Külkereskedelmi Technikum

**Byte Brigade**

**Dokumentáció**

**BOOKHUB**

**online használtkönyv-csere platform**



Készítették:

Balázs Réka, Gerecs Diána, Csajbók-Reményi László

Budapest

2024

**Tartalomjegyzék**

1.Bevezetés1

1.1 Témaválasztás indoklása2

1.2 AI által generált képek használata3

1.3 Hasonló weboldal3

1.4 A project jelenlegi állapota és a jövője3

2. Fejlesztői dokumentáció6

2.1 Fejlesztői környezet6

2.2 Fejlesztéshez használt operációs rendszerek, böngészők6

2.3 Fejlesztéshez használt programok, programozási nyelvek és eszközök7

2.4 Adatbázis11

2.5 Változók és mezők elnevezése16

2.6 Frontend kialakítása Angulárral17

2.7 Osztályok elnevezése a Bulma keretrendszeréhez igazodva19

2.8 Backend megvalósítása SpringBoot alkalmazással20

2.9 Fejlesztési lehetőségek27

3 Tesztelés28

4 Felhasználói dokumentáció31

4.1 Böngészés a honlapon31

4.2 Regisztráció a honlapon31

4.3 Elérhető funkciók a bejelentkezés után32

4.4 Hibajelzések jelentése33

4.5 Elérhető funkciók adminként34

**1 Bevezetés**

**1.1 Témaválasztás**

*„A lélek hőmérője: a könyv.  Mondd, meg mit olvasol és megmondom: mi vagy. Megmondom mi az akaratod, célod, vágyad, álmod.” (Juhász Gyula)*

A Byte Brigade által megálmodott és megalkotott Bookhub vállalat weblapja alkalmas arra, hogy azokat az embereket kösse össze, akik szeretnek olvasni, és szívesen adják tovább azt, amit már elolvastak, vagy keresgélnek azok között a könyvek között, amik elérhetőek. Ezáltal ismeretségek születhetnek azonos érdeklődésű emberek között.

Mindeközben felhasználóink felelősséggel viseltetnek a környezet felé, tisztában vannak azzal, hogy rajtunk múlik, mit adunk tovább a gyermekeinknek. Fejlesztőként ehhez azzal járulhatunk hozzá, hogy lehetőséget biztosítunk arra, hogy a saját könyveiket szabadon felajánlják, cserélgessék egymás között, mindezt olyan rendezett körülmények között, amely egyszerre védi a felhasználók adatait, és minél inkább csökkenti a lehetőségét annak, hogy valaki visszaéljen a felajánlók jóindulatával.

Szeretnénk elérni, hogy amikor valaki tovább szeretné adni a könyveit bármilyen oknál fogva, akkor ne ütközzön abba az akadályba, hogy nem tudja kinek, és emiatt egy jó állapotú könyv a hulladékban végzi, pedig más szívesen olvasná, csak nincs rá módja, hogy megvegye, esetleg nem is tud róla, hogy másnak a birtokában van, de már nincsen szüksége rá. Mindezeken túl olyan könyveket is megóvhatnánk a kidobástól, amiket már csak használtan lehet megszerezni egy-egy antikváriumban, pedig az oktatás során egyszerűen nélkülözhetetlen.

Nagy mennyiségű könyvek cseréjét az adminoknak lehetne jelezni, az adminisztrációs terhek csökkentése érdekében. Leginkább felsőfokú oktatási intézményekben jellemző, hogy a tantárgy oktatási színvonalát jelentősen csökkentheti egy-egy már el nem érhető könyv, amit a tanulók csak addig használnak, amíg meg nem szerzik az oklevelet.

**1.2 AI által generált képek használata**

A tervezgetés kezdeti fázisában elkezdtünk AI által generált képeket gyártani Hamar kialakult az egyetértés abban, hogy ezek megfelelő alapot fognak adni a vizuális megjelenéshez.

Az első a *Gencraft – AI Art Image and Video Generator* (<https://gencraft.com/>) volt, ezzel ingyenesen egy nap korlátos számú képet lehet legenerálni.

A második az *Image Creator from Microsoft Designer* ([A Microsoft Tervező Képkészítője (bing.com)](https://www.bing.com/images/create?toWww=1&redig=54A8DCDEEAC14C918E6B53B3D6112638)), ezzel korlátlan számú kép generálható egy nap, de az első 15 gyorsabban generálható.

A téma alapját a mackók adták, mégpedig azért, mert a fejlesztés korai szakaszában Gerecs Diána fejlesztő javaslatára bevezettük a rendszeres gumicukor fogyasztást, ami többnyire mackó formájú volt. Így aztán a mackók uralták a témákat a képeknél. Külön-külön generáltuk a képeket Csajbók-Reményik László és Balázs Réka. Az online találkozók alkalmával közösen döntöttük el, hogy melyik felületre melyik kép kerülhet. Tapasztalatok szerint lényegesen több képet kell generálni, mint amennyi végül is jó lesz.

A képek generálása közben sokat megtanultunk arról, hogy hogyan lehet és érdemes kiadni az utasításokat olyan módon a mesterséges intelligenciának, hogy az olyan képeket generáljon, amik közel állnak az elképzeléseinkhez. A végeredmény magáért beszél, azok a külső szemlélők, akik megtekintették a weblapot ezekkel a képekkel egyöntetűen elismerően nyilatkoztak róluk.

**1.3 Hasonló weboldal**

Jelenleg hasonló működő weboldalt nem találtunk, ami Magyarországon működik, ingyenesen elérhető, és használható. Azonban több olyan fórumot is találtunk, ahol igény mutatkozik arra, hogy jó lenne egy ilyen szolgáltatás, ahol a használt könyveket cserélhetik a regisztrált felhasználók.

Több nagyvárosban vannak kihelyezett ingyenes könyvmegállók, és ezeket szívesen használják az emberek. Ide azonban személyesen kell elmenni, és azok számára, akik közelében nincsen ilyen kihelyezve nem jelent megoldást. Arról nem beszélve, hogy sokkal egyszerűbb úgy keresni a tartalmak között, hogy csak akkor veszi fel valaki a kapcsolatot a felajánlóval, ha az ő számára érdekes könyv van a kínálatban. A kihelyezett könyvek esetében azonban ez a kényelmesebb áttekintése a választéknak egyszerűen nem lehetséges.

**1.4 A projekt jelenlegi állapota és jövője**

A projekt jelenlegi állapotában csak egy elképzelt vállalat egy jól felépített szerethető weboldallal, olyan képekkel, amiket mindenki szívesen nézeget, és aminek sokan örülnének, ha megvalósulna. Egyelőre nem kerestünk hozzá támogatót, és nem tervezzük azt, hogy ezt valóban életre keltsük, azonban nem is zárjuk ki teljesen ennek az lehetőségét.

A korlátokat egyértelműen az anyagi forrás megteremtése okozza, illetve az, hogy jelenleg egyikünk sem dolgozik főállásban fejlesztőként, így nehézséget okozna a fenntartása és üzemeltetése annak ellenére, hogy akiknek megmutattuk tetszetősnek találták, és látnának benne fantáziát.

**2. Fejlesztői dokumentáció**

**2.1 Fejlesztői környezet**

A hozzáférési lehetőségek úgy vannak kialakítva, hogy minden admin hozzáfér a forráskódokhoz, aki meghívást kapott a közös Github tárolóhoz. A backend és a frontend illetve a dokumentáció egyértelműen elkülönül a könyvtárszerkezetben. A fejlesztés közben verzió követést alkalmaztunk, az aktuális fejlesztés feltöltése előtt letöltöttük a további változtatásokat, ezen kívül rendszeresen tartottunk online megbeszéléseket zoom-on.

A fejlesztés folytatásához elsőként a Github-ról kell letölteni a program aktuális állapotát. A változtatások feltöltéséhez jogosultság szükséges.

Github link : <https://github.com/BalazsR2022/Byte_Brigade.git>

Az Angulár 16.0.1-es verziót használtuk, ehhez igazodjon a többi letöltött program kompatibilis verziója. A letöltendő programok listája az alábbiakban a használt programok cím alatt található.

A frontend indítása az Visual Studio Code-ban történik, npm start paranccsal. A XAMPP Controll panelen az Apache ,MySQL, Tomcat indítása szükséges. A helyi gépen létre kell hozni egy book nevű adatbázis-t. A backend indítása az IntelliJ IDEA-ben lehetséges, futtatáskor feltölti adatokkal az adatbázist.

**2.2 Fejlesztéshez használt operációs rendszerek:**

|  |  |
| --- | --- |
|  | Windows |
| A képen fekete, kör, fekete-fehér, csillagászat látható  Automatikusan generált leírás | Debian |

**Fejlesztéshez használt böngészők:**

|  |  |
| --- | --- |
|  | Edge |
| A képen kör, Színesség, Grafika, Grafikus tervezés látható  Automatikusan generált leírás | Chrome |
| A képen kör, Színesség, gömb, bolygó látható  Automatikusan generált leírás | Chromium |

**Fejlesztéshez használt programok:**

|  |  |
| --- | --- |
| logo | Visual Studio Code |
| A képen szimbólum, Betűtípus, Grafika, képernyőkép látható  Automatikusan generált leírás | IntelliJ |
| A képen szimbólum látható  Automatikusan generált leírás | Angular |
|  | Bulma keretrendszer |
| A képen szimbólum, Grafika, tervezés látható  Automatikusan generált leírás | XAMPP |
| A képen szöveg, embléma, Betűtípus, tervezés látható  Automatikusan generált leírás | PHPMyAdmin |
| A képen óra, multimédia, tervezés látható  Automatikusan generált leírás | Terminátor |
|  | Chocolatey |
| A képen fekete, sötétség látható  Automatikusan generált leírás | Postman |

**Fejlesztéshez használt programozási nyelvek:**

|  |  |
| --- | --- |
| **A képen fekete, sötétség látható  Automatikusan generált leírás** | **HTML** |
| **A képen tervezés, művészet látható  Automatikusan generált leírás alacsony megbízhatósággal** | **JAVA** |
| **A képen Grafika, Színesség, zöld látható  Automatikusan generált leírás** | **CSS** |
| A képen embléma, Betűtípus, szimbólum, szöveg látható  Automatikusan generált leírás | **PHP** |
|  | **JavaScript** |

**Fejlesztéshez használt eszközök**

DESKTOP-P5Q6LET Processzor AMD Ryzen 5 3600 6-Core Processor 3.59 GHz Memória mérete 16,0 GB , Termékazonosító 00330-80000-00000-AA780 Rendszer típusa 64 bites operációs rendszer, x64-alapú processzor Kiadás Windows 10 Pro Verzió 22H2 Minőség Windows Feature Experience Pack 1000.19054.1000.0

ASUS-VIVOBOOK X515EA-BQ1182 15.6’’ FullHD laptop, Intel Core i3-115G4, 8GB, 256G SSd, Intel

ASUS VIVOBOOK S15 S513 OLED Intel® Core™ i7-1165G7 Processor (12M Cache, up to 4.70 GHz, with IPU) 16GB DDR4 3200MHz RAM (2x8GB) Intel® Iris® Xe Graphics

**2.4 Adatbázis**

A képen szöveg, Betűtípus, szoftver, szám látható

Automatikusan generált leírás

**A honlap adattáblái a következőket tartalmazzák:**

**books**

**id**: Ez az elsődleges kulcs, azaz minden sorban egyedi azonosítót jelöl. Típusa **int(11)**, ami egy egész számot jelent maximum 11 számjeggyel. Az **AUTO\_INCREMENT** azt jelenti, hogy az adatbázis automatikusan növeli az értéket minden új sor hozzáadásakor. Ezáltal minden sor egyedi azonosítót kap, amely egyedülállóan azonosítja a könyvet az adatbázisban.

**author**: Ez a könyv szerzőjét tárolja. Típusa **varchar(50)**, ami azt jelenti, hogy legfeljebb 50 karakter hosszú karakterláncot fogad el. Ez a mező elfogadja az ékezetes karaktereket is (UTF-8 kódolás), és az adatbázis rendezési szabályai az általános értelmezésen alapulnak (**utf8mb4\_general\_ci**).

**category**: A könyv kategóriáját tárolja. Ez is egy karakterlánc (**varchar(30)**), ami legfeljebb 30 karakter hosszú lehet. Az ékezetes karaktereket elfogadja, és az általános értelmezésű rendezési szabályokat alkalmazza.

**county**: Ez a könyv kiadóját tárolja. Típusa hasonló a szerzőhöz (**varchar(50)**), elfogadva az ékezetes karaktereket, és az általános értelmezésű rendezési szabályokat alkalmazva.

**publisher**: Ez a könyv kiadóját tárolja. Hasonlóan a szerző és a kategória mezőkhöz, típusa **varchar(50)** és az általános értelmezésű rendezési szabályokat alkalmazza.

**quality**: Ez a könyv minőségét tárolja. Ez is egy karakterlánc (**varchar(50)**), elfogadva az ékezetes karaktereket, és az általános értelmezésű rendezési szabályokat alkalmazva.

**reserved**: Ez a mező egy bit típusú, amely azt jelzi, hogy a könyv foglalt-e vagy sem. A bit típus általában 0 vagy 1 értékeket tárol.

**title**: A könyv címét tárolja. Ez a mező legfeljebb 100 karakter hosszú karakterláncot fogad el (**varchar(100)**), elfogadva az ékezetes karaktereket, és az általános értelmezésű rendezési szabályokat alkalmazva.

**year**: Ez a könyv kiadási évét tárolja. Típusa **int(11)**, ami egy egész számot jelent maximum 11 számjeggyel.

**owner\_id**: Ez az index mező arra utal, hogy van-e egy másik tábla, amelyben az adott könyv tulajdonosát tárolják. Ez egy egész szám (**int(11)**), ami egy másik tábla elsődleges kulcsára mutathat.

Ezek a mezők és tulajdonságaik segítenek az adatbázisban tárolt könyvek részletes információinak kezelésében és lekérdezésében.

**bookussers**

**id**: Ez az elsődleges kulcs, ami egyedi azonosítót biztosít minden felhasználó számára az adatbázisban. Típusa **int(11)**, ami egy egész számot jelent maximum 11 számjeggyel. Az **AUTO\_INCREMENT** azt jelenti, hogy az adatbázis automatikusan növeli az értéket minden új sor hozzáadásakor. Így minden új felhasználó egyedi azonosítót kap.

**email**: Ez a felhasználó e-mail címét tárolja. Típusa **varchar(100)**, tehát legfeljebb 100 karakter hosszú karakterláncot fogad el. Az **utf8mb4\_general\_ci** jelöli, hogy az adatbázis a karaktereket az UTF-8 kódolás szerint értelmezi és rendezteti.

**is\_admin**: Ez a mező egy bit típusú, amely azt jelzi, hogy a felhasználó rendszergazdai jogosultságokkal rendelkezik-e vagy sem. A bit típus általában 0 vagy 1 értékeket tárol, tehát ez az adat tárolja, hogy az adott felhasználó rendszergazda-e vagy sem.

**logged\_in**: Ez a mező egy bit típusú, amely azt jelzi, hogy a felhasználó be van-e jelentkezve a rendszerbe vagy sem. Ez az információ segíthet abban, hogy az alkalmazás tudja, hogy mely felhasználók aktívak éppen a rendszerben.

**password**: Ez a felhasználó jelszavát tárolja. Típusa **varchar(100)**, tehát legfeljebb 100 karakter hosszú karakterláncot fogad el. Az **utf8mb4\_general\_ci** jelöli, hogy az adatbázis a karaktereket az UTF-8 kódolás szerint értelmezi és rendezteti.

**role**: Ez a mező a felhasználó szerepét vagy jogosultsági szintjét tárolja. Típusa **varchar(255)**, tehát legfeljebb 255 karakter hosszú karakterláncot fogad el. Ez az adat segíthet abban, hogy az alkalmazás tudja, hogy a felhasználó milyen jogosultságokkal rendelkezik.

**username**: Ez a felhasználó felhasználónevét tárolja. Típusa **varchar(50)**, tehát legfeljebb 50 karakter hosszú karakterláncot fogad el. Az **utf8mb4\_general\_ci** jelöli, hogy az adatbázis a karaktereket az UTF-8 kódolás szerint értelmezi és rendezteti.

Ezek a mezők és tulajdonságaik segítenek az adatbázisban tárolt könyvek részletes információinak kezelésében és lekérdezésében.

**role**

**id**: Ez az elsődleges kulcs, ami egyedi azonosítót biztosít minden szerep számára az adatbázisban. Típusa **int(11)**, ami egy egész számot jelent maximum 11 számjeggyel. Az **AUTO\_INCREMENT** azt jelenti, hogy az adatbázis automatikusan növeli az értéket minden új sor hozzáadásakor. Így minden új szerep egyedi azonosítót kap.

**name**: Ez a szerep nevét tárolja. Típusa **varchar(255)**, tehát legfeljebb 255 karakter hosszú karakterláncot fogad el. Az **utf8mb4\_general\_ci** jelöli, hogy az adatbázis a karaktereket az UTF-8 kódolás szerint értelmezi és rendezteti.

Ez a mező segít abban, hogy az adatbázisban tárolt különböző szerepek (például felhasználók, adminisztrátorok stb.) egyszerűen azonosíthatóak és kezelhetőek legyenek. Az id mező egyedi azonosítója lehetővé teszi, hogy minden szerepnek legyen egy sajátos azonosítója, a name mező pedig magát a szerep nevét tárolja.

**user role**

**id**: Ez az elsődleges kulcs, ami egyedi azonosítót biztosít minden felhasználó szerepkapcsolatához az adatbázisban. Típusa **int(11)**, ami egy egész számot jelent, maximum 11 számjeggyel. Az **AUTO\_INCREMENT** azt jelenti, hogy az adatbázis automatikusan növeli az értéket minden új sor hozzáadásakor, így minden új szerepkapcsolat egyedi azonosítót kap.

**name**: Ez a mező tárolja a felhasználó nevét, akihez a szerepkapcsolat tartozik. Típusa **varchar(255)**, tehát legfeljebb 255 karakter hosszú karakterláncot fogad el. Az **utf8mb4\_general\_ci** jelöli, hogy az adatbázis a karaktereket az UTF-8 kódolás szerint értelmezi és rendezteti.

**role**: Ez a mező tárolja a felhasználóhoz rendelt szerepet. Típusa ugyanúgy **varchar(255)**, tehát legfeljebb 255 karakter hosszú karakterláncot fogad el. Az **utf8mb4\_general\_ci** jelöli, hogy az adatbázis a karaktereket az UTF-8 kódolás szerint értelmezi és rendezteti.

Ezek a mezők segítenek az adatbázisban tárolt felhasználók és szerepek közötti kapcsolatok részletes kezelésében és lekérdezésében. Az id mező egyedi azonosítója lehetővé teszi, hogy minden szerepkapcsolatnak legyen egy sajátos azonosítója, a name mező a felhasználó nevét tárolja, a role mező pedig a felhasználóhoz rendelt szerepet.

**user roles**

**id**: Ez az elsődleges kulcs, ami egyedi azonosítót biztosít minden szerepkapcsolatnak az adatbázisban. Típusa **int(11)**, ami egy egész számot jelent maximum 11 számjeggyel. Az **AUTO\_INCREMENT** azt jelenti, hogy az adatbázis automatikusan növeli az értéket minden új sor hozzáadásakor, így minden új szerepkapcsolat egyedi azonosítót kap.

**role**: Ez a mező tárolja a felhasználóhoz rendelt szerepet. Típusa **varchar(255)**, tehát legfeljebb 255 karakter hosszú karakterláncot fogad el. Az **utf8mb4\_general\_ci** jelöli, hogy az adatbázis a karaktereket az UTF-8 kódolás szerint értelmezi és rendezteti.

**user\_id**: Ez egy index, ami egy másik tábla, a user role táblája felé mutat. Ez azért fontos, mert kapcsolatot teremt a szerepkapcsolat és a felhasználók között. Típusa **int(11)**, ami egy egész számot jelent maximum 11 számjeggyel.

Ezek a mezők segítenek az adatbázisban tárolt szerepkapcsolatok részletes kezelésében és lekérdezésében. Az id mező egyedi azonosítója lehetővé teszi, hogy minden szerepkapcsolatnak legyen egy sajátos azonosítója, a role mező pedig a felhasználóhoz rendelt szerepet tárolja. A user\_id mező pedig megmutatja, melyik felhasználóhoz tartozik a szerep. Az index ezen a mezőn lehetővé teszi az adatok hatékonyabb lekérdezését és kezelését.

**2.5 Változók és mezők elnevezése**

A frontendben és a backendben is a tiszta kódnak megfelelő változóneveket használtunk, amelyek egyértelműen utalnak az adott változó céljára és használatára.

Frontend deklarált változónevek:

* user (regisztrált felhasználó)
* name(regisztrált felhasználó neve)
* email(regisztrált felhasználó emailcíme)
* password(regisztrált felhasználó jelszava)
* county(regisztrált felhasználó által jelölt megye)

Backend deklarált változónevek:

* book (felajánlott könyv)
* id (könyv id-ja)
* title (könyv címe)
* author (könyv szerzője)
* publisher (felajánló regisztrált felhasználó)
* category (könyv kategóriája)
* county (regisztrált felhasználó által jelölt megye)
* quality (könyv állapota)

A regisztrációkor megadott adatokat tároljuk változókban, azoknak vizsgálata csak a formai követelményeknek való megfelelés szempontjából szükséges. Azok valódiságát semmilyen szempontból nem vizsgáljuk, tartalmukért a regisztrált felhasználó a felelős.

**2.6 Frontend kialakítása Angulárral**

Az Angular használatában hamar meg tudtunk állapodni. Mellette szóltak a következő érvek:

1. *Keresztplatformos támogatás:* Angular támogatja a keresztplatformos fejlesztést, amely lehetővé teszi az alkalmazások futtatását többféle platformon, beleértve a webböngészőket, mobilalkalmazásokat és asztali alkalmazásokat is.
2. *Kiterjedt funkciókészlet:* Angular számos beépített funkciója és modullja megkönnyíti a fejlesztést, mint például komponensek, modulok, szolgáltatások, HTTP kliens, formokkezelés.
3. *MVC architektúra:* Az Angular keretrendszer MVC (Model-View-Controller) architektúrát alkalmaz, amely lehetővé teszi az alkalmazások komponensekre és modulokra történő logikus felosztását, ami javítja az alkalmazás karbantarthatóságát és skálázhatóságát.
4. *Egyértelmű dokumentáció*: Az Angularnak nagyon jó dokumentációja van, és mivel már azelőtt elkezdtük az ismerkedést a programmal, hogy az iskolában tananyag lett volna, mindeképpen olyan keretrendszert keretünk, amit egyedül is el tudunk kezdeni tanulni.
5. *Tesztelhetőség:* Angular támogatja az egységteszteket és az integrációs teszteket, amelyek segítenek biztosítani az alkalmazások stabilitását és megbízhatóságát. Egyszerűsíti a tesztelést a projektmunkában.
6. *Optimalizált teljesítmény:* Az Angular beépített eszközökkel rendelkezik a teljesítmény optimalizálására, gyors betöltési időt tesz lehetővé. Ez egy honlapnál jelentős előnyt jelent. A felhasználók nem szeretnek sokat várni egy honlap betöltődésére.
7. *Rugalmas adatkötés*: Az Angular kétirányú adatkötést biztosít, amely lehetővé teszi az adatok dinamikus megjelenítését az alkalmazás felületén, és azonnali frissítést biztosít az adatok megváltozása esetén.

|  |  |
| --- | --- |
| ***komponens neve*** | ***feladata*** |
| *about* | bemutatkozás, célok bemutatása, küldetés meghatározása |
| *books-admin* | a könyvek kezelésére szolgáló felület |
| *bulma-card* | dinamikusan generált listát jelenít meg a könyvekről |
| *bulma-footer* | láblécet jelenít meg a honlapon |
| *contact* | kapcsolati adatok megjelenítése |
| *home* | a Bulma Card komponensét használja a felhasználói felületének megjelenítésére |
| *login* | bejelentkezéshez szükséges űrlapot jelenít meg |
| *navbar* | navigációs sávot jelenít meg a honlap tetején. A logóra való kattintás visszaviszi az oldal főoldalára míg a hamburger ikonra való kattintás megnyitja vagy bezárja a mobilmenüt. |
| *redbutton* | a felhasználót visszanavigálja arra az oldalra, ahol nem a jó gombot nyomta meg |
| *service* | bemutatja a honlapon igénybevehető szolgáltatásokat |
| *signup* | feliratkozáshoz szükséges űrlapot jelenít meg |
| *surprise* | lehetőséget biztosít a felhasználónak egy véletlenszerű könyv választására |
| *this-book* | fejlesztés után lehetőséget biztosíthat egy könyv adatainak a szerkesztésére |
| *user* | megjeleníti a regisztrált felhasználó adatait |

**2.7 Osztályok elnevezése a Bulma keretrendszer szerint**

Például az alábbi kódrészletben a container osztály használatával a div elem felveszi a konténer tulajdonságait. és a tartalom középre igazítva jelenik meg igazodva az oldal szélességéhez

A column class segítségével a "has-text-primary-dark" egy osztálynevet jelöl, amelyet a Bulma CSS keretrendszer használ. Ez a stílus beállítja a szöveg színét sötét árnyalatú kékre. A Bulma dokumentációban kereshetető ki a megfelelő színárnyalat, amit látni szeretnénk a megjelenő honlapon. A „columns” class hozzáadásával az alárendelt div elem a Bulma grid rendszerének megfelelően jelenik meg, ami sokkal esztétikusabb megjelenítést tesz lehetővé az eredeti html kódhoz képest egy egyszerű kódmódosítással.

A margin értéket sem szükséges külön beállítani a css kódban, az mt-2 class hozzáadásával margin-top értéket ad az oszlopoknak, így a felső margót növelve beállítja a megfelelő távolságot a honlapon megjelenő elemek között. A column class az oszlopok egymás mellé rendezésében segít.

A card class kártyaszerű megjelenést ad a benne elhelyezett elemeknek. Ez a kártya szerű megjelenés leegyszerűsítette a képek megjelenését és formázását az oldalon, egységes megjelenést tett lehetővé.

Az alábbi szín a következő színkódot jelöli a táblázat szerint: "hsl(171, 100%, 29%)", amely a HSL (Hue, Saturation, Lightness) színmodell alapján határozza meg a szín árnyalatát, telítettségét és világosságát. Ebben az esetben a szín egy kék árnyalata.

A title class használatával a paragraph elem kep kiemelt megjelenést további formázás nélkül is.

<div class="container">

  <div class="columns mt-2">

    <div class="column">

      <div class="card has-background-primary-light">

        <div class="card-content">

          <p class="title has-text-primary-dark has-text-primary-dark">

            "Olvass, cserélj, játssz."

          </p>

**2.8 Backend megvalósítása SpringBoot alkalmazással**

A backend a SpringBoot alkalmazásban lett megvalósítva. A SpringBoot használata nagyban egyszerűsítette a backend megvalósítását beépített mechanizmusok segítségével. Ezeknek a funkciónak az eléréséhez importokat kell az egyes fájlok elején használni.

A program belépési pontja az alábbiakban található *BackendApplication.java* fájl. A *public static void main(String[] args* metódus van benne definiálva, amely a program végrehajtásának kezdetén indul el. Ez a metódus hívja meg a Spring Boot alkalmazás indításához szükséges *SpringApplication.run(BackEndApplication.class, args)* függvényt.

A fájl tartalmazza továbbá az *@SpringBootApplication* annotációt is, ami azt jelzi, hogy ez a fájl a Spring Boot alkalmazás fő konfigurációs osztálya. Az annotáció kombinálja a *@Configuration, @EnableAutoConfiguration, és @ComponentScan* annotációkat, ezáltal automatikusan konfigurálja a Spring alkalmazást és betölti az összes szükséges beállítást. Ez teszi lehetővé, hogy az alkalmazás könnyedén futtatható legyen a Spring Boot alapértelmezett beállításaival. A fájlban található *@Bean* annotáció pedig egy *PasswordEncoder* típusú bean létrehozására szolgál. Ez a bean a jelszavak titkosítására szolgáló *BCryptPasswordEncoder* példányt hozza létre, amelyet később a biztonsági konfigurációban lehet használni.

package com.geolidth.BackEnd;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.context.annotation.Bean;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.web.servlet.config.annotation.CorsRegistry;  
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;  
  
  
@SpringBootApplication  
public class BackEndApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(BackEndApplication.class, args);  
 }  
 @Bean  
 public WebMvcConfigurer corsConfigurer() {  
 return new WebMvcConfigurer() {  
 @Override  
 public void addCorsMappings(CorsRegistry registry) {  
 registry.addMapping("/home").allowedOrigins("http://localhost:4200");  
 registry.addMapping("/users").allowedOrigins("http://localhost:4200");  
 }  
 };  
 }  
}

**Alábbiakban található az AdminController.java fájl tartalma**

A AdminController osztály felelős az adminisztratív műveletek végrehajtásáért, például felhasználók és könyvek kezeléséért. Az osztály számos végpontot definiál a következő funkciókhoz:

* Felhasználók lekérdezése, létrehozása, frissítése és törlése.
* Könyvek lekérdezése, létrehozása, frissítése, törlése és lefoglalása.

Az osztályt annotációkkal láttuk el, például @RestController, @RequestMapping, @PostMapping, @PutMapping, @DeleteMapping stb., amelyek meghatározzák az egyes végpontok típusát és útvonalát.

A Spring Boot alkalmazás fő konfigurációs osztálya, a BackendApplication.java, szintén megtalálható, amely tartalmazza az alkalmazás belépési pontját. Ez az osztály inicializálja és konfigurálja az alkalmazást a Spring Boot alapértelmezett beállításaival, lehetővé téve az alkalmazás egyszerű futtatását és konfigurálását. Emellett a @Bean annotáció segítségével egy PasswordEncoder típusú bean kerül létrehozásra, amely egy BCryptPasswordEncoder példányt biztosít a jelszavak titkosításához, amelyet később a biztonsági konfigurációban lehet használni.

package com.geolidth.BackEnd.Controllers;  
  
import com.geolidth.BackEnd.exceptions.ForbiddenActionException;  
import com.geolidth.BackEnd.exceptions.NoSuchBookException;  
import com.geolidth.BackEnd.exceptions.NoSuchUserException;  
import com.geolidth.BackEnd.models.dao.Book;  
import com.geolidth.BackEnd.models.dao.BookUser;  
import com.geolidth.BackEnd.models.dto.NewBook;  
import com.geolidth.BackEnd.models.dto.NewUser;  
import com.geolidth.BackEnd.models.dto.UpdateBook;  
import com.geolidth.BackEnd.services.BookService;  
import com.geolidth.BackEnd.services.UserService;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.security.access.prepost.PreAuthorize;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
@CrossOrigin(origins = "http://localhost:4200")  
@RestController  
@RequestMapping("/api/admin")  
//@PreAuthorize("hasRole('ROLE\_ADMIN')")  
public class AdminController {  
  
 private final UserService userService;  
 private final BookService bookService;  
  
 public AdminController(UserService userService, BookService bookService) {  
 this.userService = userService;  
 this.bookService = bookService;  
 }  
 @GetMapping("/admin/all")  
 public ResponseEntity<List<BookUser>> getAllUsers() {  
 List<BookUser> users = userService.getAllUsers();  
 return ResponseEntity.*status*(HttpStatus.*OK*).body(users);  
 }  
 @GetMapping("/books/{id}")  
 public ResponseEntity<Book> getBookById(@PathVariable Integer id) {  
 try {  
 Book book = bookService.getById(id);  
 return ResponseEntity.*ok*(book);  
 } catch (NoSuchBookException e) {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
 @GetMapping("/admin/username/{username}")  
 public ResponseEntity<?> findUserByUsername(@PathVariable String username) {  
 try {  
 BookUser user = userService.findUserByUsername(username);  
 if (user != null) {  
 return ResponseEntity.*status*(HttpStatus.*OK*).body(user);  
 } else {  
 throw new NoSuchUserException("Nincs ilyen felhasználó: " + username);  
 }  
 } catch (NoSuchUserException e) {  
 return ResponseEntity.*status*(HttpStatus.*NOT\_FOUND*).body(e.getMessage());  
 }  
 }  
 @GetMapping("/admin/{userId}")  
 public ResponseEntity<BookUser> findUserById(@PathVariable Integer userId) {  
 BookUser user = userService.getById(userId);  
 return ResponseEntity.*status*(HttpStatus.*OK*).body(user);  
 }  
 @PostMapping("/users")  
 public ResponseEntity<?> createUser(@RequestBody NewUser newUser) {  
 BookUser savedUser = userService.save(new BookUser());  
 return ResponseEntity.*status*(HttpStatus.*CREATED*).body(savedUser);  
 }  
 @PostMapping("/books")  
 public ResponseEntity<?> createBook(@RequestBody NewBook newBook) {  
 Book savedBook = bookService.save(newBook);  
 return ResponseEntity.*status*(HttpStatus.*CREATED*).body(savedBook);  
 }  
 @PutMapping("/users/{userId}")  
 public ResponseEntity<?> updateUser(@PathVariable Integer userId, @RequestBody NewUser userDetails) {  
 BookUser updatedUser = userService.updateUser(userId, userDetails);  
 return ResponseEntity.*ok*(updatedUser);  
 }  
 @PutMapping("/books/{bookId}")  
 public ResponseEntity<Book> updateBook(@PathVariable Integer bookId, @RequestBody UpdateBook updateBook) {  
 try {  
 Book updatedBook = bookService.updateBook(bookId, updateBook);  
 return ResponseEntity.*ok*(updatedBook);  
 } catch (NoSuchBookException | ForbiddenActionException e) {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
 @DeleteMapping("/users/{userId}")  
 public ResponseEntity<?> deleteUser(@PathVariable Integer userId) {  
 userService.deleteUser(userId);  
 return ResponseEntity.*noContent*().build();  
 }  
 @DeleteMapping("/books/{bookId}")  
 public ResponseEntity<Void> deleteBook(@PathVariable Integer bookId) {  
 try {  
 bookService.deleteBook(bookId);  
 return ResponseEntity.*noContent*().build();  
 } catch (NoSuchBookException | ForbiddenActionException e) {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
 @PostMapping("/books/{bookId}/reserve")  
 public ResponseEntity<Void> reserveBook(@PathVariable Integer bookId) {  
 try {  
 Book book = bookService.getById(bookId);  
 book.setReserved(true);  
 UpdateBook updateBook = new UpdateBook();  
 updateBook.setReserved(true);  
 bookService.updateBook(bookId, updateBook);  
 bookService.deleteBook(bookId);  
 return ResponseEntity.*status*(HttpStatus.*CREATED*).build();  
 } catch (NoSuchBookException e) {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
}

Az alábbiakban található a ***BookController.java*** fájl tartalma, amely a könyvekkel kapcsolatos HTTP kérések kezéséért felelős. A fájlban a REST API funkciói valósulnak meg a következő módon:

1. *Könyvek lekérdezése: Az @GetMapping* annotáció segítségével a *getBooks* metódus felelős az összes könyv lekérdezéséért.
2. *Könyv részleteinek lekérdezése: Az @GetMapping("/{id}")* annotációval ellátott *getById* metódus felelős egy adott könyv részleteinek lekérdezéséért az azonosító alapján.
3. *Könyv mentése: Az @PostMapping* annotációval ellátott *save* metódus felelős az új könyv mentéséért. A kliens által küldött új könyv adatait egy *NewBook* objektumban kapja meg, és aztán a *bookService* segítségével elmenti az adatbázisba.
4. *Könyv frissítése: Az @PutMapping("/{bookId}")* annotációval ellátott *updateBook* metódus felelős egy meglévő könyv frissítéséért. A kliens által küldött frissített könyv adatokat egy *UpdateBook* objektumban kapja meg, és aztán a *bookService* segítségével frissíti az adatbázisban.
5. *Könyv törlése: Az @DeleteMapping("/{bookId}")* annotációval ellátott *deleteBook* metódus felelős egy könyv törléséért az azonosító alapján.

package com.geolidth.BackEnd.Controllers;  
  
import com.geolidth.BackEnd.exceptions.NoSuchBookException;  
import com.geolidth.BackEnd.models.UserRole;  
import com.geolidth.BackEnd.models.dao.BookUser;  
import com.geolidth.BackEnd.models.dto.UpdateBook;  
import com.geolidth.BackEnd.models.dao.Book;  
import com.geolidth.BackEnd.models.dto.NewBook;  
import com.geolidth.BackEnd.services.BookService;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.security.core.Authentication;  
import org.springframework.security.core.context.SecurityContextHolder;  
import org.springframework.web.bind.annotation.\*;  
  
  
import java.util.List;  
import java.util.Optional;  
@CrossOrigin(origins="http://localhost:4200")  
//@CrossOrigin(origins="\*",allowedHeaders = "\*")  
@RestController  
@RequestMapping("/books")  
public class BookController {  
 private final BookService bookService;  
 public BookController(BookService bookService) {  
 this.bookService = bookService;  
 }  
  
 @GetMapping  
 public ResponseEntity<List<Book>> getBooks(@RequestParam(required = false) String query) {  
 List<Book> books = bookService.searchBooks(query);  
 return ResponseEntity.*ok*(books);  
 }  
  
 @GetMapping("/{id}")  
 public ResponseEntity<Book> getById(@PathVariable int id) {  
 Book book = bookService.getById(id);  
 return ResponseEntity.*status*(HttpStatus.*OK*).body(book);  
 }  
 @PostMapping  
 public ResponseEntity<Book> save(@RequestBody NewBook newBook) {  
 Book savedBook = bookService.save(newBook);  
 return ResponseEntity.*status*(HttpStatus.*CREATED*).body(savedBook);  
 }  
 @PutMapping("/{bookId}")  
 public ResponseEntity<Book> updateBook(  
 @PathVariable int bookId,  
 @RequestBody UpdateBook updateBook) {  
 Book updatedBook = bookService.updateBook(bookId, updateBook);  
 return ResponseEntity.*status*(HttpStatus.*OK*).body(updatedBook);  
 }  
 @DeleteMapping("/{bookId}")  
 public ResponseEntity<?> deleteBook(@PathVariable Integer bookId) {  
 bookService.deleteBook(bookId);  
 return ResponseEntity.*status*(HttpStatus.*NO\_CONTENT*).build();  
 }  
 @PostMapping("/{bookId}/reserve")  
 public ResponseEntity<Void> reserveBook(@PathVariable Integer bookId, Authentication auth) {  
 if (auth != null && auth.isAuthenticated()) {  
 try {  
 bookService.reserveBook(bookId);  
 return ResponseEntity.*status*(HttpStatus.*CREATED*).build();  
 } catch (NoSuchBookException e) {  
 return ResponseEntity.*notFound*().build();  
 }  
 } else {  
 return ResponseEntity.*status*(HttpStatus.*UNAUTHORIZED*).build();  
 }  
 }  
}

**Az alábbaikban a GuestController.java fájl tartalma található:**

A GuestController osztály felelős az olyan kérések kiszolgálásáért, amelyeket vendég felhasználók intéznek az alkalmazáshoz. A vendég felhasználók nem rendelkeznek teljes hozzáféréssel az alkalmazáshoz, például nem tudnak könyveket lefoglalni vagy bizonyos adatokat módosítani.

Az osztály a következő funkcionalitásokat valósítja meg:

1. Könyvek lekérdezése: A getAllBooks() metódus az összes rendelkezésre álló könyvet adja vissza. Vendég felhasználók számára ez a legtöbb információt megjelenítő kérés.
2. Könyvek keresése: A searchBooks() metódus lehetővé teszi a könyvek keresését cím és/vagy szerző alapján. Ez a funkció lehetővé teszi a vendég felhasználók számára, hogy megfelelő könyveket találjanak az érdeklődési területeik vagy igényeik alapján.
3. Könyv lefoglalása: A reserveBook() metódus egy üzenetet ad vissza, ami tájékoztatja a felhasználót, hogy a könyv lefoglalásához be kell jelentkeznie. Ez azért fontos, mert a könyv lefoglalása olyan művelet, amely kizárólag bejelentkezett felhasználóknak engedélyezett.

Az osztályhoz annotációk vannak rendelve, például @RestController, @RequestMapping, @GetMapping, @PostMapping, amelyek meghatározzák az egyes végpontok típusát, útvonalát és működését. Emellett a @CrossOrigin annotáció engedélyezi a kéréseket az adott eredeti szerverről érkező kliensoldali JavaScript alkalmazásokból, ami lehetővé teszi a kereséseket egy adott eredeti szerverről, például http://localhost:4200-ról.

package com.geolidth.BackEnd.Controllers;  
  
import com.geolidth.BackEnd.exceptions.NoSuchBookException;  
import com.geolidth.BackEnd.models.dao.Book;  
import com.geolidth.BackEnd.services.BookService;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.Collections;  
import java.util.List;  
@CrossOrigin(origins = "http://localhost:4200")  
//@CrossOrigin(origins = "\*", allowedHeaders = "\*")  
@RestController  
@RequestMapping("/guest")  
public class GuestController {  
 private final BookService bookService;  
  
 public GuestController(BookService bookService) {  
 this.bookService = bookService;  
 }  
  
 @GetMapping("/books")  
 public ResponseEntity<List<Book>> getAllBooks() {  
 List<Book> books = bookService.getBooks();  
 return ResponseEntity.*ok*(books);  
 }  
  
 @GetMapping("/books/search")  
 public ResponseEntity<List<Book>> searchBooks(@RequestParam(required = false) String title,  
 @RequestParam(required = false) String author) {  
 try {  
 List<Book> books;  
 if (title != null && author != null) {  
 books = bookService.searchBooksByTitleAndAuthor(title, author);  
 } else if (title != null) {  
 books = bookService.searchBooksByTitle(title);  
 } else if (author != null) {  
 books = bookService.searchBooksByAuthor(author);  
 } else {  
 books = Collections.*emptyList*();  
 }  
 return ResponseEntity.*ok*(books);  
 } catch (NoSuchBookException e) {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
  
 @PostMapping("/reserve")  
 public ResponseEntity<String> reserveBook() {  
 String message = "Bejelentkezés szükséges a könyv lefoglalásához.";  
 message += "Ha még nem regisztrált, akkor regisztráljon a /api/v1/auth/register végponton keresztül";  
 message += "vagy jelentkezzen be a /api/v1/auth/login végponton.";  
 return new ResponseEntity<>(message, HttpStatus.*UNAUTHORIZED*);  
 }  
}

A **RoleController** osztály felelős a felhasználói jogok kezeléséért az alkalmazásban. A jogok lehetővé teszik az alkalmazásban történő különböző tevékenységek szabályozását és korlátozását a felhasználók számára.

Az osztály a következő funkciókat valósítja meg:

1. **Jogok lekérdezése**: A **getAllRoles()** metódus az összes rendelkezésre álló jogot adja vissza.
2. **Jog hozzáadása**: A **addRole()** metódus lehetővé teszi új jog hozzáadását az alkalmazáshoz.
3. **Jog frissítése**: A **updateRole()** metódus lehetővé teszi egy meglévő jog frissítését azonosítója alapján.
4. **Jog törlése**: A **deleteRole()** metódus lehetővé teszi egy meglévő jog törlését azonosítója alapján.
5. **Jog lekérdezése név alapján**: A **getRoleByName()** metódus lehetővé teszi egy meglévő jog lekérdezését a neve alapján.

Az osztályhoz annotációk vannak rendelve, például **@RestController**, **@RequestMapping**, **@GetMapping**, **@PostMapping**, **@PutMapping**, **@DeleteMapping**, amelyek meghatározzák az egyes végpontok típusát, útvonalát és működését. Emellett a **@CrossOrigin** annotáció engedélyezi a kéréseket az adott eredeti szerverről érkező kliensoldali JavaScript alkalmazásokból, ami lehetővé teszi a kereséseket egy adott eredeti szerverről, például **http://localhost:4200**-ról.

package com.geolidth.BackEnd.Controllers;  
  
import com.geolidth.BackEnd.models.UserRole;  
import com.geolidth.BackEnd.services.RoleService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
@CrossOrigin(origins = "http://localhost:4200")  
@RestController  
@RequestMapping("/roles")  
public class RoleController {  
  
 private final RoleService roleService;  
 @Autowired  
 public RoleController(RoleService roleService) {  
 this.roleService = roleService;  
 }  
  
 @GetMapping("/")  
 public ResponseEntity<List<UserRole>> getAllRoles() {  
 List<UserRole> roles = roleService.getAllRoles();  
 return new ResponseEntity<>(roles, HttpStatus.*OK*);  
 }  
  
 @PostMapping("/")  
 public ResponseEntity<Void> addRole(@RequestBody UserRole role) {  
 roleService.addRole(role);  
 return new ResponseEntity<>(HttpStatus.*CREATED*);  
 }  
  
 @PutMapping("/{id}")  
 public ResponseEntity<Void> updateRole(@PathVariable Integer id, @RequestBody UserRole role) {  
 role.setId(id);  
 roleService.updateRole(role);  
 return new ResponseEntity<>(HttpStatus.*NO\_CONTENT*);  
 }  
  
 @DeleteMapping("/{id}")  
 public ResponseEntity<Void> deleteRole(@PathVariable Integer id) {  
 roleService.deleteRoleById(id);  
 return new ResponseEntity<>(HttpStatus.*NO\_CONTENT*);  
 }  
  
 @GetMapping("/{roleName}")  
 public ResponseEntity<UserRole> getRoleByName(@PathVariable String roleName) {  
 UserRole role = roleService.getRoleByName(roleName);  
 return role != null ?  
 new ResponseEntity<>(role, HttpStatus.*OK*) :  
 new ResponseEntity<>(HttpStatus.*NOT\_FOUND*);  
 }  
}

A **UserController** osztály felelős a felhasználókkal kapcsolatos funkciók kezeléséért az alkalmazásban. Ezek a funkciók lehetővé teszik a felhasználók regisztrációját, profiljuk lekérdezését, adataik frissítését és törlését, valamint könyvek hozzáadását, törlését és lefoglalását. Az osztály a következő funkcionalitásokat valósítja meg:

1. **Felhasználók lekérdezése**: A **getUsers()** metódus az összes felhasználót adja vissza.
2. **Felhasználó profiljának lekérdezése**: A **getUserProfile()** metódus a bejelentkezett felhasználó profilját adja vissza.
3. **Felhasználó regisztrációja**: A **signUp()** metódus lehetővé teszi új felhasználó regisztrációját.
4. **Könyvek hozzáadása a felhasználóhoz**: A **addBook()** metódus lehetővé teszi könyvek hozzáadását a felhasználóhoz.
5. **Felhasználó adatok frissítése**: A **updateUser()** metódus lehetővé teszi a felhasználó adatainak frissítését.
6. **Könyvek frissítése és törlése**: A **updateBook()** és **deleteBook()** metódusok lehetővé teszik a felhasználók által tulajdonolt könyvek frissítését és törlését.
7. **Könyv lefoglalása**: A **reserveBook()** metódus lehetővé teszi egy könyv lefoglalását a felhasználó számára.

Az osztályhoz annotációk vannak rendelve, például **@RestController**, **@RequestMapping**, **@GetMapping**, **@PostMapping**, **@PutMapping**, **@DeleteMapping**, amelyek meghatározzák az egyes végpontok típusát, útvonalát és működését. Emellett a **@CrossOrigin** annotáció engedélyezi a kéréseket az adott eredeti szerverről érkező kliensoldali JavaScript alkalmazásokból, ami lehetővé teszi a kereséseket egy adott eredeti szerverről, például **http://localhost:4200**-ról. Az osztály továbbá tartalmaz néhány biztonsági intézkedést is, például ellenőrzi, hogy a felhasználó jogosult-e a könyv módosítására vagy törlésére.

package com.geolidth.BackEnd.Controllers;  
  
import com.geolidth.BackEnd.exceptions.ForbiddenActionException;  
import com.geolidth.BackEnd.exceptions.NoSuchBookException;  
import com.geolidth.BackEnd.models.UserRole;  
import com.geolidth.BackEnd.models.dao.Book;  
import com.geolidth.BackEnd.models.dao.BookUser;  
import com.geolidth.BackEnd.models.dto.NewUser;  
import com.geolidth.BackEnd.models.dto.UpdateBook;  
import com.geolidth.BackEnd.services.BookService;  
import com.geolidth.BackEnd.services.UserService;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.security.access.prepost.PreAuthorize;  
import org.springframework.security.core.Authentication;  
import org.springframework.security.core.context.SecurityContextHolder;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
import java.util.Objects;  
  
import static org.springframework.http.HttpStatus.*CREATED*;  
  
@CrossOrigin(origins = "http://localhost:4200",allowedHeaders = "\*")  
//@CrossOrigin(origins = "\*",allowedHeaders = "\*")  
@RestController  
//@PreAuthorize("hasAnyRole('ROLE\_USER', 'ROLE\_ADMIN')")  
//@RequestMapping("/api")  
@RequestMapping("/users")  
public class UserController {  
 private final UserService userService;  
 private final BookService bookService;  
  
 public UserController(UserService userService, BookService bookService) {  
 this.userService = userService;  
 this.bookService = bookService;  
 }  
  
 @GetMapping  
 public ResponseEntity<?> getUsers(){  
 List<BookUser> users=userService.getAllUsers();  
 return ResponseEntity.*ok*(users);  
 }  
  
  
 @GetMapping("/user/profile")  
 public ResponseEntity<?> getUserProfile() {  
 UserDetails userDetails = (UserDetails) SecurityContextHolder.*getContext*().getAuthentication().getPrincipal();  
 String username = userDetails.getUsername();  
 BookUser user = userService.findUserByUsername(username);  
 return ResponseEntity.*ok*(user);  
 }  
 @PostMapping  
 public ResponseEntity<BookUser> signUp(@RequestBody NewUser newUserRequest) {  
 BookUser savedUser = userService.save(new BookUser(newUserRequest));  
 if(savedUser.getRole().toString()!="ADMIN\_ROLE"){  
 savedUser.setRole(UserRole.Role.*USER\_ROLE*);  
 savedUser.setAdmin(false);  
 }  
 return ResponseEntity.*status*(*CREATED*).body(savedUser);  
 }  
  
 @PostMapping("/users/book")  
 public ResponseEntity<Book> addBook(@RequestBody Book book) {  
 return ResponseEntity.*status*(*CREATED*).body(userService.addBook(book));  
 }  
  
 @PutMapping("/{userId}")  
 public ResponseEntity<BookUser> updateUser(@PathVariable Integer userId,  
 @RequestBody NewUser userDetails) {  
 return ResponseEntity.*status*(HttpStatus.*OK*)  
 .body(userService.updateUser(userId, userDetails));  
 }  
  
 @PutMapping("/books/{bookId}")  
 public ResponseEntity<Book> updateBook(@PathVariable Integer bookId, @RequestBody UpdateBook updateBook) {  
 UserDetails currentUser = (UserDetails) SecurityContextHolder.*getContext*().getAuthentication().getPrincipal();  
 String currentUsername = currentUser.getUsername();  
  
 try {  
 Book book = bookService.getById(bookId);  
 if (!book.getOwner().getUsername().equals(currentUsername)) {  
 return ResponseEntity.*status*(HttpStatus.*FORBIDDEN*).build();  
 }  
 Book updatedBook = bookService.updateBook(bookId, updateBook);  
 return ResponseEntity.*ok*(updatedBook);  
 } catch (NoSuchBookException | ForbiddenActionException e) {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
  
 @DeleteMapping("/{userId}")  
 public ResponseEntity<Void> deleteUserData(@PathVariable Integer userId) {  
 userService.deleteUser(userId);  
 return ResponseEntity.*status*(HttpStatus.*OK*).build();  
 }  
 @DeleteMapping("/books/{bookId}")  
 public ResponseEntity<Void> deleteBook(@PathVariable Integer bookId) {  
 UserDetails currentUser = (UserDetails) SecurityContextHolder.*getContext*().getAuthentication().getPrincipal();  
 String currentUsername = currentUser.getUsername();  
  
 try {  
 Book book = bookService.getById(bookId);  
 if (!book.getOwner().getUsername().equals(currentUsername)) {  
 return ResponseEntity.*status*(HttpStatus.*FORBIDDEN*).build();  
 }  
 bookService.deleteBook(bookId);  
 return ResponseEntity.*noContent*().build();  
 } catch (NoSuchBookException e) {  
 return ResponseEntity.*notFound*().build();  
 } catch (ForbiddenActionException e) {  
 return ResponseEntity.*status*(HttpStatus.*FORBIDDEN*).build();  
 }  
 }  
 @PostMapping("/reserve/{bookId}")  
 public ResponseEntity<Void> reserveBook(@PathVariable Long bookId,  
 Authentication authentication) {  
 userService.reserveBook(bookId, ((BookUser) authentication.getPrincipal()).getId());  
 return ResponseEntity.*status*(*CREATED*).build();  
 }  
}

A Book.java fájl a com.geolidth.BackEnd.models.dao csomagban részt vesz az adatbázis-séma modellezésében, és az alkalmazásban a könyvek tárolásáért és kezeléséért felelős. Néhány fontos dolog, amit érdemes megjegyezni ebben az osztályban:

1. Az *@Entity* annotációval megjelölt osztályokat JPA entitásoknak tekintik, amelyek adatbázistáblákhoz rendelődnek.
2. A *@Table* annotáció segítségével meghatározható, hogy melyik adatbázistábla tartalmazza az entitást. Ebben az esetben a tábla neve "books".
3. A *@Id* annotációval megjelölt mező az adatbázisban a főkulcs (primary key).
4. A *@GeneratedValue* annotáció jelzi, hogy az *id* mező automatikusan generálódik, és az adatbázis generált értéket használja.
5. A *@Column* annotáció segítségével konfigurálhatók az oszlopok a táblában.
6. A *@ManyToOne* annotáció jelzi, hogy ez a könyv entitás egy másik entitáshoz, egy *BookUser* objektumhoz tartozik.
7. A *@JsonIdentityReference* és *@JsonProperty* annotációk segítenek az objektum hivatkozásainak kezelésében, amikor JSON-be kerül konvertálásra.

A BookUser.java osztály felelős az alkalmazásban a felhasználók tárolásáért és kezeléséért felelős, valamint a felhasználókhoz rendelt könyvek kezeléséért. Néhány annotáció, amely az előző fájlban nem volt használva:

1. A *@OneToMany* annotáció jelzi, hogy ez a felhasználó entitás több könyvet is tartalmazhat, és az adatbázisban ez a kapcsolat egy-minden kapcsolatot jelent.
2. Az osztály implementálja a Spring Security *UserDetails* interfészt, ami azt jelenti, hogy ez a felhasználó osztály a Spring Security keretrendszer által kezelt felhasználót reprezentálja.
3. Az *@JsonIgnore* annotáció jelzi, hogy a jelszó mezőt nem kell JSON formátumban reprezentálni, így az nem fog megjelenni az API válaszokban

A dto csomagban található fájlok DTO (Data Transfer Object) osztályokat definiálnak a *com.geolidth.BackEnd.models.dto* csomagban. Ezeknek a dto fájloknak a célja, hogy hibákat tartalmazó üzeneteket reprezentáljon a backend alkalmazásban. Néhány fontos dolog, amit meg kell jegyezni ebben az osztályban:

1. Az *@Data* annotáció a Lombok könyvtár egyik funkcióját jelenti, ami automatikusan generál gettereket, settereket, *toString(), equals() és hashCode()* metódusokat az osztály attribútumaihoz.
2. Az *ErrorMessage* osztály két attribútumot tartalmaz: *status és message.*
3. Az osztályban van egy konstruktor, ami egy *message* paramétert vár. Ez a konstruktor inicializálja az *status* attribútumot az "error" szöveggel, és a *message* attribútumot a paraméterben kapott üzenettel.

**2.9 Fejlesztési lehetőségek**

Megfelelő komponensek hozzáadásával a honlap alkalmassá tehető regisztrált felhasználólk közötti közvetlen kommunikációra is forum felület kialakításával. Ez azonban mindenféleképpen erőforrás igényes, folyamatos felügyeletet igényel. Szükséges szűrőprogramok alkalmazása a kommunkáció stílusának és céljának ellenőrzésére . Ebben az esetben a felhasználói feltételek olyan kialakítása válna szükségessé amellyel a legtöbb visszaélés megakadályozható.

Lehetséges az adminok részére blog kialakítása, ahol személyes élményeket oszthatnának meg a felhasználókkal elősígítve az oldal aktív használatát.

1. **Tesztelés**

**1.Teszt: Ellenőriztük az adatbázisban létrehozott táblák neveit és mezőit.**

* + Elvárt eredmény: Az adatbázisban léteznek a következő táblák:
    - users
    - roles
    - user\_roles

1. **Teszt: Ellenőriztük a users tábla mezőit és típusait.**
   * Elvárt eredmény: A users tábla mezői:
     + id: int(11), AUTO\_INCREMENT, Elsődleges kulcs
     + email: varchar(100)
     + is\_admin: bit(1)
     + logged\_in: bit(1)
     + password: varchar(100)
     + role: varchar(255)
     + username: varchar(50)
2. **Teszt: Ellenőriztük a roles tábla mezőit és típusait.**
   * Elvárt eredmény: A roles tábla mezői:
     + id: int(11), AUTO\_INCREMENT, Elsődleges kulcs
     + name: varchar(255)
3. Teszt: Ellenőriztük a user\_roles tábla mezőit és típusait.
   * Elvárt eredmény: A user\_roles tábla mezői:
     + id: int(11), AUTO\_INCREMENT, Elsődleges kulcs
     + role: varchar(255)
     + user\_id: int(11), Index

**2. Felhasználók és szerepek kezelése**

Cél: Bizonyosodjunk meg róla, hogy a felhasználók és szerepek kezelése megfelelően működik az alkalmazásban.

1. **Teszt: Ellenőriztük, hogy a felhasználókat és szerepeket sikeresen hozzá lehet adni az adatbázishoz.**
   * Elvárt eredmény: A felhasználók és szerepek hozzáadása során nincsenek hibaüzenetek, és az adatbázis megfelelően frissül.
2. **Teszt: Ellenőriztük, hogy a felhasználókhoz és szerepekhez tartozó adatok megfelelően megjelennek az alkalmazásban.**
   * Elvárt eredmény: A felhasználók és szerepek a megfelelő felületeken jelennek meg, és azok a megadott adatokkal rendelkeznek.
3. **Teszt: Ellenőriztük, hogy a felhasználók szerepeinek módosítása helyesen működik.**
   * Elvárt eredmény: A felhasználók szerepeinek módosítása során az adatbázis megfelelően frissül, és az alkalmazás megfelelő visszajelzést ad a sikeres műveletről.
4. **Jogosultságkezelés**

Cél: Bizonyosodjunk meg róla, hogy a felhasználók megfelelő hozzáféréssel rendelkeznek az alkalmazás különböző részeihez.

1. **Teszt: Ellenőriztük, hogy a rendszergazdák számára megfelelő funkciók érhetőek el az alkalmazásban.**
   * Elvárt eredmény: A rendszergazdák hozzáférést kapnak az összes adminisztrációs funkcióhoz, például felhasználók kezelése, szerepek módosítása stb.
2. **Teszt: Ellenőriztük, hogy a nem rendszergazdák számára korlátozott funkciók érhetőek csak el az alkalmazásban.**
   * Elvárt eredmény: A nem rendszergazdák csak a saját profiljuk kezeléséhez és más alapvető funkciókhoz férhetnek hozzá, de nem érhetnek el adminisztrációs funkciókat.

**A következő adatokkal végeztük el a tesztelést:**

books

1. id: 1, author: "John Doe", category: "Fiction", county: "Somerset", publisher: "Random House", quality: "Good", reserved: 0, title: "The Great Adventure", year: 2005, owner\_id: 101
2. id: 2, author: "Jane Smith", category: "Non-fiction", county: "Devon", publisher: "Penguin Books", quality: "Excellent", reserved: 1, title: "The Power of Now", year: 2010, owner\_id: 102
3. id: 3, author: "Mark Johnson", category: "Science Fiction", county: "London", publisher: "HarperCollins", quality: "Fair", reserved: 0, title: "Space Odyssey", year: 2018, owner\_id: 103
4. id: 4, author: "Emily Brown", category: "Romance", county: "Yorkshire", publisher: "Simon & Schuster", quality: "Very Good", reserved: 1, title: "Love in the Highlands", year: 2008, owner\_id: 104
5. id: 5, author: "Michael Williams", category: "Mystery", county: "Edinburgh", publisher: "Macmillan Publishers", quality: "Excellent", reserved: 0, title: "The Enigma of the Sphinx", year: 2015, owner\_id: 105

users

1. id: 1, email: "john.doe@example.com", is\_admin: 1, logged\_in: 1, password: "hashedpassword123", role: "admin", username: "johndoe"
2. id: 2, email: "jane.smith@example.com", is\_admin: 0, logged\_in: 0, password: "hashedpassword456", role: "user", username: "janesmith"
3. id: 3, email: "mark.johnson@example.com", is\_admin: 1, logged\_in: 1, password: "hashedpassword789", role: "admin", username: "markjohnson"
4. id: 4, email: "emily.brown@example.com", is\_admin: 0, logged\_in: 0, password: "hashedpasswordabc", role: "user", username: "emilybrown"
5. id: 5, email: "michael.williams@example.com", is\_admin: 0, logged\_in: 1, password: "hashedpassworddef", role: "user", username: "michaelwilliams"

role

1. id: 1, name: "John Doe"
2. id: 2, name: "Jane Smith"
3. id: 3, name: "Mark Johnson"
4. id: 4, name: "Emily Brown"
5. id: 5, name: "Michael Williams"

user\_roles

1. id: 1, name: "John Doe", role: "user"
2. id: 2, name: "Jane Smith", role: "admin"
3. id: 3, name: "Mark Johnson", role: "user"
4. id: 4, name: "Emily Brown", role: "user"
5. id: 5, name: "Michael Williams", role: "admin"

**4 Felhasználói dokumentáció**

**4.1. Böngészés a honlapon**

A weboldal használata lehetséges asztali gépről, laptopról, táblagépről, vagy mobiltelefonról. Webböngésző telepítésén kívül más program telepítésére és futtatására nem lesz szükséged, csak internetkapcsolatra.

Az alábbi böngészők bármelyikét elindíthatod:

|  |  |  |  |
| --- | --- | --- | --- |
| A képen kör, Színesség, Grafika, Grafikus tervezés látható  Automatikusan generált leírás |  | A képen rajzfilm látható  Automatikusan generált leírás | A képen kör, Színesség, gömb, bolygó látható  Automatikusan generált leírás |

Dupla kattintással tudod megnyitni a böngészőt és a következő url-t kell beírnod a kereső mezőbe:

A Bookhub weboldala egy különleges lehetőséget biztosít minden könyvszerető embernek, hogy a könyveid ne csak porosodjanak a polcodon, hanem új otthont és új gazdát találjanak. Fedezd fel a könyvcsere kalandját, ahol nem csak olvasol, hanem megiszthatod és cserélheted is a kedvenc történeteidet. csak jelentkezz be, és kezdődjön a kaland!

A képen szimbólum, Grafika, képernyőkép, Téglalap látható

Automatikusan generált leírás Erre az ikonra kattintva minden oldalról a főoldalra jutsz vissza. A főoldalon az oldal tetején megjelenő menüben tudsz választani a honlap szolgáltatásai között.

A könyvek a főoldalon töltődnek be, itt megtekintheted a választékot bejelentkezés nélkül is. Válassz egy könyvet, jelezd a tulajdonosnak, és beszéljétek meg hogyan tudod átvenni. Töltsd fel régi könyveidet, és várd, hogy jelentkezzen érte az új tulajdonos!

**4.2 Regisztráció a honlapon**

Regisztrációhoz a Sign up menüpontban tudod megadni az adataidat. Az adatok valódisága a te felelősséged.

A képen képernyőkép látható

Automatikusan generált leírás

Adatvédelem: az ön adatait(nevét, címét, elérhetőségét) csak akkor ismerheti meg más felhasználó, amikor jelzi önnek, hogy valamelyik könyvét szeretné átvenni, és felveszik a kapcsolatot egymással. Kérjük járjon el figyelmesen, hogy kivel veszi fel a kapcsolatot. A felhasználó felelőssége, hogy valós adatokat adjon meg, azt a weboldal adminjainak nem áll módjában ellenőrizni. Visszaélés esetén kérem jelezze az adminoknak, hogy meg tudjuk tenni a további intézkedéseket.

**4.3 Elérhető funkciók bejelentkezés után**

*\*saját adattaid megtekintése, módosítása*

*\*saját feltöltött könyveid megtekintése, módosítása, új könyvek felvitele*

*\*mások által feltöltött könyvek foglallása*

*\*zsákbamackó választása*

*\*honlapon böngészés*

*\*kapcsolatfelvétel adminokkal*

*\*hiba jelentése*

Ha nem tudsz választani, kérd meg a zsákbamackót, hogy segítsen!

A képen szöveg, képernyőkép, szoftver, Webhely látható

Automatikusan generált leírás

Ha kiváncsi vagy az alapítókra válaszd az about menüpontot.

A kapcsolat menüpontban nyugodtan üzenhetsz az adminisztrátoroknak bármilyen problémával vagy kéréssel. Ne habozz igénybevenni ezt a lehetőséget, ha valamit nem találsz, vagy Vannak ötleteid, hogyn fejleszthetnénk tovább a honlapot. Tudod jelezni nekünk továbbá a tömeges cserére vonatkozó kérésedet is, így nem kell minden egyes könyvet egyesével felrögzítened.

A könyvek cseréjének gyakorlati megvalósítását segítendő a könyveknél jelezve van a megye, ahol átvehető személyesen .

Minden olyan könyvnél tudod az átvételi szándékot, amelyik a honlapon látható. A már lefoglalt könyvek eltűnnek a kínálatból, így nem tudsz olyan könyvet véletlenül lefoglalni, amit más már lefoglalt magának.

**4.4 Hibajelzések, elérhetőségek**

Hibás adatok a regisztrációnál: Kérlek ellenőrizd, hogy a név mezőben nem szerepel-e téves karakter, a cím mezőben nem szerepel-e hibás karakter, illetve minden mezőt kitöltöttél-e a regisztrációs felületen?

Hibás adatok a könyvnél: Ellenőrizd újra, hogy a könyv címe nem túl hosszú-e? Ha nem fér ki a rendelkezésre álló mezőben, használj rövidítést.

Amennyiben a honlap hibásan jelenik meg, ellenőrizd, hogy a böngésződ frissítve van, és valóban a legújabb verziót használod.

Az adminok elérhetőségét az about menüpont alatt találod, a honlapon keresztül tudsz nekünk üzenetet küldeni. Ne habozz jelezni bármilyen hibát, amit találsz!

**4.5 Elérhető funkciók adminként**

*\*saját adataidat módosíthatod*

*\*felhasználók adatait módosíthatod, törölheted*

*\*módosíthatod a honlap funkcióit, funkciók elérhetőégeit*

*\*válaszolhatsz az adminoknak érkezett üzenetekre*

*+\*üzeneteket küldhetsz a többi adminnak*

*+\*üzeneteket küldhetsz a többi felhasználónak*

*\*könyveket foglalhatsz*

*\*könyveket ajánlhatsz fel*