó adatainak módosítása az azonosító alapján.
  + **Funkciók**:
    - A metódus megkeresi a felhasználót az adatbázisban a megadott id alapján.
    - A frissítendő mezők:
      * **name**: Ha van új érték, frissíti a felhasználó nevét.
      * **email**: Frissíti az email címet, ha szükséges, biztosítja, hogy az új email egyedi legyen.
      * **edu\_id**: Ha szükséges, frissíti az edu\_id értéket, és biztosítja az egyediségét.
      * **password**: Ha a jelszó változik, akkor titkosítja és frissíti azt.
      * **role**: A felhasználó szerepe is módosítható, például "admin" vagy "student".
  + **Paraméter:**
    - id: A user id-ját tartalmazza, amelyet a route paramétereként kap
  + **Válasz**:
    - **Sikeres:**
      * A frissített felhasználó adatai.
    - **Hiba:**
      * Ha a felhasználó nem található:

*Sikeres frissítés*

*json*

*{*

*"id": 1,*

*"name": "Teszt User",*

*"email": "tesztuser@example.com",*

*"edu\_id": "712345678901",*

*"status": "active",*

*"role": "student"*

*}*

*Sikertelen frissítés(angolul)*

*json*

*{*

*"message": "User not found"*

*}*

* **updateStatus** / *PUT*: /users/{id}/status
  + **Leírás:**
    - A felhasználó státuszának módosítása, például "active" vagy "inactive".
  + **Funkció**k:
    - A metódus megkeresi a felhasználót az adatbázisban a megadott id alapján.
    - Frissíti a státuszt, ha az új státusz "inactive" és a felhasználó "admin", akkor nem engedélyezi a státusz módosítását.
    - A státuszt validálja, hogy csak "active" vagy "inactive" értékek lehetségesek.
  + **Paraméter:**
    - id: A user id-ját tartalmazza, amelyet a route paramétereként kap
  + Válasz:
    - **Sikeres:**
      * A frissített státuszt és a felhasználó adatait visszaadja.
    - **Hibás:**
      * Ha a felhasználó nem található, vagy ha az admin státuszt próbálják inaktiválni:

*Sikeres frissítés(angolul)*

*json*

*{*

*"message": "User status updated",*

*"user": {*

*"id": 1,*

*"name": "Teszt User",*

*"email": "tesztuser@example.com",*

*"edu\_id": "712345678901",*

*"status": "inactive",*

*"role": "admin"*

*}*

*}*

*Sikertelen frissítés(angolul)*

*json*

*{*

*"message": "User not found"*

*}*

*Sikertelen frissítés(angolul)*

*json*

*{*

*"message": "Cannot deactivate admin user"*

*}*

**Modellek**

**Book Modell**

* **$fillable Tömb**
  + **book\_type\_id**: A könyv típusának azonosítója, amely a *BookType* modellel kapcsolódik.
  + **inventory\_number**: Az adott könyv egyedi azonosítója
* **Kapcsolat:**
  + A *bookType()* metódus a BookType modellel való kapcsolatot definiálja. Mivel a könyv típusát a book\_type\_id mező határozza meg, ez a metódus **egy-hez-több** kapcsolatot alakít ki a Book és a BookType modellek között.
  + A *belongsTo()* metódus a következőket jelenti:
    - A **Book** modell a **BookType** modellhez tartozik.
    - A **book\_type\_id** kulcsot használja a kapcsolat létrehozásához.

**BookType Modell**

* **$fillable Tömb**
  + **inventory\_number\_base**: Az alapértelmezett könyvazonosító, amelyet a könyv egyedi azonosítójához használnak.
  + **title**: A könyv címe.
  + **author**: A könyv szerzője.
  + **price**: A könyv ára.
  + **copies**: A könyv elérhető példányainak száma.
* **Kapcsolat**
  + Bár a **BookType** modell jelenleg nem tartalmaz explicite meghatározott kapcsolatokat más modellekkel, a **Book** modellel kapcsolatos kapcsolatot úgy kezelhetjük, hogy a **Book** modell *bookType()* metódusa a **BookType** modellhez kapcsolódik.
    - Ez azt jelenti, hogy egy **BookType**-nak több kapcsolódó **Book** példánya lehet.

**BorrowedBook Modell**

* **$fillable Tömb**
  + **user\_edu\_id**: A kölcsönző felhasználó **edu\_id** azonosítója.
  + **inventory\_number**: A kölcsönzött könyv **inventory\_number** (egyedi azonosítója).
  + **status**: A könyv kölcsönzési státusza (pl. "borrowed", "requested\_return", "returned").
* **Kapcsolatok**
  + A **BorrowedBook** modell két fontos relációt határoz meg:
    - **user()**: A kölcsönzött könyv egy **User** (felhasználó) objektumhoz tartozik. A kapcsolatot a belongsTo metódus határozza meg, amely a **User** modellhez kapcsolja a kölcsönzött könyvet a **user\_edu\_id** mező alapján.

*php*

*public function user()*

*{*

*return $this->belongsTo(User::class);*

*}*

* + - Ez a kapcsolat lehetővé teszi a kölcsönzött könyv felhasználójának lekérdezését.
* **book()**: A kölcsönzött könyv egy **Book** (könyv) objektumhoz tartozik. A belongsTo kapcsolat itt a **Book** modellhez kapcsolja a könyvet az **inventory\_number** alapján, amely a könyv egyedi azonosítója.

*php*

*public function book()*

*{*

*return $this->belongsTo(Book::class, 'inventory\_number', 'inventory\_number');*

*}*

* + Ez lehetővé teszi a kölcsönzött könyv adatait, például címét, szerzőjét, árát és egyéb információit lekérdezni.

**Student Modell**

* **$fillable Tömb**
  + **name**: A diák neve.
  + **edu\_id**: A diák egyedi azonosítója (pl. egy egyetemi azonosító, amely a users táblában is szerepel).
  + **status**: A diák státusza (pl. "active" vagy "inactive").
* **Kapcsolat (Relációk)**
  + A **Student** modellben egy **User** modellel létesített kapcsolatot találunk. A kapcsolat belongsTo típusú, mivel minden egyes diák egy felhasználóhoz tartozik.

*php*

*public function user()*

*{*

*return $this->belongsTo(User::class, 'edu\_id', 'edu\_id');*

*}*

* + Ez a *user()* metódus biztosítja, hogy minden diák adatainál lekérhetjük az adott diákhoz tartozó felhasználó adatokat. A kapcsolat a **Student** modell **edu\_id** mezőjét köti össze a **User** modell **edu\_id** mezőjével.

**User Modell**

* **$fillable és $hidden**
  + **name**: A felhasználó neve.
  + **email**: A felhasználó email címe.
  + **edu\_id**: Az egyedi azonosító (ami a diákoknál használt, és kapcsolódik a Student modellhez).
  + **password**: A felhasználó jelszava.
  + **role**: A felhasználó szerepköre (pl. "admin" vagy "student").
  + **status**: A felhasználó státusza (pl. "active" vagy "inactive").
  + **$hidden**: Azokat a mezőket tartalmazza, amelyek a modellek többi részében nem jelennek meg. Például a **password** és **remember\_token** mezők rejtve vannak, hogy ne jelenjenek meg, amikor a felhasználói adatokat visszaadjuk
* **Kapcsolat a Student modellel**
  + A User modell egy **hasOne** kapcsolatot definiál a **Student** modellel:

php

public function student()

{

return $this->hasOne(Student::class, 'edu\_id', 'edu\_id');

}

* + Ez azt jelenti, hogy minden **User** modellhez tartozik egy **Student** rekord, amely a **edu\_id** mezőt használja a kapcsolódásra. A *hasOne()* típusú kapcsolat azt jelenti, hogy egy felhasználóhoz egyetlen diák tartozik. Ezt a kapcsolatot az **edu\_id** alapján hozzák létre a két modell között.

**Seederek**

**BookSeeder**

1. **CSV fájl ellenőrzése és beolvasása**

*php*

*$csvFile = 'public/booktypes.csv';*

*if (!Storage::exists($csvFile)) {*

*$this->command->error("Hiba: A CSV fájl nem található!");*

*return;*

*}*

*$csvPath = Storage::path($csvFile);*

*$csv = Reader::createFromPath($csvPath, 'r');*

*$csv->setHeaderOffset(0);*

* **CSV fájl ellenőrzése**: Az első lépésben a kód ellenőrzi, hogy a **booktypes.csv** fájl létezik-e a public mappában.
  + Ha nem található a fájl, akkor egy hibaüzenetet ad vissza a konzolon, és leállítja a feldolgozást.
  + Ha megtalálja a fájlt, a *Storage::path()* segítségével lekéri a fájl abszolút elérési útját, és a *League\Csv\Reader* osztály segítségével beolvassa azt.
* **CSV fejlécek kezelése**: A CSV fájl első sorában találhatóak a mezők nevei, amiket a *setHeaderOffset(0)* utasítással az osztály azonosít.

1. **CSV rekordok feldolgozása**

*php*

*foreach ($csv as $record) {*

*$id = $record['id'];*

*$inventoryPrefix = $record['inventory\_number\_base'];*

*$title = $record['title'];*

*$author = $record['author'];*

*$price = (int) $record['price'];*

*$copies = (int) $record['copies'];*

* Az adatokat a CSV-ből egyesével olvassa be, és azokat a $record változóba menti.
* A rekordok minden egyes sora egy könyvet képvisel, amelynek mezői:
  + **id**: A könyv típusa.
  + **inventory\_number\_base**: Az alapértelmezett előtag az egyes könyvek azonosítóinak generálásához.
  + **title**: A könyv címe.
  + **author**: A szerző neve.
  + **price**: A könyv ára (egész számra konvertálva).
  + **copies**: A példányok száma, azaz hány példányban lesz a könyv elérhető.

1. **Könyvek hozzáadása**

*php*

*for ($i = 1; $i <= $copies; $i++) {*

*$inventoryNumber = sprintf("%s-%03d", $inventoryPrefix, $i);*

*if (Book::where('inventory\_number', $inventoryNumber)->exists()) {*

*continue;*

*}*

*Book::create([*

*'book\_type\_id' => $id,*

*'inventory\_number' => $inventoryNumber*

*]);*

*}*

* **Példányok létrehozása**: A kód a **copies** változótól függően ciklust futtat, hogy az adott könyvből annyi példányt hozzon létre, amennyit a CSV fájlban meghatározott.
* **Egyedi könyvazonosító generálása**: Minden egyes példányhoz egyedi inventory\_number azonosítót generálunk a következő formátumban: **<prefix>-<sorszám>**. Például ha a könyv prefixe "ABC" és három példány van, akkor az azonosítók: **ABC-001, ABC-002, ABC-003**.
* **Egyediség biztosítása**: Mielőtt új könyvet hoznánk létre, ellenőrizzük, hogy a **inventory\_number** már nem létezik-e az adatbázisban a *Book::where('inventory\_number', $inventoryNumber)->exists()* lekérdezéssel. Ha létezik, akkor kihagyjuk az adott rekordot, és nem hozunk létre duplikált könyvet.
* **Könyv létrehozása**: Ha az azonosító nem létezik, akkor létrehozzuk az új könyvet az adatbázisban a *Book::create()* segítségével.

1. **Hogyan futtathatjuk a BookSeeder-t**
2. **Seeder futtatása**: A parancs a db:seed parancs segítségével hívható meg:

bash

php artisan db:seed --class=BookSeeder

1. **Seeder tesztelése**: A könyveknek most már elérhetőknek kell lenniük az adatbázisban, és az alkalmazásban is használhatók a könyvek, például lekérdezéssel vagy kölcsönzés során.
2. **Tippek**

* Ha a fájl hiányzik vagy problémák merülnek fel a fájl elérhetőségével, ügyelj arra, hogy a fájl a megfelelő helyen legyen (**storage/app/private/public**).
* A CSV fájl hibamentes adatokat tartalmazzon (pl. érvényes számok a példányok számára, létező könyv azonosítók).

**BookTypeSeeder**

1. **CSV fájl ellenőrzése és beolvasása**

*php*

*$csvFile = 'public/booktypes.csv';*

*if (!Storage::exists($csvFile)) {*

*echo "CSV fájl nem található: " . Storage::path($csvFile) . "\n";*

*return;*

*}*

*$csvPath = Storage::path($csvFile);*

*$csv = Reader::createFromPath($csvPath, 'r');*

*$csv->setHeaderOffset(0);*

* Az első lépésben a *Storage::exists()* ellenőrzi, hogy a **booktypes.csv** fájl létezik-e a public mappában.
* Ha nem találja a fájlt, akkor kiír egy hibaüzenetet, és a seeder leáll.
* Ha a fájl megtalálható, a *Storage::path()* segítségével lekéri a fájl abszolút elérési útját, majd a *League\Csv\Reader* osztály segítségével beolvassa a fájlt.
* A *setHeaderOffset(0)* beállítás azt jelenti, hogy az első sor a fájlban a fejléc, tehát az oszlopnevek találhatóak benne.

1. **Rekordok feldolgozása és a könyv típusok hozzáadása**

*php*

*foreach ($csv as $record) {*

*$existingBookType = BookType::where('inventory\_number\_base', $record['inventory\_number\_base'])->first();*

*if (!$existingBookType) {*

*BookType::create([*

*'inventory\_number\_base' => $record['inventory\_number\_base'],*

*'title' => $record['title'],*

*'author' => $record['author'],*

*'price' => (int)$record['price'],*

*'copies' => (int)$record['copies'],*

*]);*

*}*

* **CSV rekordok feldolgozása**: A foreach ciklus minden egyes rekordot (sor) bejár a CSV fájlból.
* **Egyediség biztosítása**: A ciklusban először megnézi, hogy a könyv típus (**inventory\_number\_base**) már létezik-e az adatbázisban. Ha létezik, akkor nem csinál semmit, és átugorja a rekordot.
* **Új könyv típus hozzáadása**: Ha a könyv típus nem létezik, akkor új rekordot hoz létre a *BookType::create()* segítségével. Az oszlopok a CSV fájl adatai alapján kerülnek be a BookType modellbe:
  + **inventory\_number\_base**: Az alapértelmezett azonosító prefix a könyv típusokhoz.
  + **title**: A könyv címe.
  + **author**: A könyv szerzője.
  + **price**: A könyv ára (számként konvertálva).
  + **copies**: Az elérhető példányok száma (számként konvertálva).

1. **. Seeder futtatása**

*bash*

*php artisan db:seed --class=BookTypeSeeder*

**StudentSeeder**

1. **CSV fájl ellenőrzése és beolvasása**

*php*

*$csvFile = 'public\students.csv';*

*if (!Storage::exists($csvFile)) {*

*echo "CSV fájl nem található: " . Storage::path($csvFile) . "\n";*

*return;*

*}*

*$csvPath = Storage::path($csvFile);*

*$csv = Reader::createFromPath($csvPath, 'r');*

*$csv->setHeaderOffset(0);*

* **CSV fájl elérhetősége**: A kód először ellenőrzi, hogy létezik-e a **students.csv** fájl a public mappában. Ha nem található, hibaüzenetet ad, és leállítja a futást.
* **Fájl beolvasása**: Ha a fájl létezik, a *League\Csv\Reader* osztály segítségével beolvassa. A *setHeaderOffset(0)* beállítás biztosítja, hogy az első sor a fájlban fejléc legyen, ami oszlopneveket tartalmaz.

1. **Hallgató rekordok feldolgozása**

*php*

*foreach ($csv as $record) {*

*$existingStudent = Student::where('edu\_id', $record['edu\_id'])->first();*

*if (!$existingStudent) {*

*Student::create([*

*'name' => $record['name'],*

*'status' => 'active',*

*'edu\_id' => $record['edu\_id']*

*]);*

*}*

*}*

* **Egyediség biztosítása**: A kód minden egyes rekordot bejár a CSV fájlból, és ellenőrzi, hogy létezik-e már a hallgató az adatbázisban a edu\_id alapján. Ha a hallgató már létezik, nem csinál semmit, és átugorja a rekordot.
* **Új hallgató hozzáadása**: Ha a hallgató még nem szerepel az adatbázisban, akkor új rekordot hoz létre az **edu\_id**, **name**, és **status** mezőkkel. A status értékét active-ra állítja, alapértelmezetten.

1. **Seeder futtatása**

*bash*

*php artisan db:seed --class=StudentSeeder*

**UserSeeder**

1. **Felhasználó létrehozása**

*php*

*User::create([*

*'name' => 'Admin',*

*'email' => 'admin@example.com',*

*'password' => Hash::make('Admin123'),*

*'role' => 'admin',*

*]);*

* **name**: A felhasználó neve (Admin).
* **email**: A felhasználó email címe (admin@example.com).
* **password**: A felhasználó jelszava, amelyet a Hash::make() segítségével titkosítunk. A jelszó: Admin123.
* **role**: A felhasználó szerepe (admin), ami azt jelenti, hogy ez a felhasználó adminisztrátori jogosultságokkal rendelkezik.

1. **A seeder futtatása**

*bash*

*php artisan db:seed --class=UserSeeder*

**További javaslatok:**

* **Jelszó erőssége**: Bár az admin felhasználó jelszavának erősnek kell lennie, ügyelj arra, hogy a **Admin123** jelszó nem biztosítja a legmagasabb biztonságot, mivel elég egyszerű. Érdemes erősebb jelszót választani az éles környezetben.
* **Role és jogosultságok**: Az **admin** szerepkört célszerű lehet dinamikusan kezelni, például jogosultságok beállításával, hogy különböző felhasználók különböző szintű hozzáférést kapjanak.
* **Hiba kezelés**: Ha már létezik admin felhasználó, célszerű elkerülni a duplikált rekordok létrehozását. Ehhez érdemes leellenőrizni, hogy létezik-e már admin felhasználó:

*php*

*if (!User::where('role', 'admin')->exists()) {*

*User::create([*

*'name' => 'Admin',*

*'email' => 'admin@example.com',*

*'password' => Hash::make('Admin123'),*

*'role' => 'admin',*

*]);*

*}*

**DatabaseSeeder**

1. **Seeder osztályok futtatása**

*php*

*$this->call([*

*UserSeeder::class,*

*BooktypeSeeder::class,*

*BookSeeder::class,*

*StudentSeeder::class,*

*]);*

Itt a run metódusban a call függvény segítségével a különböző seeder osztályokat hívjuk meg. Minden egyes seeder osztály egy-egy táblát inicializál, és adatokat tölt be az adatbázisba.

* **UserSeeder::class**: Az admin felhasználó létrehozása. (A UserSeeder biztosítja, hogy legyen egy admin felhasználó az alkalmazásban.)
* **BooktypeSeeder::class**: A könyv típusok adatainak feltöltése. (A BookTypeSeeder tölti fel a könyv típusokat.)
* **BookSeeder::class**: A könyvek adatainak feltöltése. (A BookSeeder pedig a könyveket hozza létre a megadott BookType típusok alapján.)
* **StudentSeeder::class**: A diákok adatainak feltöltése. (A StudentSeeder az edu\_id és más alapadatok alapján hozza létre a diákokat.)

**2. A Seeder futtatása**

bash

php artisan db:seed

**3. Új Seeder osztályok hozzáadása**

Ha új táblát vagy új adatokat szeretnél betölteni a rendszerbe, egyszerűen létrehozhatsz egy új seeder osztályt, és hozzáadhatod a DatabaseSeeder-hez a következő módon:

* Például, ha szeretnél egy CategorySeeder osztályt a könyvek kategóriáinak feltöltésére, létrehozhatod a CategorySeeder osztályt, és hozzáadhatod a DatabaseSeeder-hez:

*php*

*$this->call([*

*UserSeeder::class,*

*BooktypeSeeder::class,*

*BookSeeder::class,*

*StudentSeeder::class,*

*CategorySeeder::class, // Új seeder hozzáadása*

*]);*

**Javaslatok:**

1. **Többszörös futtatás**: Ha több seeder osztály van, célszerű lehet először azokat az adatokat betölteni, amelyek más adatokhoz kapcsolódnak. Például a **BookSeeder**-t csak akkor futtathatjuk, miután a **BooktypeSeeder** lefutott, hogy a könyvek a megfelelő típusokkal rendelkezzenek.
2. **Adatellenőrzés**: A seeder osztályoknál célszerű lehet adatellenőrzést végezni, hogy elkerüljük az adatduplikációt. Például, ha egy könyv típus már létezik, akkor ne hozd létre újra.
3. **Adatok teszteléshez**: A fejlesztés során tesztadatokkal is feltöltheted az adatbázist, például különböző felhasználói szerepekkel, könyvekkel vagy diákokkal, hogy minden funkció működését biztosítani tudd.

**Befejezés**

A könyvtárkezelő rendszer fejlesztése során számos izgalmas élményben volt részünk, miközben próbáltuk megvalósítani a projekt céljait. Az alkalmazás létrehozása közben nemcsak technikai kihívásokkal kellett szembenéznünk, hanem rengeteg új tapasztalattal gazdagodtunk is. Bár az út nem volt mentes a nehézségektől, minden egyes megoldott probléma egyre inkább megerősítette bennünk, hogy jó irányba haladunk.

Az egyik legnagyobb kihívás számunkra a rendszer különböző technológiai komponenseinek integrálása volt. Mivel Laravel-t használtunk a backend fejlesztéséhez, Angular-t a frontendhez és XAMPP-t az adatbázis kezeléséhez, nem mindig volt egyszerű biztosítani, hogy minden zökkenőmentesen működjön együtt. Sok órát töltöttünk a dokumentációk tanulmányozásával, és próbálkoztunk különböző megoldásokkal, hogy az egyes rendszerek kompatibilisek legyenek. Az egyik legnehezebb pillanat talán akkor jött el, amikor egy fontos adatbázis-kapcsolatot nem sikerült elsőre beállítanunk, és órákig hibakereséssel próbáltuk megoldani. Az ilyen pillanatok igazán próbára tették a türelmünket, de végül sikerült mindent megoldani, és az érzés, hogy egy-egy problémát sikerült leküzdeni, mindent megért.

A projekt másik fontos tanulsága az volt, hogy a felhasználói élményre való fókuszálás elengedhetetlen. Az alkalmazásnak nemcsak működnie kellett, hanem könnyen használhatónak is. Az első tesztelési fázis során sok visszajelzést kaptunk, amely segített finomítani a kezelőfelületet. Az egyszerű navigáció, a gyors keresési funkciók és a világos, átlátható struktúra mind olyan elemek voltak, amelyek folyamatosan fejlődtek, miközben próbáltuk az összes felhasználói igényt figyelembe venni.

A rendszer fejlesztése során nemcsak technikai ismereteket szereztünk, hanem rengeteg új szemléletmódot és problémamegoldó készséget is elsajátítottunk. A folyamatos tanulás és a problémák gyors megoldása során szorosabbá vált a csapatmunka, és ráébredtünk, hogy a sikeres fejlesztés egyik kulcsa a kitartás és a rugalmasság. Az ilyen típusú kihívások valóban segítenek abban, hogy fejlődjünk, és felkészültebben álljunk a jövőbeli projektek elé.

Összegzésül, a projekt nemcsak egy technikai megoldás létrehozásáról szólt, hanem arról is, hogy egy csapatként hogyan tudunk együtt dolgozni a problémák megoldásán. Bár a munka során nem mindig volt minden egyszerű, úgy érezzük, hogy az összes nehézség és kihívás meghozta gyümölcsét, és egy olyan rendszert sikerült alkotnunk, amely hosszú távon is segíthet a könyvtári folyamatok hatékony kezelésében.