

**SZEGEDI TUDOMÁNYEGYETEM
INFORMATIKAI INTÉZET**

Szakdolgozat

Hencsár Dorottya

2022

SZEGEDI TUDOMÁNYEGYETEM
INFORMATIKAI INTÉZET

Könyvtár adminisztrációs feladatainak kezelése webalkalmazásban

(Library administration database in a web application)

Szakdolgozat

Készítette:
Hencsár Dorottya
Programtervező informatikus
alapszakos hallgató

Témavezető:
Dr. Bilicki Vilmos
Egyetemi adjunktus

Szeged
2022

Feladatkírás

A szakdolgozatom során egy webalkalmazást készítettem, amely egy könyvtár adminisztrációs feladatait hivatott ellátni. Szükség lesz felhasználók regisztrálására, azok kezelésére, hogy követni tudjuk, ki vette ki a könyveket. Lehetőség lesz a könyvek felvitelére, kilistázására és hozzárendelése a felhasználókhöz. Ehhez kliens oldalon JavaScriptes ReactJs keretrendszert használtam, illetve a backendet Spring Boot-ban valósítottam meg. Ezek egymás között Swagger-rel kommunikálnak. PostgreSQL adatbázis biztosítja az adatok tárolását, a futtatási környezetet pedig Docker szolgálja ki.

Tartalmi összefoglaló

Téma megnevezése: A szakdolgozat céljából kitűzött témában egy webalkalmazást készítettem ReactJs keretrendszerben, amely egy könyvtár adminisztrációs feladatait kívánja könnyebbé tenni a felhasználók részére.

Feladat megfogalmazása: A webalkalmazás lehetővé teszi, hogy könyveket tároljunk, és azokat a felhasználókhoz rendelhetjük. Látni fogjuk majd, hogy ki melyik könyveket kölcsönözte ki, ez által azt is látjuk, hogy ki mennyire aktív felhasználó, tehát mennyi könyvet vesz ki. Könyveket is látjuk majd, hogy kik vették már ki, melyik könyv a legnépszerűbb.

Megoldási mód: Az alkalmazást egy weblap formájában valósítottam meg. Ehhez a ReactJs keretrendszert választottam, valamint a Java Spring Boot-ot a szerveroldalon. A projekt elkészítése során igyekeztem arra figyelni, hogy minél felhasználóbarátabb legyen az alkalmazás. Mivel ezt az alkalmazást egy civil szervezet részére készítettem, melynek én is a vezetőségi tagja vagyok, ezért igyekeztem arra figyelni, hogy mire van szüksége a szervezetnek. Tudtam, hogy mire akarjuk majd a későbbiekben használni, és hogy ehhez mire lesz szükség. Igyekeztem minél jobban kielégíteni az igényeket.

Alkalmazott eszközök, módszerek: A webalkalmazás elkészítéséhez ReactJs keretrendszert használtam. A backend-et pedig Java Spring Boot-ban állítottam össze. Használtam továbbá például a Node.js-t is, valamint a Swagger OpenAPI-ját is hasznosítottam a dolgozat során. Fejlesztői környezetet tekintve az IntelliJ-t és a Visual Studio Code-ot használtam. Verziókövetéshez pedig a GitHub-ot.

Elért eredmény: Úgy gondolom, hogy sikerült egy olyan webalkalmazást elkészíteni, amely kielégíti azokat a szükségleteket, melyeket a civil szervezetem megkövetel. Figyeltem arra is, hogy megismerjem azokat a technikákat, amelyeket majd a későbbiekben akár a munkahelyemen is kamatoztathatok.

Kulcsszavak: ReactJs, Spring Boot, Swagger, webalkalmazás

Tartalomjegyzék

Feladatkiírás	3
Tartalmi összefoglaló	4
Tartalomjegyzék	5
Motiváció	7
1. Terület áttekintése	8
1.1. Navigációs sáv	8
1.2. Mobil telefonra való méretbeli optimalizáció	8
1.3. Jogosultságok kiosztása	9
1.4. Domain név	10
1.5. Cég neve és logója	11
1.6. Kapcsolattartási lehetőségek leírása	11
2. A fejlesztés megkezdése	13
2.1. Szükséges programok telepítése	13
2.1.1. Java és Maven	13
2.1.2. ReactJs és Node.js	13
2.1.3. OpenAPI és Swagger	14
2.1.4. Verziókövetés	14
2.3. Kód tisztán tartása	15
3. Fejlesztés menete	16
3.1. Adatmodell	16
3.2. Use-Case Diagram	19
3.2.1 Regisztráció	20
3.2.2. Be- és kijelentkezés	22

3.2.3. Könyvek kilistázása és szerkesztése	23
3.2.4. Saját adatlap és annak szerkesztése	25
3.2.5. Könyvek hozzáadása	26
3.2.6. Könyv atlap és kölcsönzés hozzáadása	26
3.2.7. Kölcsönzések kilistázása	28
3.2.8. Felhasználók kilistázása	28
3.3. Szekvencia diagram	28
3.3.1. Regisztráció	29
3.3.2. Be- és kijelentkezés	30
3.3.3. Könyvek kilistázása, adatlapja, szerkesztése és hozzáadása	30
3.3.4. Saját adatlap szerkesztése és jelszócsere	33
3.3.5. Kölcsönzések listázása és hozzáadása	34
4. Architektúra	36
4.1. Frontend	36
4.1.1. A Login hook-jának bemutatása	36
4.1.2. Jelszócsere hook-jának bemutatása	38
4.2. Backend	39
4.2.1. A könyv szerkesztésének bemutatása a backend-ben	40
4.2.2. Nem triviális query-k a BookRepository-ból	41
5. Összefoglaló, továbbfejlesztési lehetőségek	42
Felhasznált irodalom, források	43
Nyilatkozat	44
Köszönetnyilvánítás	45
Elektronikus mellékletek	47

Motiváció

Pár éve az ifjúsági civil szervezetünk kapott egy kisebb könyvadományt. Az első pillanattól kezdve az volt az ötletünk, hogy jó lenne ezeket a könyveket “jóra fordítani”, hiszen ezek a könyvek nem a mi tulajdonunk, hanem a falunkban az összes fiatalé. Innen jött az ötlet, hogy akkor működjünk úgy, mint egy könyvtár. Mivel a vezetőségi tagok közül én voltam az, akit érdekelt a projekt, magamra vállaltam, hogy írok egy webalkalmazást, ami a kölcsönzéseket kezeli. Teljesen szabad kezet kaptam minden döntést illetően. Ez egyszerre volt izgalmas és félelmetes is.

Sokat gondolkodtam, hogy milyen technológiákat használva tudnék a legtöbbet tanulni. Úgy döntöttem, hogy Spring Boot-ban fogom megírni a backendet. Java az egyik legelterjedtebb programnyelv, mivel igen rugalmas, ezért mindig is szerettem volna vele komolyabban is foglalkozni, most megvolt rá az alkalom.

Miután megjelent a Web 2.0, főleg különböző könyvtárakat és keretrendszereket használunk az egységes és könnyebb fejlesztés miatt. Mivel Springben íródik a backend, lényegében szabadon tudtam válogatni a keretrendszerek közül. Megint az egyik legelterjedtebbre, a ReactJs-re esett a választás. Többek között ez a Meta (Facebook) keretrendszere is. Ideális egyéni és csapatmunkára is. A React sajátossága még, minden egyetlen html oldalon történik, a komponensek kicserélésével. Egyetlen egyszer kéri a webböngésző a weblapot a szervertől, és utána bármi interakció során, csak a különböző react komponensen cserélődnek ki a képernyőn.

1. Terület áttekintése

Mivel úgy döntöttem, hogy az elképzelésemet webalkalmazásban valósítom meg, azzal kezdtem, hogy utánanéztem, hogy mik azok az elengedhetetlen fontosságú feature-ök, amik könnyen használhatóvá teszik az alkalmazásomat. Illetve, hogy ezeket hogyan tudom majd megvalósítani, milyen nyelveket érdemes használni, és miért.

1.1. Navigációs sáv

Az egyik ilyen feature a könnyen használható navigációs sáv. Felhasználók szempontjából nagyon fontos, hogy gyorsan odataláljanak a keresett felületre. Ne kelljen több kattintással odatalálni, ha ezt meg lehet valósítani egy kattintással is. Érdemes ezt a sávot jól látható helyre helyezni a sávot, hogy a felhasználó könnyen észre tudja venni. Érdemes minden oldalon használni a sávot, lehetőleg ugyanazt. Ez lehet akár az oldal felső- vagy alsó részére is, de jobb vagy bal oldalra is helyezhetjük. Esetleg valahogy megkülönböztethetjük azt a menüpontot, amin épp vagyunk, így nem zavarjuk össze a felhasználót. Ezen kívül az is nagyon fontos, hogy értelmesek legyenek a menüpontok. A legjobb, ha beszédes a név, viszont tömör. Például hiába a jól csengő elnevezés, ha nem egyértelmű, hogy mit takar. A navigációs sávnak az a feladata, hogy segítsen, és nem azért, hogy újabb kérdéseket tegyen fel. Az sem előnyös, ha túl sok menüpont van. Sokáig tartana kikeresni a kívánt mezőt, ami nem túl kényelmes. Érdemes ezért rendszerezni a témákat. Erre alkalmas lehet akár egy dropdown menü is. Tehetünk bele akár különböző szűrőket, ezzel is megkönnyítve a felhasználói élményt.

1.2. Mobil telefonra való méretbeli optimalizáció

A mai világban elengedhetetlen feature továbbá az is, hogy mobiltelefonra optimalizált webalkalmazást adjunk ki a kezeink közül. 2020-ra már megközelítőleg 6,4 milliárd embernek van a kezében okostelefon. Ez a szám 2027-re elérheti akár a 7,5 milliárd főt is. Emellett az is

nagyon fontos, hogy ma már sokkal többször csatlakozunk a világhálózhoz mobiltelefonnal, mint számítógéppel. Így ahhoz, hogy fenntartsuk a felhasználók érdeklődését, érdemes számolni a különböző mobiltelefonok kijelzőméreteivel is. Míg egy hatalmas gomb jól mutathat a számítógépek nagyobb kijelzőjén, addig a kisebb telefonok esetében, akár ki is akarhatja a további tartalmakat. De ez fordított esetben is igaz. Ha egy túl kicsi gomb a számítógép kijelzőjén elveszik, és sok időt elvesztegethetünk a felkutatásukkal. Emelett érdemes a fent említett navigációs sávot elrejtetni a telefonokon, hogy ezekkel ne zúfoljuk túl a kijelzőket, viszont ha szükség van, rá, akkor könnyen elérhető legyen. A méretkülönbséget érdemes a felugró reklámoknál is figyelembe venni, ügyelve arra, hogy ne takarja ki az oldal fő mondanivalóját.

1.3. Jogosultságok kiosztása

Figyelembe kell venni, hogy kinek a számára készítjük a webalkalmazást. Jó esetben sok felhasználó fogja használni az alkalmazást, ebből kifolyólag nem lenne jó, ha mindenkinek mindent megengednénk. Hiszen ha így lenne, akkor már elég hamar azt káoszba fulladna az egész. Ennek a problémának elkerülése érdekében érdemes több jogosultságot is felvenni. Ez azt jelenti, hogy egyes felhasználóknak több, míg másoknak kevesebb lehetőség nyílik meg az alkalmazás használata közben. Érdemes lehet egy adminisztrátor jogosultságot is bevezetni. Ennek az lenne a lényege, hogy az adminisztrátor felhasználó mindent lát, mindenhez van hozzáférése. Így ha egy sima felhasználó elront valamit, például valamit félrekattintva elrontja a nevét, amire már nincs lehetősége javítani, az adminisztrátor jogosultságokkal felhatalmazott felhasználó akár könnyedén ki is tudja ezt javítani. Ilyenkor viszont vigyáznunk kell, hogy mégis kinek adunk ilyen jogosultságot. Hiszen nagy hatalma van az alkalmazás felett. Ezért jó, ha többféle jogosultságot hozunk be az alkalmazásba, felosztva a feladatokat. Így kivédve azt, hogy valaki mindenhez való hozzáférést kihasználva káoszt kreáljon az alkalmazásunk használata közben. Arra is kell figyelmet fordítani, hogy az alkalmazást többféle felhasználót akar fenntartani, az adminok mellett. Például különböző közösségi oldalakon jellemzően van egy moderátor nevezett jogorultság is. Az ő feladatuk főleg a szabályok betartatása. Például joguk lehet törölni a szabályba ütköző posztokat, illetve tiltani a szerzőjüket. Esetemben egy könyvtár

adminisztrációs alkalmazást hoztam létre, ahol vannak sima felhasználók, és könyvtárosok is az adminok mellett. A könyvtárosoknak is van azokhoz jogosultságuk, amikhez a sima felhasználóknak, ez mellett még van pár plusz lehetséges funkciójuk is, mint például, hogy új könyvet vegyenek fel, vagy hogy a már meglévő könyvet egy másik felhasználóhoz kössék, vagyis kölcsönzést hozzon létre a felhasználó és a könyv között. Az adminnak pedig még több jogosultsága lesz. Az admin a felhasználókat is tudja majd szerkeszteni, akár törölni is, míg a könyvtárosok ezt nem tehetik meg.

1.4. Domain név

A webalkalmazás URL-je nagyon fontos. Ez lesz az a cím, amik keresztül a felhasználók elérik a weboldalt. Jó ötlet lehet tehát az, hogy egyszerű, könnyen megjegyezhető nevet választani. Figyelve a felhasználókra, fontos lehet, hogy a megjegyezhetőség mellett könnyen gépelhető is legyen. Ez azt jelenti, hogy érdemes olyan nevet választani, ami rövid és nincsenek benne különböző írásjelek, mint például a kötőjel, stb. Sok esetben a brand nevét szökták megadni domain névnek, ami általában véve jó választásnak bizonyul. Ha a kívánt cím más foglalt, akkor sem szabad kétségbe esni. Ilyenkor kell elővenni a kreativitást, és előállni egy hasonló névvel. Vannak jól működő "töltelékszavak" ezekre az esetekre. Ilyen lehet, ha a brand elé írjuk a "the" szócskát, magyar nyelv esetében a szükséges határozott névelőt. De ha ez esetleg nem tetszene, akkor lehetőség van még arra is, hogy a brand után írjuk az "online" szavat. De ezek csak szokások, nyugodtan el is térhetünk tőlük, ha úgy érezzük jónak. Nem jó, ha egy olyan nevet akarunk választani az alkalmazásunknak, ami nagyon hasonló egy már létező oldal domain nevéhez. Ha a felhasználón sokszor véletlenül ehhez a hasonló domain nevű oldalhoz látogat, könnyen megunhatja ezt, és végül nem is szeretné majd meglátogatni a mi oldalunkat. Érdemes lehet a különböző számokat is elkerülni a domain névben. Kivételt képez az az eset, mikor a brand, akinek dolgozunk, tartalmaz számot.

1.5. Cég neve és logója

A cég nevének meg kell jelennie az oldalon. Lehetőleg jól látható helyen, a nyitó oldalon mindenképp. Fontos, hogy a felhasználóknak ne kelljen keresni azt, és ne kelljen görgetni, hogy megtudják, hogy kinek az oldalára is látogattak el. A cég neve mellett a logót is érdemes megjeleníteni. A logó egy kicsi, könnyen megjegyezhető kép, ami segít felépíteni egy kapcsolatot a felhasználók agyában a cég és a kép között. Lehet, hogy a felhasználók épp nem emlékeznek a cég nevére, de a képet megjegyzik. Ezért fontos ezt is jól látható helyre helyezni. A cég neve és logója mellett jó ötlet lehet, ha adunk egy kisebb leírást a cégről, illetve, hogy mire használható az adott alkalmazás, amit fejlesztünk. Így a felhasználó rögtön tudni fogja, hogy ez az az oldal, amit valójában keres, vagy nem. Ez főleg a még kisebb cégek esetében elengedhetetlen. Egy kis leírás elmondja, hogy mivel foglalkozik a cég, és a webalkalmazás minek a céljából jött létre.

1.6. Kapcsolattartási lehetőségek leírása

A kapcsolattartási lehetőségek alatt a cég telefonszámát, címét,, email címét, stb értjük. Így a felhasználó tudni fogja, hogy milyen elérhetőségeken keresztül keresheti fel a céget, ha esetleg valami problémája akad. Ezt sok esetben a lábjegyzetbe szokták helyezni. A lábjegyzetbe ezek mellett érdemes lehet odahelyezni, hogy mikor is van nyitva a cég, ez által azt is biztosítjuk a felhasználók számára, hogy tudják, hogy mikor hívhatják a céget, illetve mikor várhatnak választ az elküldött emailjeikre. Ha úgy érezzük, hogy van elég kihasználatlan hely a lábjegyzetben, akkor beszúrhatunk még egy biográfiát is a cégről. Ez mellett, ha azt érezzük, hogy nem elég csak a lábjegyzetbe feltüntetni ezeket az információkat, kiírhatjuk mindezt egy külön oldalon is. Ezt követően pedig létrehozhatunk egy külön mezőt erre a navigációs sávban. De a kettő működhet egyszerre is. Így lesz külön oldal a kapcsolatnak, és mellette minden más oldalon ott lesz lábjegyzetben is. De figyelniünk kell a részletekre is, értem ez alatt, hogy ügyeljünk arra,

hogy a kapcsolat oldalon ne legyenek a lábjegyzetben is ugyanazok az információk. Teljesen felesleges, és nem is feltétlenül esztétikus.

2. A fejlesztés megkezdése

2.1. Szükséges programok telepítése

2.1.1. Java és Maven

A fejlesztést azzal kezdtem, hogy utánanéztem milyen programokra lesz szükségem a fejlesztés alatt. Ezek között volt például az IntelliJ¹ is. Ezt a IDE-t használják legtöbben Java nyelvű fejlesztésekhez, ezért én is e mellett döntöttem. Természetesen Javat is le kellett töltenem. Maven project-et a <https://start.spring.io/> oldalról hoztam létre, majd letöltöttem. Itt be tudtam állítani, hogy Java nyelven szeretnék fejleszteni a továbbiakban, illetve, hogy mi legyen a project neve, stb.

2.1.2. ReactJs és Node.js

Az IntelliJ-en kívül tudtam, hogy szükségem lesz még egy másik IDE-re is, amiben a React alapú frontendet fogom megvalósítani. Az egyetemen töltött éveim alatt sok különböző IDE-t is kipróbáltam már, ezért tudtam, hogy mit válasszak. Így a Visual Studio Code²-ra esett a választásom.

Node.js³ egy olyan programcsomag, amely főleg JavaScript-es csomagokat foglal össze. Innen töltöttem le minden React-hoz szükséges csomagot is. A Node.js-re a Node Package Manager, azaz az npm miatt volt szükségem. Többek között a CSS file-okat is innen töltöttem le, ugyanis a PrimeReact⁴ és Bootstrap⁵ csomagokat használtam, hogy szebbé tegyem a webalkalmazást.

¹ <https://www.jetbrains.com/idea/>

² <https://code.visualstudio.com/>

³ <https://nodejs.org/en/>

⁴ <https://www.primefaces.org/primereact/>

⁵ <https://getbootstrap.com/>

2.1.3. OpenAPI és Swagger

Az OpenAPI és Swagger⁶ sok esetben ugyanazt jelenti, de hivatalosan mégsem. A Swagger egy szoftvercsomag, az OpenApi pedig egy szabványosított változat. Tehát nem túl szépen fogalmazva, az OpenAPI a Swagger-ből jön létre.

A Swagger Codegen-t az api végpontjainak összekötéséhez használtam. Ez legenerálta a backend kódot, hogy aztán a végpontokat összekössem. Valamint használtam a Swagger Editort-t és UI-t, hogy ellenőrizni tudjam, hogy minden funkciójól működik-e.

2.1.4. Verziókövetés

A verziókezelést sok esetben akkor használjuk, ha többen dolgozunk egy projekten. Viszont úgy gondoltam, hogy én is szeretném ezt használni. Jó lehet a későbbiekben arra, hogy lássam, hogy hogyan haladtam egyes elemekkel. Könnyebben áttekinthető a fejlesztés menete.

Verziókövetéshez a git⁷-et használtam. Távoli repository-nak pedig a GitHub⁸-ot választottam. Mivel Windows-on fejleszték, ezért a git-et is külön le kellett töltenem.

2.2. Tanulás

Ahhoz, hogy el tudjak indulni, fel kellett szednem némi tudást is. Már az egyetem alatt is tanultam Java nyelvet, így elindulnom nem volt nehéz. De ez mellett úgy éreztem, hogy szeretném kicsit bővíteni és gyakorolni a meglévő tudásomat. Ezért egy Udemy kurzust választottam, hogy kicsit felelevenítsem a már tanultakat (Java Programming Masterclass updated to Java 17⁹). Mivel Spring Boot-ot szerettem volna használni, amit még sosem használtam előtte, ezért úgy éreztem, hogy mindenképp szükségem lesz tanulásra. Ezért több YouTube tutoriált is végigkövettem, hogy több oldalról, jobban is megismerjem a Springet, a

⁶ <https://swagger.io/>

⁷ <https://git-scm.com/>

⁸ <https://github.com/>

⁹ <https://www.udemy.com/course/java-the-complete-java-developer-course/>

különböző annotációk használatának fontosságát (Spring Boot Tutorial | Full Course [2022] [NEW]¹⁰).

A Java nyelv mellett még a JavaScript tudásom felfrissítésére is szükség volt. Mivel az egyetemi éveim alatt sokat foglalkoztam már vele, valamint mivel kedveltem ezt a nyelvet, többször is gyakoroltam otthon, ezért egy kisebb YouTube kurzust néztem meg ezzel kapcsolatban. Viszont ez mellett React-tal még sosem találkoztam. Nem ismertem még az alap koncepciókat, ezért ebben az esetben is egy Udemy kurzust választottam a tanuláshoz (React - The Complete Guide (incl Hooks, React Router, Redux)¹¹). Ez a kurzus megismertette velem az alap működést, hogy hogyan épül fel egy project React-ban.

2.3. Kód tisztán tartása

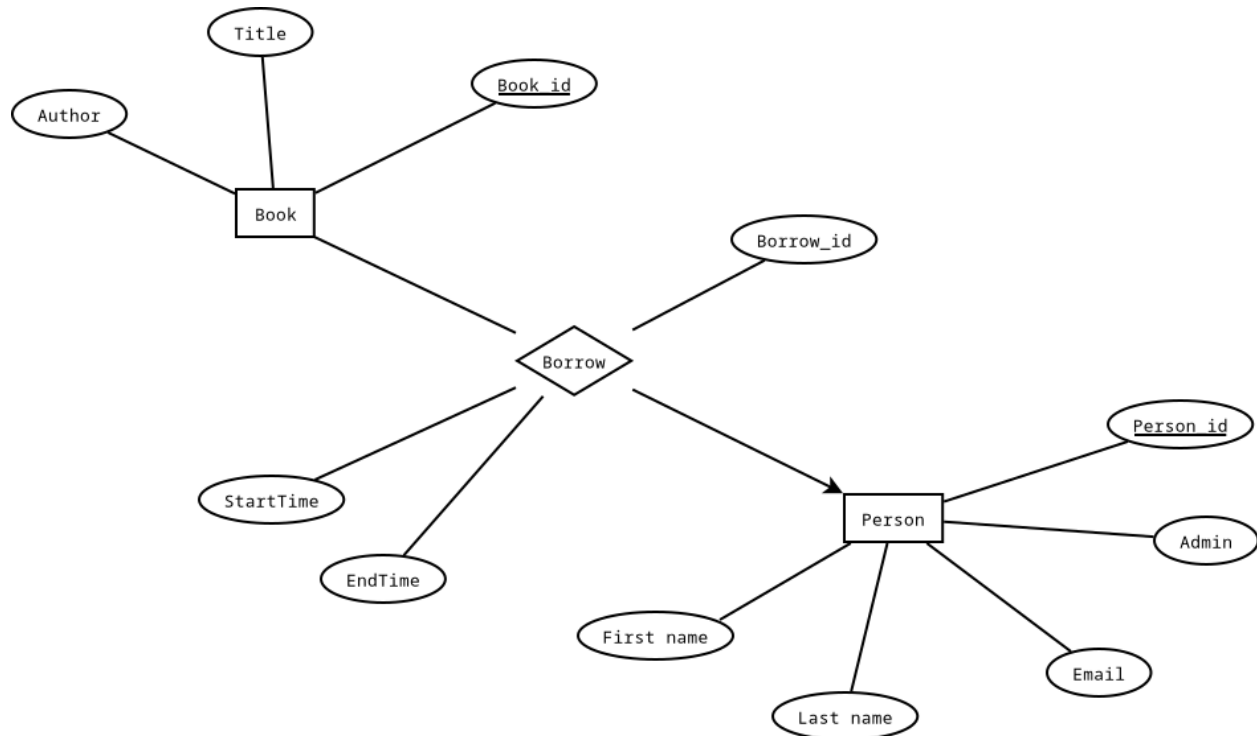
Szerettem volna, ha a fejlesztés viszonylag szépen, áttekinthetően halad, így én sem veszek el a részletekben. Ezért fontosnak tartottam azt, hogy legyen egy rugalmas, viszont hatékony időbeosztásom. Ebbe beleérttem azt is, hogy időnként átnéztem a már kész kódot, hogy az jó-e, illetve lehet-e azt egyszerűsíteni, szebben megfogalmazni, vagy könnyebben olvashatóvá tenni, figyelve arra, hogy a nyelv használati szokásokat betartsam. Nem szerettem volna magam határidőkbe szorítani, inkább csak arra szerettem volna törekedni, hogy időnként időt fordítsak arra is, hogy átnézzem a kódot. Így megelőzve azt is, hogy egy most még jól, de később problémát okozó kódrészlet még idejében ki legyen javítva, ezzel sok fejfájást megspórolva a jövőben.

¹⁰ <https://www.youtube.com/watch?v=9SGDpanrc8U>

¹¹ <https://www.udemy.com/course/react-the-complete-guide-incl-redux/>

3. Fejlesztés menete

3.1. Adatmodell



3.1.1. Ábra - adatmodell

A tervezést egy Adatmodell séma elkészítésével kezdtem. Ebben a sémában leírtam, hogy milyen entitásokkal fogok dolgozni, és hogy köztük milyen kapcsolat fog felépülni.

Két entitást vettem fel: Book (könyv) és Person (felhasználó). Valamint a köztük lévő Borrow (kölcsönzés) kapcsolatot.

A Book (Könyv) entitásnak csak három attribútumot vettem fel. Ezek pedig a könyv címe és annak szerzője, valamint a könyv generált egyedi azonosítója. Ennek az volt az oka, hogy szerettem volna kezdetben egyszerű adatmodellt készíteni, ami majd a későbbiekben igény szerint is módosítható. Ha szükséges, akkor például fel lehet venni a kiadás évét, a kiadót, oldalszámot, stb. De mivel tudom, hogy a szervezet, ami ezt a webalkalmazást használni fogja a jövőben, az egyszerűsége törekedik, ezért nem szerettem volna túlszűfolni esetlegesen felesleges adatokkal, inkább akkor bővíteni, ha szükséges. Egyedi azonosítóra mindenképp

szükség lesz, hiszen egy adott könyvből akár több példány is előfordulhat a raktáron, ezért fontos, hogy valahogyan meg tudjuk különböztetni azt, hogy konkrétan melyik könyvet kölcsönözte ki egy adott felhasználó. Így könnyebben elkerülhetővé téve az esetleges kavarodást a kölcsönzéseket illetően.

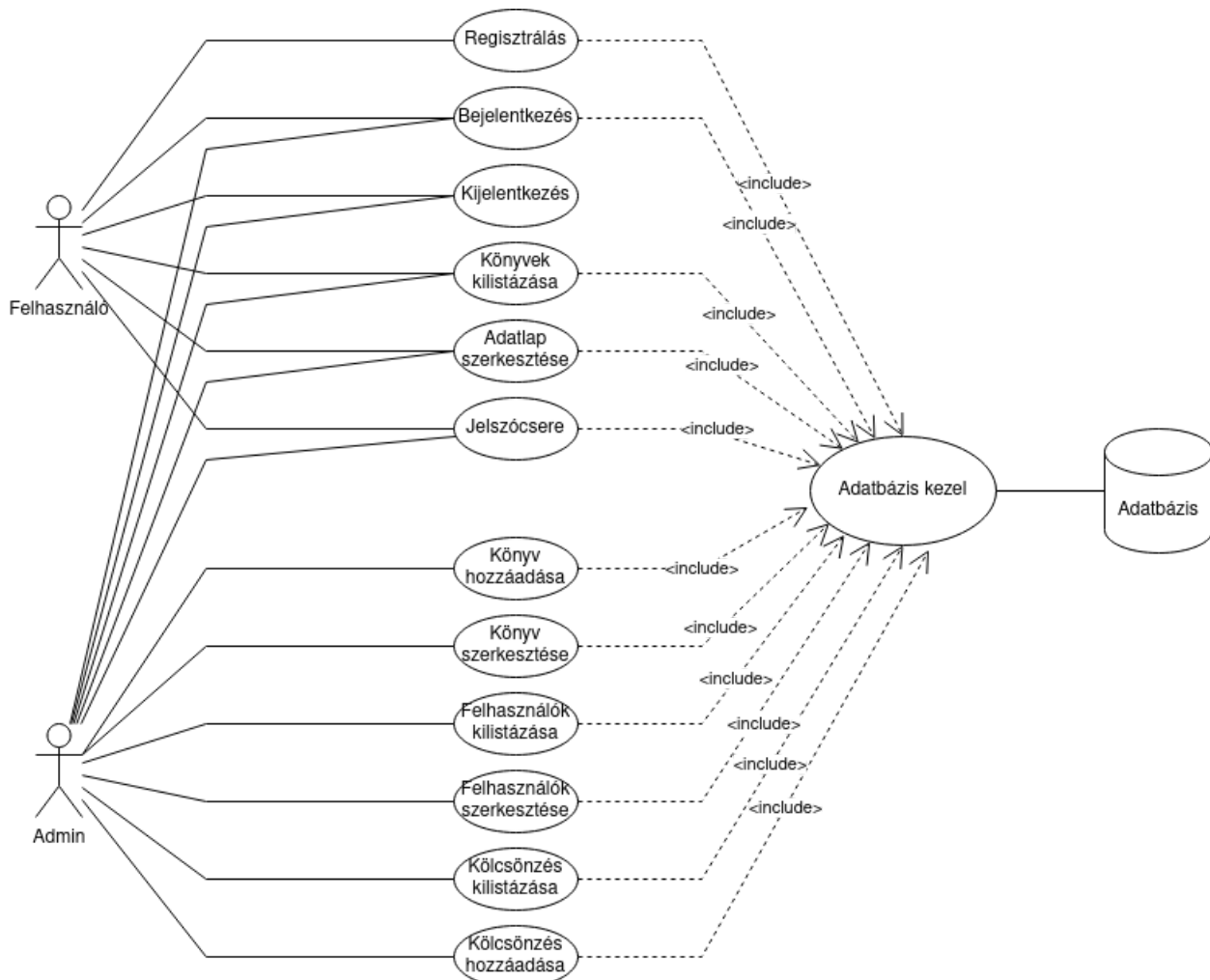
A Person (Személy/Felhasználó) entitásnak már kicsivel több attribútuma van, viszont itt is igyekeztem az egyszerűsége törekedni. Keresztnév, vezetéknév, email cím, jelszó és admin attribútumok, valamint itt is helyet foglal a generált egyedi azonosító. Itt azért van szükség a kereszt- illetve utónév különvétele, mert szerettem volna, ha a bejelentkezéskor külön, névre szólóan tudja majd az oldal köszönteni a bejelentkezett felhasználót. De nem szabad azt sem kihagyni, hogy így sokkal esztétikusabb maga a beregisztrálásra használt felület is. Az email cím szükséges, és egyedi (tehát nem lehet majd egy email címmel többször is beregisztrálni), mivel mindenképp kell majd egy elérhetőség is, hogy a szervezet, ami használni fogja a webalkalmazást, tudjon egyénileg is kommunikálni a felhasználókkal, ha szükséges. Ez a későbbiekben majd bővíthető lehet például egy telefonszámmal is, de kezdetben úgy gondoltam, hogy az email cím is elegendő lesz. Természetesen a bejelentkezéshez szükséges egy jelszó is. Regisztrációnál majd mindenképp kérni fogjuk majd a felhasználót, hogy ismétlje meg a beírt jelszavat, és ha ez a két jelszó megegyezik, csak abban az esetben fogjuk lementeni azt. Ezt a jelszavat majd titkosítani fogjuk a későbbiekben. Az admin attribútum egy igaz vagy hamis értéket fog felvenni. Ez azt fogja jelenteni, hogy az adott felhasználónak milyen jogosultságai vannak. Tehát hogy adminisztrátor jogokkal van-e ellátva, vagy nem. Mindezek mellett még felvettem egy plusz attribútumot is. Egy egyedi azonosítót is. Az email cím is egyedi lesz, és azt is mindenképp meg kell majd adni, tehát az sem hiányozhat semmiképp, viszont úgy láttam jónak, ha mindenképp felveszem ezt az egyedi azonosító attribútumot is. Sokkal áttekinthetőbbé és olvashatóvá teszi ezáltal a kódot. Ez mellett még sokkal bővíhetőbbé teszi a webalkalmazást. Ha a későbbiekben például szeretnénk, ha egy fiókhoz több email cím is tartozhasson, akkor nem kell majd az egész projektet átírni, hiszen a fióknak saját azonosítója lesz.

A Borrow (Kölcsönzés) kapcsolathoz is fel kellett vennem attribútumokat. Ezek az attribútumok a mettől és a meddig. Ez azt jelenti, hogy a könyv mikor lett kikölcsönözve, illetve, hogy mikor lett visszahozva. Ez által majd többféle szűrőt is fel tudunk használni a könyvek kilistázásához. Például meg tudjuk majd nézni, hogy melyik könyvet milyen sokszor

kölcsönöztek ki, tehát hogy melyek a legnépszerűbb könyvek. Ez mellett rá tudunk lesni arra is, hogy melyek azok a könyvek amelyek szabadok, így tudunk majd könyvet ajánlani a kölcsönözni vágyó felhasználóknak. De fontos azt is tudnunk, hogy melyek azok a könyvek, amelyek éppen nem elérhetőek, vagyis már ki vannak adva. Ezzel javítani tudom a felhasználói élményt mint a sima felhasználóknak, mint a könyvtárosoknak egyaránt. Ez a két attribútum mellé még felvettem egy harmadikat is. A már fent említett entitásokhoz hasonlóan, itt is felvettem egy egyedi azonosítót. Erre azért volt szükség, hogy nyomon tudjuk követni a kölcsönzéseket is. Ki tudjuk listázni az összes kölcsönzést, vagy egy könyvhöz tartozó kölcsönzéseket, vagy hogy melyik felhasználó mennyi könyvet kölcsönzött már ki.

Értelemszerűen egy könyvet egyszerre csak egy felhasználó tud majd kölcsönözni, és egy felhasználó több könyvet is tud majd kölcsönözni egyszerre.

3.2. Use-Case Diagram

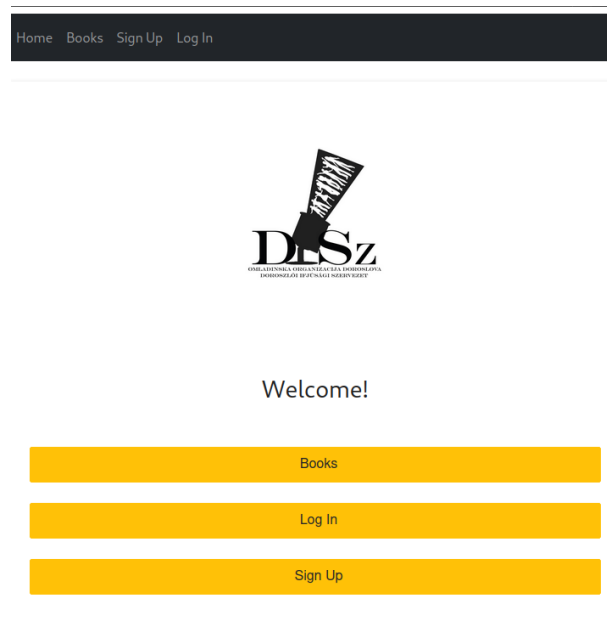


3.2.1. Ábra - Use-case diagram

A felhasználó a nyitó oldalon három mezőt is fog majd látni. Ugyan ezeket a mezőket fogja látni a navigációs sávban is. Ezek pedig: regisztráció, bejelentkezés és a könyvek kilistázása. Valamint ott még lesz egy olyan mező is, amivel újra a nyitóoldalra tudunk jutni.

Egy navigációs sáv is lesz majd az oldal tetején, ami megkönnyíti a felhasználói élményt. Bejelentkezés előtt ebben a navigációs sávban négy gombot fogunk látni. Az első a nyitóoldalra fog navigálni. A második egy könyv feliratú gomb lesz. Ha erre kattintunk, akkor a könyvek listáját fogjuk látni. A harmadik gomb a regisztrációhoz fog vezetni, a negyedik pedig a bejelentkezés oldalhoz.

3.2.1 Regisztráció




3.2.1.1. Ábra - Home Page GUI

A nyitó oldalon látni fogunk egy regisztráció feliratú gombot, melyre kattintva meg fog jelenni a regisztrációs felület. Ezt a felületet máshogy is el tudjuk majd érni. A navigációs sávban is lesz egy ilyen gomb, ugyancsak regisztráció felirattal. Ez által is meg tudjuk nyitni a regisztrációs oldalt. De ezek mellett lesz még egy lehetőség is. A bejelentkezési oldalon lesz egy link, ami ehhez a regisztrációs oldalhoz fog vezetni. Ez után lesznek azok a mezők, amelyekbe a saját adatokat kell majd begépelni. Ezek: keresztnév, vezetéknév, email és jelszó. A mezők alatt három gomb is lesz. Az első gombbal a regisztrációt tudjuk befejezni, amennyiben mindent helyesen megadtunk. A második gomb egy törlő gomb. Ezzel tudjuk törölni a más begépelte adatokat. Ez akkor lehet hasznos, ha elrontottunk esetleg több mezőt is, ezzel könnyebb ezeket törölni. Fontos, hogy ennek a gombnak a lenyomása után az összes mezőt törölni, nem csak azt, amibe a gomb lenyomása előtt írtunk. A harmadik pedig egy olyan gomb lesz, aminek a lenyomásával a nyitóoldalra tudunk visszajutni.

A regisztrációhoz szükséges lesz megadni a felhasználó keresztnévét, vezetéknévét és email címét. Ez mellett természetesen egy jelszavat is kérni fogunk a felhasználótól. A jelszavat kétszer kérjük majd begépelni, hogy biztosan helyesen írjuk be a kívánt jelszavat. Ha ez valamilyen okból kifolyólag mégsem sikerül, akkor ebben az esetben hibaüzenettel fogjuk

jelezni, hogy mi a probléma. Az email cím egyedi lesz, tehát egy email címmel csak egyszer lehet majd regisztrálni. Abban az esetben, ha már egy regisztrált email címmel próbálunk regisztrálni, akkor ezt természetesen nem fogja engedni a backend, ezért hibaüzenetet fogunk kiírni. Ha valamit rosszul írunk be, akkor a hibaüzenet megjelenésekor minden előzőleg beírt adat törlődik, ezért érdemes, ha figyelmesen töltjük ki a mezőket. Ha beeregistráltunk, az nem azt fogja jelenteni, hogy rögtön be is jelentkezünk ez által. Ha sikerül a regisztráció, akkor egy nyitólaphoz hasonló oldalra fogunk jutni. Ezen az oldalon majd megjelenik két gomb. Egy gomb, ami átnavigál a bejelentkezés oldalra, majd itt tudunk bejelentkezni. A másik gomb pedig visszánavigál a nyitólapra, amennyiben azt szeretnénk.

[Home](#) [Books](#) [Sign Up](#) [Log In](#)


DSZ
DOKUMENTÁCIÓS SZERZŐI ZSOLNOK
DOKUMENTÁCIÓS SZERZŐI ZSOLNOK

Welcome!

Do you already have an account?
[Log In!](#)

First Name

Last Name

Email

Password

Password Again

Sign Up

Clear

Back to Home


3.2.1.2. Ábra - Regisztráció GUI

3.2.2. Be- és kijelentkezés

Bejelentkezést három úton is elérhetjük. A nyitó oldalon lesz egy gomb, amire ha rákattintunk, akkor megjelenik a bejelentkezéshez szükséges oldal. De ez mellett lesz egy gomb a navigációs sávban is. Ez azért jó, mert ha a nyitóoldalról először a könyveket nézzük meg, akkor nem kell visszamennünk a nyitó oldalra, és aztán a bejelentkezésre, hanem rögtön a navigációs sáv segítségével a bejelentkezéshez tudunk navigálni. Ezen a két lehetőségen kívül lesz még egy harmadik út is a bejelentkezési felület eléréséhez. Ha a regisztrációs felületet nyitjuk meg, lesz egy link, ami átvezet a bejelentkezéshez. Ez azért lehet hasznos, mivel megeshet, hogy véletlenül félre kattintunk, vagy út közben eszünkbe jut, hogy már regisztráltunk korábban, és csak bejelentkezésre lesz szükségünk.

A bejelentkezési felület először köszönti a felhasználót, majd alatta egy link segítségével átnavigálhatunk a regisztrációs felületre is, amennyiben még nem regisztráltunk volna. A regisztrációs oldalon pedig meg kell adnunk az adatainkat. Ezek pedig: keresztnév, vezetéknév, email cím és a jelszó. A bejelentkezés oldalon az email és jelszó beírása után jelentkezhetünk be, amennyiben jó párosítást adunk meg. Ha nem, akkor egy hibaüzenetet írunk ki a felhasználónak. Ha sikeres a bejelentkezés, akkor pedig egy olyan feliratot írunk ki, hogy sikerült a bejelentkezés, és üdvözljük a felhasználót. Ez utána folytathatjuk tovább bejelentkezve a böngészést.

[Home](#) [Books](#) [Sign Up](#) [Log In](#)



DSZ
DOKUMENTÁCIÓS SZOLGÁLTATÁSOK ZRT.

Welcome Back!

Don't have an account?
[Create today!](#)

Email

Password

Sign In

3.2.2.1. Ábra - Login GUI

Mikor bejelentkezőnk, a navigációs sáv is megváltozik. Sima felhasználók esetében ez csupán annyit fog jelenteni, hogy az alaptól a navigációs sávban lévő gombok kicsit megváltoznak (ezek a gombok: nyitóoldal, könyvek, bejelentkezés, regisztráció). Mivel más be vagyunk jelentkezve, értelemszerűen a bejelentkezés és regisztráció oldalra már nem lesz szükség, ezért ezek a navigációs sávból eltűnnek. Az első két gomb, ami a nyitóoldal, és könyvek kilistázása, továbbra is marad, hiszen ugyanúgy szükségünk lehet rá. Harmadik helyen a saját adatlapunkhoz vezető gomb fog elhelyezkedni. Negyedik helyen pedig egy jól megkülönböztethető gomb fog a kijelentkezésért felelni. Fontosnak tartottam, hogy jól látható legyen ez a gomb, hiszen nagyban megkönnyítheti a felhasználói élményt.

Az adminoknál ezzel szemben jóval több változást vehetünk majd észre. Első három helyen a sima felhasználóhoz hasonlóan, a nyitó oldalra vezető gomb, a könyvek kilistázásához vezető gomb és a saját adatlap szerkesztéséhez vezető gomb lesz látható. Ez után egy olyan gombot fogunk látni, ami arra az oldalra fog átnavigálni minket, ahol új könyveket tudunk majd hozzáadni a webalkalmazásunkhoz. Ez mellett lesz egy olyan fül, amely az összes felhasználót hivatott kilistázni az admin részére. Ezt követően pedig egy olyan gomb helyezkedik el, ami a kölcsönzéseket listázza ki. Utolsó helyen itt is a kijelentkezéshez szükséges gomb fog helyet foglalni.

3.2.3. Könyvek kilistázása és szerkesztése

A könyvek kilistázása itt azt jelenti, hogy az összes könyvet látni fogjuk majd. Ezt az oldalra látogató minden felhasználó látni fog, bejelentkezés nélkül is. Nem lesz megjeleníthető az összes tervezett szűrő a könyvekre. Ahhoz hogy a szűrőket meg tudjuk tekinteni, be kell jelentkezni. De ez sem lesz minden esetben elegendő. A sima felhasználó csak az összes könyvet, illetve az épp szabad könyveket tudja majd megtekinteni.

Itt majd egy táblázat fog megjelenni. A sima felhasználók, illetve a még nem bejelentkezett látogatók három oszlopot fog látni. A könyv szerzőjét, könyv címét és ez mellett még egy gombot is.

Ha erre a gombra kattintunk, akkor megjelenik a könyv adatlapja. Ez a sima felhasználóknak nem fog sok új dolgot mutatni. Látni fogjuk a könyv szerzőjét illetve a könyv címét. Az adminok szempontjából majd sokkal több információt fog ez az oldal nyújtani. Nekik

a könyv szerzője és a könyv címe mellett látni fogunk egy táblázatot, amiben látjuk, hogy kik azok a felhasználók, akik ezt az adott könyvet már kikölcsönözték. Ez mellett azt is látni fogjuk, hogy mikor lett kikölcsönözve, és meddig tartott a kölcsönzés. Ezek az információk mellett, még lesz egy gomb is, amivel majd a kölcsönzést vehetjük fel. Erről majd a továbbiakban fogunk részletesebben beszélni.

A könyvek bejelentkezés nélküli kilistázására azért lesz szükség, hogy a felhasználók lássák, hogy milyen könyveket van lehetőségük kikölcsönözni. Hogy érdemes-e bemenni a könyvtárba a könyvért, vagy érdemes-e beregisztrálni az oldalra. Ezzel azt is el tudjuk érni, hogy a felhasználók már céltudatosan menjenek be a könyvtárba, így akár a sima felhasználó, és az adminok idejét is meg tudjuk spórolni.

Az admin jogosultsággal rendelkező felhasználók bejelentkezés után sokkal több dolgot fognak látni, mint a sima felhasználók. Ki tudják listázni a nem csak azokat a könyveket, amik még szabad státuszban vannak, hanem azokat is, amik már ki vannak kölcsönözve. De egyben az összes könyvet is meg tudják majd tekinteni. Ha az admin listázza ki a könyveket, akkor egyben szerkeszteni is tudja. Ez azt jelenti, hogy a sorban három gomb fog megjelenni. A szerkesztés gomb lenyomása után tudjuk szerkeszteni a könyv címét illetve annak szerzőjét. A törlés gombra kattintva értelemszerűen töröljük az adott könyvet, és annak összes kölcsönzését. Ez mellett lesz még egy olyan gomb is, amire ha rákattintunk, akkor meg fog jelenni az adott könyv adatlapja.

3.2.4. Saját adatlap és annak szerkesztése

Home Books My Page Sign Out	
Personal Data	
You can change your personal data if you want.	
Edit	
Last Name	Hencsár
First Name	Dorottya
Email	dorottya@hencsar.com
Password	Change password
Your Books	

3.2.4.1. Ábra - személyes adatok szerkesztése GUI

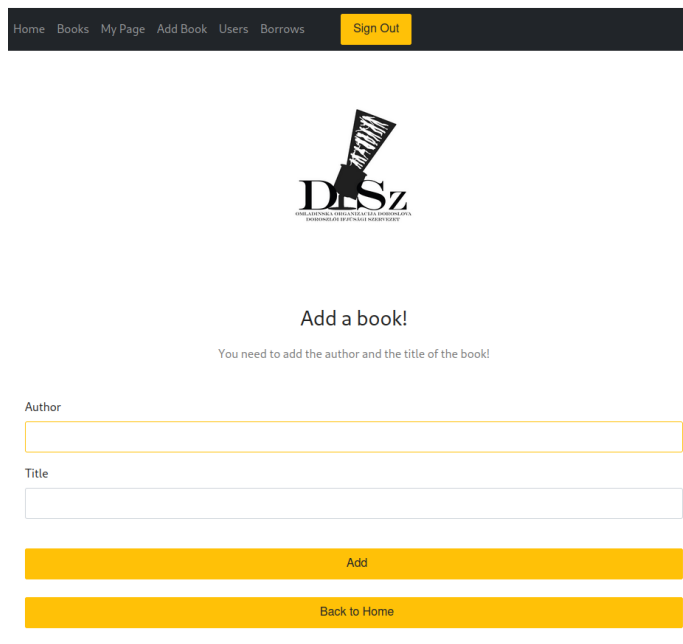
A bejelentkezést követően a sima felhasználók és az adminok is meg tudják majd tekinteni a saját adatlapjukat is. Ezt a navigációs sávon lévő harmadik gomb lenyomása után tehetjük meg. Ez a gomb fog minket a saját adatlapunkhoz vezetni. Ezen az adatlapon megtekinthető a felhasználó keresztnéve, vezetéknéve, email címe és azoknak a könyveknek a listája, amelyeket a felhasználó kikölcsönzött.

Ezeket az adatokat ezen az adatlapon szerkeszteni is tudjuk majd. Egy gomb lenyomásával jelennek meg a mezők amelyekbe majd az új értékeket tudjuk majd megadni. Ezekben a mezőkben halványan fog látszódni az eredeti érték. Ha szerkesztés nélkül mentjük le az adatainkat, akkor vigyáznunk kell, mert megtörténhet, hogy az összes adatunkat töröljük. Szerkeszthetjük majd a keresztnévünket, vezetéknévünket és az email címünket is. Ahogy már fent említettem, az email cím egyedi. Ebben az esetben ez azt jelenti hogy ha egy olyan email címet szeretnénk megadni amit már más használ, akkor hibaüzenet fog figyelmeztetni arra, hogy ezt nem tehetjük meg. A keresztnév és vezetéknév esetében nincsenek ehhez hasonló megkötések.

A jelszavat is cserélhetjük, ha arra lenne igény. Ezt egy másik oldalon tudjuk majd megtenni, amire a személyes adatlapról tudunk majd átnavigálni egy gomb segítségével. Ha rákattintunk erre a gombra, megjelenik a jelszó változtatásra alkalmas felület. Először be kell

írunk a mostani, még aktuális jelszavat, majd ahogyan a regisztrációnál is, kétszer kell elgépelés nélkül beírni az új jelszavat. Ha ez sikerül, akkor sikeresen megváltoztattuk a jelszavat. Ha valami elgépelés történt, vagyis ha rosszul adtuk meg az aktuális jelszavat, vagy ha a két új jelszó nem egyezik meg, akkor kiírunk egy hibaüzenetet, hogy nem sikerült a jelszóváltoztatás.

3.2.5. Könyvek hozzáadása



3.2.5.1. Ábra - Könyvek hozzáadása GUI

Könyveket hozzáadni az adatbázishoz csak az adminok tudnak majd. A könyvek hozzáadása oldalra majd a navigációs sávon keresztül tudunk majd eljutni. Be kell írni a könyv szerzőjét illetve a címét. Ez a két mező alatt két gombot fogunk látni. Az egyikkel hozzáadjuk a könyvet, a másikkal pedig visszajuthatunk a nyitóoldalra. Ha sikerült felvenni a könyvet, kani fogunk egy üzenetet, hogy sikeres volt a felvétel.

3.2.6. Könyv atlap és kölcsönzés hozzáadása

A könyv adatlapját a könyvek kilistázása után érhetjük el. A listában a sima felhasználók három oszlopot látnak majd, a könyv szerzőjét, a könyv címét, és még egy oszlopot. Ebben a plusz egy oszlopban lesz egy gomb. Erre a gombra kattintva nyílik majd meg a kiválasztott könyv

adatlapja. Ezen az adatlapon a felhasználó szemével majd látni fogjuk a könyv címét és szerzőjét.

[Home](#) [Books](#) [My Page](#) [Add Book](#) [Users](#) [Borrows](#) [Sign Out](#)

Ernest Hemingway

Az öreg halász és a tenger

Author

Ernest Hemingway

Title

Az öreg halász és a tenger

Borrow It

Who borrowed it?

Who

From

To

3.2.6.1. Ábra - Könyv adatlapja GUI

Az adminok szempontjából sokkal több információt látunk majd. A könyvek kilistázásakor három gombot fogunk látni az egyes könyvek mellett. Ezek a szerkesztés, törlés és az adatlap megtekintése. Ha az adatlap megtekintésére kattintunk, akkor a sima felhasználóhoz hasonlóan látni fogjuk a könyv címét, és annak szerzőjét, de ez mellett még egy táblázatot is látni fogunk, ami azt fogja tartalmazni, hogy ki kölcsönözte ki az adott könyvet, mikor kezdődött a kölcsönzés, és meddig tartott. Ezek mellett lesz még egy gomb is. Erre kattintva tudják majd az adminok felvenni az adott könyvhöz tartozó új kölcsönzést.

Ezen a kölcsönzés felületen elsősorban ki fogjuk írni a könyv címét és szerzőjét, hogy tudjuk, hogy melyik könyvről van szó, és alatta választhatjuk ki, hogy ki az aki kölcsönözni szeretne. Ezt majd az email alapján lehet megtenni, mivel ez is egy egyedi azonosító. Tehát ha egy felhasználó könyvet szeretne kölcsönözni, akkor így is úgy is be kell mennie a könyvtárba, hogy átvegye a könyvet, ezért jó megoldás lehet az is, hogy mikor az admin felveszi a kölcsönzést, akkor a felhasználó bemondja az email címét, és ez alapján fogjuk hozzárendelni a könyvhöz a felhasználót. Ezt az email címet majd egy dropdown menüből tudjuk majd kiválasztani. Amennyiben a megadott email cím nincs a dropdown menü email címei között, akkor az azt jelenti, hogy még ezzel az email címmel még nem regisztráltak a

webalkalmazásunkban. Ebben az esetben a felhasználónak regisztrálnia kell, ahhoz hogy haza tudja vinni a könyvet. Az email cím mellett be kell állítanunk a dátumot is, hogy mikortól lett kikölcsönözve a könyv. A kölcsönzés befejezésének dátumát nem itt fogjuk majd beállítani. Ezt a kölcsönzések kilistázásánál fogjuk megtenni. Erről majd a későbbiekben esik szó.

3.2.7. Kölcsönzések kilistázása

A kölcsönzéseket csak az admin tudja majd kilistázni. Erre is több szűrő áll majd rendelkezésükre. Ki lehet majd listázni az összes kölcsönzést, a még nyitott kölcsönzést és a már befejezett kölcsönzést, azaz a még nem visszahozott könyvek. A kölcsönzéseket tudjuk majd szerkeszteni, ez által tudjuk megadni az üres mezőbe a kölcsönzés befejezésének dátumát. Fontos, hogy minden más mezőt is tudunk majd szerkeszteni ez mellett. Ezt majd egy naptár segítségével tudjuk megadni.

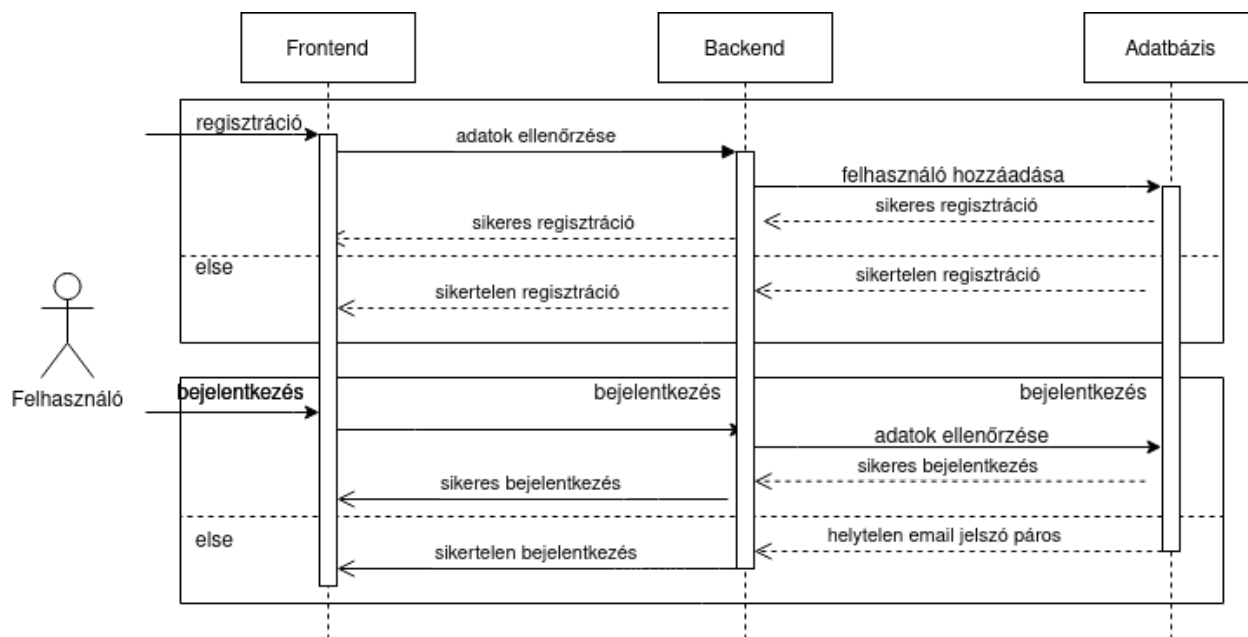
3.2.8. Felhasználók kilistázása

Ezek mellett még az összes felhasználót is ki tudjuk listázni, természetesen csak az adminoknak. Az adminoknak lehetősége van az összes felhasználó adatlapját megtekinteni. Adott esetben még szerkeszteni is tudja majd a felhasználó keresztnévét, vezetéknévét és email címét. Természetesen az adminnak sem lesz joga más felhasználók jelszavát megváltoztatni, így ezt nem tudják szerkeszteni.

3.3. Szekvencia diagram

A Dto-kra, azaz a Data Transfer Object-ekre azért lesz szükség, hogy ne a tiszta entitásokat küldjük a frontend felé. Ezzel védjük az adatokat. Tehát minden változtatás először a Data Transfer Object-eken hajtodik végre, hiszen a frontenden kizárólag azokkal fogunk találkozni. Majd a backend fogja intézni a többit. Mivel Dto-kon keresztül fogunk kommunikálni, ezért szükség lesz converterekre is, hogy át tudjuk alakítani az entitásokat Data Transfer Object-ekre, és vissza. Ezt majd a Service rétegben tesszük majd meg.

3.3.1. Regisztráció



3.3.1.1. Ábra - Regisztráció és bejelentkezés szekvencia diagram

A regisztrációs oldalon először megadjuk az adatainkat, ez a `signup.js`-ben tesszük. Ezeket az adatokat elküldtük a megadott adatokat a `useSignup.js` fájlba. Az email cím formátumát még a legelső réteg, azaz a `signup.js` fogja leellenőrizni. Ez után az átkerült adatokból a `useSignup.js` létrehoz egy `PersonDto`-t. Ezt a `PersonDto`-t fogjuk továbbküldeni a backendnek egy `Post` method-dal, a `PersonController` segítségével.

A backend a `Service` rétegben először le fogja ellenőrizni, hogy a kapott email cím már létezik-e az adatbázisban. Amennyiben igen, akkor egy hibaüzenetet fogunk dobni, hogy az email cím már létezik, így ezzel nem lehet még egyszer regisztrálni. Ha pedig még szabad az adott email cím, akkor először át kell alakítanunk `PersonDto`-ból `Person` entitássá. Ehhez egy `converter` metódust hoztam létre a `Service` rétegben. Ez a `converter` fogja egyesével lekérni a `PersonDto`-tól az adatokat, és ezekkel az adatokkal fogunk létrehozni egy új `Person` objektumot. Ezt az objektumot fogja elmenteni az adatbázis.

A jelszavakat természetesen nem csak simán `String` formátumban fogjuk lementeni. Ha így tennénk, sokkal sebezhetőbb lenne a webalkalmazásunk. Ezért enkriptálni fogjuk a

jelszavakat. Erre a spring BCryptPasswordEncoder-ét választottam. Ezzel védtem le a jelszavakat, így ezzel is hozzátettem ahhoz, hogy biztonságosabb legyen a webalkalmazás.

Ha bármilyen hiba felmerülne a mentés közben, például ha nem csatlakozunk az adatbázishoz, vagy ha már létezik az email cím az adatbázisban, és ez miatt nem sikerül a mentés, akkor ezeket a hibaüzeneteket természetesen visszaszivárogtatjuk a frontendig, és meg is jelenítjük egy általános hibaüzenettel. Egyenlőre egy általános hibaüzenetet fogunk csak megjeleníteni, ha a későbbiekben szükség lenne részletesebb leírásra a hibaüzenetekkel kapcsolatban, akkor ezt majd könnyen meg is tehetjük a jövőben.

3.3.2. Be- és kijelentkezés

Bejelentkezéskor a webalkalmazás el fogja kérni az email címet és az ehhez tartozó jelszavat. Ezt a párosítást kell majd leellenőriznie a backendnek. Ezt a párosítást a login.js fogja elkérni. Ezt a párosítást aztán a useLogin.js fogja kezelni. Létre fogunk hozni egy AuthenticationDto-t. Ez a megadott email címet és jelszavat fogja tartalmazni. Ez a Data Transfer Object-et fogjuk továbbküldeni egy post method-dal. Létrehozunk UsernamePasswordAuthenticationToken-t a LoginControllerben, aminek át fogjuk adni a az email jelszó párosítást. Majd meghívjuk az Authentication osztály authenticationManager metódusát. Ez a metódus fogja autentikálni a felhasználót. Ez azt jelenti, hogy megnézzük, hogy az email címhez tartozó jelszó megegyezik-e azzal, amit megadtunk a bejelentkezésnél. Amennyiben helyes az email jelszó párosítás, akkor a felhasználónak sikerült a bejelentkezés, és megjelennek azok a funkciók is, amiket bejelentkezés nélkül nem érhetünk el. Ha viszont nem azt a jelszavat írjuk be, amelyik a megadott email címhez tartozik, vagy akár helytelenül írjuk be az email címet, vagy jelszavat, akkor unauthorized http status-t adunk vissza. Ez akkor is így lesz, ha még nem regisztráltunk az adott email címmel. Hiszen ha nem regisztráltunk be az adott email címmel, akkor nem is tudjuk megnézni, hogy milyen jelszó tartozott hozzá.

Kijelentkezéshez pedig amikor a kijelentkezés gombra kattintunk, a useLogout.js fogja meghívni a LoginController logout metódusát. Ez is egy post metódus, ami majd nullra fogja állítani az autentikációt. Ezzel a felhasználó ki lesz jelentkeztetve.

3.3.3. Könyvek kilistázása, adatlapja, szerkesztése és hozzáadása

A könyvek listáját bárki láthatja, bármitől függetlenül. Ez azt jelenti, hogy nemcsak az adminok látják, de a felhasználók is. Sőt bejelentkezés nélkül is láthatjuk a könyvek listáját.

Ha rákattintunk a gombra, amely a displayBooks-ban van definiálva, amely a könyveket listázza ki, akkor a useDisplayBooks által meghívjuk a BookController getBooks get metódusát. Ez a metódus a Service réteghez fog fordulni. A BookService egy interface a projektben, amibe az úgynevezett üzleti logikát írtam le. Ennek egy implementációt hoztam létre, amelyben kifejtettem az összes metódust, ez lett a BookServiceImp. Ebben a BookServiceImp implementációban van a findAllBooks nevű metódus is. Ez a metódus majd egy Dto-kból álló listát fog visszaadni. Ez a metódus a fog fordulni a BookRepository-hoz, amely majd vissza fogja adni a könyveket. Itt is meg kell említeni azt, hogy Data Transfer Object-eket, azaz Dto-kat fogunk visszaküldeni. A repository természetesen nem Dto-kat tárol, ezért ezt át kell konvertálni a megfelelő konverterrel. Ezt a konvertert a BookServiceImp-ben hoztam létre egy toDto metódust. Ez a metódus majd át fogja alakítani az adatbázisból kapott entitást Dto-kra. Mivel fent már említettem, hogy egy Dto-kból álló listát szeretnénk, ezért létrehoztam még egy metódust, a toDtos metódust. Ez a metódus majd egy entitásokból álló listát vár. Ebben a metódusban a kapott paraméter lista összes entitás elemére meghívja a már fent is említett toDto metódust, amivel az entitásból Dto-t készítünk, és ezeket az elemeket újra listába rendezi. Majd ezt a már BookDto-kból álló listát fogjuk visszaadni. Ezzel a konverterrel fogjuk átalakítani a BookRepository-ből kapott entitások listáját Data Transfer Object-ekből álló listává. Ezzel az összes könyvet vissza fogjuk adni ami az adatbázisban van.

Ez a BookDto-kból álló listát fogjuk a frontendnek átadni. A könyveket soronként fogjuk megjeleníteni. Az első oszlopban majd a könyv szerzőjét fogjuk megjeleníteni, a másodikban pedig a könyv címét. A harmadik oszlop egy mindenki által elérhető “View” gomb lesz. Ezzel a gombbal tudjuk majd megtekinteni a könyv adatlapját. Ez a sima felhasználóknak és a még nem bejelentkezett felhasználóknak nem mutat sok újat, ugyanis ez számukra csak a könyv címét, és a könyv szerzőjének nevét fogjuk kiírni.

Ha rákattintunk erre az említett “View” gombra, akkor az adott sorban lévő könyvhöz tartozó id azonosítóját elküldjük egy property-vel a bookPage.js file-nak. Ez után a useBookPage.js fogja megkérdezni a backendtől, hogy pontosan melyik is ez a könyv. A

BookController-ben van egy olyan metódus, ami a `getBook` névre hallgat, ami természetesen `GetMapping`. Ez egy `BookDto`-t fog visszaadni, és vár egy paramétert. Ez a paraméter a könyv egyedi azonosítója lesz. Itt is a controller a service réteghez fog fordulni, és annak a `getById` metódusához. Ez a metódus fogja az adatbázisból lekérni a megfelelő könyvet. Ha nincs olyan egyedi azonosító az adatbázisban, amit éppen keresünk, akkor egy null-t adunk vissza. Ha null-al tér vissza, akkor üres lesz a könyv címének és szerzőjének a mezője.

Az admin jogokkal rendelkező felhasználók még látnak egy táblázatot is. Ebben a táblázatban majd látható lesz hogy az adott könyv ki által és mikor lett kikölcsönözve. A könyvet kikölcsönző felhasználó vezetéknevét, keresztnévét és email címét is ki fogjuk írni. Az email cím azért lesz esetlegesen fontos, hogy azonosítsuk a felhasználót, abban az esetben ha a nevük megegyezik. Emellett még lesz két oszlop. Az egyikben azok a dátumok fognak szerepelni, amikor a könyvet kikölcsönözték, a másikban pedig, hogy mikor hozták vissza. Abban az esetben, ha a könyvet még nem hozták vissza, akkor természetesen ez a mező üresen marad.

Ezek mellett még lesz még egy gomb is. Ez a gomb fog minket átnavigálni arra az oldalra, ahol a kölcsönzést fogjuk felvenni. A könyv egyedi azonosítóját fogjuk elküldeni egy property által. Ez által tudja majd a kölcsönzés oldal, hogy melyik könyvről van szó. Erről majd a későbbiekben lesz szó.

A könyvek listázásakor nem csak a “View” gomb lesz látható az adminok számára. Ez mellett lesz még két gomb is. Ilyen az “Edit” gomb is. Ezzel majd szerkeszteni tudjuk a könyvet. Erre egy úgynevezett inline megoldást választottam. Ez azt jelenti, hogy ha rákattintunk erre a gombra, akkor a könyv szerzője és címe helyén megjelennek olyan mezők, melyekbe az új adatokat írhatjuk bele. Mivel a szerkesztést soronként oldottuk meg, ezért nem kell külön foglalkozni, hogy melyik könyvről van szó, mert adott sorban adott könyvet fogunk tudni szerkeszteni. Két gomb lesz még ez a két mező mellett. Az egyik a “Cancel” gomb, amely visszaállítja az eredeti verziót, és nem történik semmi. Újra csak a könyvek kilistázása fog megjelenni. Valamint lesz még egy “Save” gomb is. Ezzel fogjuk majd lementeni a változtatásokat. A `displayBooks.js` elküldi a `useDisplayBooks.js`-nek azokat az adatokat, amiket begépettünk a könyv szerkesztése gyanánt. Az új adatokat a BookController `updateBook` put metódusával fogjuk frissíteni. Az `updateBook` metódus egy `BookDto`-t vár paraméterül, amelyik könyvet szeretnénk szerkeszteni. Ez mellett még két stringet is vár, a cím és szerző új értékét. Ez az `updateBook` metódus is a service réteghez fog fordulni. A `BookServiceIm`ben lévő

updateBook metódus is ugyanazokat a paramétereket várja, mint a BookController-beli. Első lépésként a már fentebb is említett converter-rel átalakítjuk a Data Transfer Object-et entitássá. Ez után setterek segítségével beállítjuk az új adatokat, és lementjük az entitást. A metódus természetesen nem tér vissza semmivel sem.

Szerkesztés mellet még törlésre is lesz lehetőségünk, ha a “Delete” gombra kattintunk. Ebben az esetben először a összes könyvhöz tartozó kölcsönzést kell törölnünk, utána tudjuk majd csak az adott könyvet is törölni. A könyv egyedi azonosítóját küldjük át majd a BorrowController deleteAllByBook metódusának. A rétgeten keresztülhaladva fogjuk majd törölni az adatbázisból az adott könyvhöz tartozó összes kölcsönzést. Majd ez után fogjuk törölni magát a könyvet. Ennek a törlésének esetében is a szokásos rétegeken fogunk keresztül haladni a cél elérése érdekében.

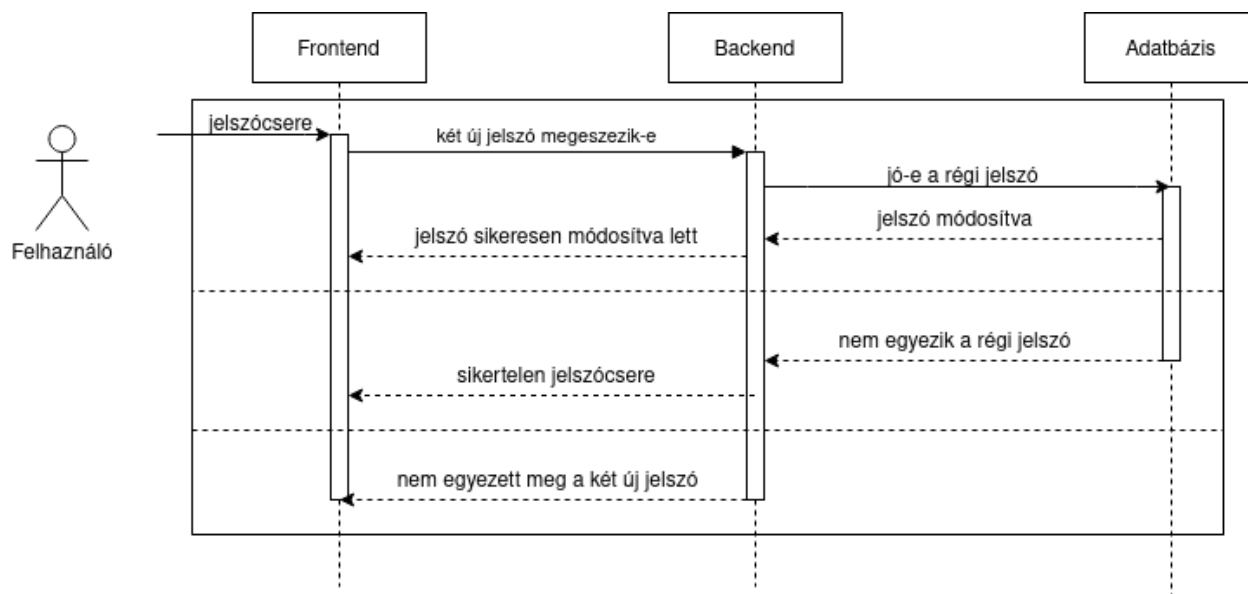
A könyvek hozzáadását is hasonlóan oldottam meg. Csak az adminok tudnak majd új könyveket felvenni. Ezt a bejelentkezés és regisztrálás oldalához nagyon hasonló oldalon tehetik meg. Csak a könyv címét és szerzőjét kell megadni. Ezt elküldjük a beakadnak, ami átalakítja a kapott BookDto-t entitássá, majd az adatbázis lementi az adatot. Útközben természetesen generálunk neki egyedi azonosítót is. Mivel a könyvtárban egy adott könyvből lehet több is, ezért nincs semmilyen követelmény arra, hogy valamelyik bevitt adat egyedi legyen.

3.3.4. Saját adatlap szerkesztése és jelszócsere

A saját adatlap szerkesztését mindenki elérheti. Tehát mindenki tudja majd szerkeszteni a saját adatlapját. Ez mellett az adminok is tudják más felhasználók adatlapját is szerkeszteni. Miután kilistázzák az összes felhasználót, meg tudják majd tekinteni a felhasználók adatlapját is. Itt lesz majd lehetőségük arra is, hogy majd a felhasználók adatait tudják szerkeszteni, ugyan úgy, mintha a saját adataikat szerkesztenék, természetesen a jelszavat nem tudják majd módosítani. A felhasználók szerkesztése esetén is ugyanazon a rétegeken fogunk átmenni, mint a könyvek esetében. Természetesen a felhasználók esetében PersonController, PersonService és PersonRepository-t fogunk használni.

A jelszavat minden felhasználó tudja majd változtatni magának. Ehhez pedig majd egy másik oldalra fog minket navigálni az alkalmazás. Egyszer majd be kell írni a még aktuális jelszavat, majd kétszer az újat. Itt majd a webalkalmazás le fogja ellenőrizni, hogy jól adtuk-e meg a

még aktuális jelszavat. Ez a Service rétegben fog megvalósulni. Azt, hogy az új jelszavat kétszer kell helyesen beírni, még a frontend fogja ellenőrizni. Majd ha ez jó, akkor átadjuk a service rétegnek a régi jelszavat ellenőrizni, majd ha az is rendben van, akkor sikeresen jelszavat fogunk cserélni.



3.3.4.1. Ábra - jelszócsere szekvenciadiagramja

3.3.5. Kölcsönzések listázása és hozzáadása

A kölcsönzéseket az adminok vehetnek fel. Fent már említettem, hogy majd a könyv adatlapjáról tudunk eljutni a kölcsönzés oldalára. Egy property segítségével fogjuk elküldeni a könyv egyedi azonosítóját az addBorrow.js-nek. Majd a kölcsönzés hozzáadása oldalon két értéket kell bevinnünk. Ez emailt fogjuk a könyvhöz rendelni, mivel ez is egyedi a személyek tekintetében. Ehhez egy dropdown menüt használunk majd. Ennek segítségével fogjuk majd kiválasztani a felhasználót, aki majd kikölcsönzi a könyvet. Ehhez meg kell hívnunk a getAllPersons metódust is a PersonContollerből.

Majd egy naptár segítségével azt is fel kell venni, hogy mikortól kezdődik a kölcsönzés. Természetesen a könyvet nem lehet kikölcsönözni abban az esetben, ha az már másnál van. Ezt egy boolean változóval kezeljük majd.

A kölcsönzéseket ki is listázhatjuk. A kilistázás alkalmával tudjuk majd szerkeszteni a kölcsönzéseket. Ha esetleg rosszul adtuk meg a kezdeti dátumot, akkor azt is tudjuk majd korrigálni. De a kölcsönzés végdátumát is itt tudjuk majd megadni. Természetesen a végdátum nem lehet korábban, mint a kezdeti dátum. Ezt már a frontendben leellenőrizzük. Ha hibás adatot adunk meg, akkor egy hibaüzenetet ad majd a weboldal. A backend oldalon pedig a Service réteg lesz az, amely nem fogja lementeni a későbbi dátumot. Így nem kaphatunk olyan hibát, amely azt eredményezi, hogy összekeveredjenek a dátumozások.

4. Architektúra

4.1. Frontend

A frontend a felelős a különböző input mezőkért, és az adatok szépen való megjelenítéséért. A frontendet a Javascriptes ReactJs keretrendszerben írtam meg. React esetében minden olyan funkciót, amelyet több helyen is felhasználható, külön modulokba rendezzük.

React-ra jellemzőek az úgynevezett Hook-ok. Minden hook egy-egy funkcionalitásra írunk meg. Ez azt jelenti, hogy akár többször is felhasználhatjuk őket. Tehát ez által sokkal szebb, és átláthatóbb kódot kapunk. Ez a továbbiakban abban is segít, hogy könnyebben bővíteni tudjuk majd a projektet.

A useState és a useEffect két alapvető hook. Ezeket a React biztosítja a fejlesztőknek. A useState adatot tárol, amit változtatni lehet a weblap megjelenítése közben. A useState általános definíciója így néz ki: `const [data, setData] = useState(null)`. Egy változónevet adunk meg neki elsőnek, aztán pedig a hozzá tartozó függvényt, amivel módosítani tudjuk. A szakdolgozat írása közben főleg ezt használtam.

Ez mellett a useEffect célja, hogy egy függvényt, amelyet bizonyos esetekben le akarunk futtatni, akkor ezt elintézzék nekünk. Tehát minden egyes esetben, mikor változik a kapott megkötés, akkor le fogja futtatni azt a bizonyos függvényt.

4.1.1. A Login hook-jának bemutatása

Az alábbi képen a bejelentkezés Hook-ját mutatom meg. Láthatjuk már az importokon is, hogy mit fogunk használni. A LoginController-rel a backend végpontjához fogunk kapcsolódni. Az AuthenticationDto-t arra fogjuk használni, hogy elküldjük a backendnek az email és jelszó párost. Ezt fogja majd utána leellenőrizni a backend.

A login.js fogja megjeleníteni a weboldalt. Itt elkérjük az adatokat, vagyis az email címet, és a jelszavat. Rákattintunk a Bejelentkezés gombra, és ekkor hívjuk meg a useLogin handleSubmit függvényét.

```

import { useState } from "react";
import LoginControllerApi from "../../api/src/api/LoginControllerApi";
import AuthenticationDto from "../../api/src/model/AuthenticationDto";
import useGetLoggedInUser from "../getLoggedInUser";

const useLogin = () => {
  const LoginController = new LoginControllerApi()

  const {getLoggedInUser, user} = useGetLoggedInUser()
  const [error, setError] = useState()

  const handleSubmit = (e, email, password) => {
    e.preventDefault();
    const login = new AuthenticationDto()
    login.email = email
    login.password = password
    LoginController.loginUsingPOST(login, function(e){
      if(e !== null){
        setError("Your authentication was unsuccessful.\n\nPlease check your email address, password, or your network connection.")
      }else{
        getLoggedInUser()
      }
    })
  }
  return {handleSubmit, user, error}
}

export default useLogin

```

4.1.1.1. Ábra - Login hook-ja

Létrehozunk egy login nevű változót, amely egy AuthenticationDto-t fog tárolni. Majd a kapott email címet és jelszavat átadjuk neki. Ezt a login változót küldjük át a LoginController segítségével a backendnek. Ahogyan meghívjuk a LoginController loginUsingPOST metódusát, átadjuk neki a login változót, valamint egy névtelen függvényt is. Lényegében ez fogja visszaadni a backend válaszát, amennyiben történik valami hiba. Ebben az esetben egy hibaüzenetet fogunk kiírni, hogy a felhasználó is láthassa, hogy mi lehet a gond. Ebben az esetben azt osztjuk meg a felhasználóval, hogy sikertelen volt a bejelentkezés, majd felhívjuk a figyelmét arra, hogy mi lehet a probléma. Itt most felhívjuk a figyelmet arra, hogy előfordulhat, hogy rossz email címet adott meg, vagy esetleg a jelszavat gépelte el a user. De arra is figyelmeztetünk, hogy az is megtörténhet, hogy az internetkapcsolat szakadt meg, és ez miatt nem sikerül a bejelentkezés. A későbbiekben az akár bővíthető is lehet, hogy külön kezeljük a különböző eseményeket. Tehát ha rossz a párosítás, akkor azt írjuk ki hibaüzenetként. Ha pedig

az internetkapcsolattal lenne a probléma, akkor pedig olyan hibaüzenetet adunk, ami erre figyelmeztet.

4.1.2. Jelszócsere hook-jának bemutatása

```
import { useState } from "react"
import PersonControllerApi from "../../api/src/api/PersonControllerApi"

const useChangePsw = () => {

  const PersonController = new PersonControllerApi()
  const [error, setError] = useState(null)
  const [success, setSuccess] = useState()

  const handleSubmit = (e, person, oldPsw, newPsw1, newPsw2) => {
    e.preventDefault()
    try{
      if(newPsw1 !== newPsw2){
        throw Error("New passwords not match")
      }else{
        if(run){
          PersonController.pswChangeUsingPUT(newPsw1, oldPsw, person, function(error){
            if(error===null){
              setSuccess("Your password has changed.")
            }else{
              throw Error(error)
            }
          })
        }
      }
    }catch(err){
      setError(err)
      console.log(error)
    }
  }

  return [handleSubmit, error, success]
}
export default useChangePsw
```

4.1.2.1. Ábra - Jelszócsere hook-ja

A jelszó cseréjét illetően látjuk, hogy csak a PersonController-re lesz szükségünk. De először még a changePsw.js bekéri az adatokat. Egyszer a régit és kétszer az újat, valamint azt a felhasználót, aki épp a jelszavát szeretné frissíteni. Azzal kezdünk, hogy még a frontend oldalon, hogy leellenőrizzük, hogy az új jelszónak megadott két érték megegyezik-e. Amennyiben ez a feltétel nem valósul meg, akkor nem is fogjuk továbbküldeni a backendnek az adatokat további ellenőrzésre. Felesleges lenne, hiszen mivel nem egyeznek az új jelszónak szánt értékek, így

amúgy sem mentenénk el új jelszóként. Ha ez a hiba fennáll, akkor adunk egy hibaüzenetet a felhasználónak, hogy figyelmeztessük a problémára.

Amennyiben ezen az ellenőrzésen továbbjutunk, akkor adjuk át a backend-nek az adatokat. Meghívjuk a `PersonController` `pswChangeUsingPUT` végpontját. Átadjuk a régi, illetve újnak szánt jelszavat is, valamint a felhasználót. Itt is lesz egy névtelen függvény. Ha az *error* változó nem *null*, azaz a backend valami hibával tér vissza, akkor azt itt is továbbdobjuk, felszivárogtatjuk a `console`-ig.

Ha viszont nincs semmi probléma, akkor a *success* változónak adunk értéket. Ezt az értéket átadjuk majd vissza a `changePsw.js`-nek. Amennyiben sikerül a jelszóváltoztatás, beállítjuk a változót, majd ez az üzenet megjelenik a weboldalon.

4.2. Backend

A backend fog szerepet vállalni abban, hogy a webalkalmazás gondtalanul működjön. Az ő szerepe lesz az is, hogy az adatokat megjelenítse. Azokat az adatokat, amit épp szeretnénk látni az adatbázistól. Ehhez Java Spring Boot-ot választottam. Valamint az Apache Maven-t használtam a build folyamatok automatizálásához.

A Maven egyik sajátossága a POM nevű `.xml` file. A név jelentése Project Object Model, azaz magyarul Projekt Objektummodell. Ebben írjuk le a különböző függőségeket. Már mikor a <https://start.spring.io/> weboldalról letöltjük a projektet, akkor is már az alap függőségekkel töltjük le azt. Ezen kívül a fejlesztés során felvettem minden olyan függőséget is, amely a folytatáshoz szükséges volt.

Az entitás osztályok felvételével kezdtem. Három entitás lett. A `Person`, `Book` és `Borrow`. Ezekhez az *@Entity* annotációt használtam. Ezek mellé még felvettem a hozzájuk tartozó Data Transfer Objecteket. Felvettem az ezekhez tartozó Repository interface-eket is. Ezek mentik el az adatokat az adatbázisban, valamint ennek a segítségével tudunk adatokat is elkérni. Az alap lekérdezéseket implementáció nélkül is tudni fogja, ilyen például a `getAll()` metódus is. Ha pedig valami nem ennyire triviális, akkor pedig megadhatjuk azt a query-t is, amivel a kívánt adatokat tudjuk lekérni.

A Service réteg is érdekes számunkra. Először Service nevű interface-eket hoztam létre. Ebből is hármat, minden entitáshoz egyet. Ezek fogják tartalmazni az úgynevezett üzleti logikát. Ide leírunk minden olyan függvényt, amit majd a későbbiekben implementálunk, és szeretnénk használni. Ez után a Service interface-ek mellé aztán készítettem mindegyik mellé egy-egy implementációt is. Ezekben az implementációkban adjuk meg mindazt a függvényt, amelyet majd fel fogunk használni a végpontok kialakításához.

A Controller réteg pedig magát a végpontokat jelentik. A frontendtől kapott adatokat innen továbbküldve fogjuk feldolgozni.

4.2.1. A könyv szerkesztésének bemutatása a backend-ben

```
@RolesAllowed("ROLE_ADMIN")
@PutMapping
public void updateBook(final @RequestBody BookDto bookDto, String author, String title, String category) {
    bookService.updateBook(bookDto, author, title, category);
}
```

4.2.1.1. Ábra - BookController updateBook metódusa

A `@RolesAllowed` annotációval megadtuk azt, hogy csak az admin jogosultságú felhasználót tudnak majd könyveket szerkeszteni. A `@PutMapping` pedig egyértelműen arra utal, hogy a HTML PUT kérését szeretnénk használni. Látjuk azt is, hogy milyen adatokat várunk el. Majd ezt követően pedig a Service-hez fordulunk, és meghívjuk annak az `updateBook` metódusát, és átadjuk neki az összes szükséges paramétert.

```
@Override
public void updateBook(BookDto bookDto, String author, String title, String category){
    Book book = toEntity(bookDto);
    book.setAuthor(author);
    book.setTitle(title);
    book.setCategory(category);
    bookRepository.save(book);
}
```

4.2.1.2. Ábra - BookServiceImpl updateBook metódusa

Ezt követően a Service-ben levő metódusban először a Dto-t kell entitássá alakítani. Ehhez egy privát metódus a felelős. Majd a kapott entitást setter-ekkel fogjuk módosítani. Amint az összes kapott elemet felülírtunk, lementjük a módosított entitást az adatbázisban.

4.2.2. Nem triviális query-k a BookRepository-ből

A könyvek esetében volt pár olyan lekérdezés, amely nem volt triviális a Repository számára. Ezért külön kellett query-ket írnom ahhoz, hogy megkapjam azokat az adatokat, amelyekre szükségem van.

Két ilyen is volt. Az egyik, amikor azokat a könyveket szeretnénk megjeleníteni, amelyek még nem lettek kikölcsönözve. Ennek a módszernek a *findAllNotBorrowed* nevet adtam. A másik ilyen módszer pedig azokat a könyveket adja vissza, amelyek pedig ki vannak éppen kölcsönözve. Ennek a *findAllBorrowed* nevet adtam.

```
@Repository
public interface BookRepository extends JpaRepository<Book, Long> {

    @Usage
    List<Book> findAllByAuthorContainsOrTitleContains(String query, String query1);

    @Usage
    @Query("SELECT book FROM Book book LEFT JOIN book.borrows borrow ON borrow.endTime IS NULL WHERE borrow.id IS NULL")
    List<Book> findAllNotBorrowed();

    @Usage
    @Query("SELECT book FROM Book book LEFT JOIN book.borrows borrow ON borrow.endTime IS NULL WHERE borrow.id IS NOT NULL")
    List<Book> findAllBorrowed();
}
```

4.2.2.1. Ábra - Nem triviális lekérdezések a BookRepository-ből

5. Összefoglaló, továbbfejlesztési lehetőségek

A szakdolgozat során sok dolgot tudtam tanulni. A fejlesztés megkezdése előtt sok ötletem volt. Majd lassan rájöttem, hogy ehhez még nem ismerem a technológiákat annyira, hogy megvalósítsam. Majd ahogy haladtam előre, egyre több, az elején még lehetetlennek tűnő funkciót tudtam megvalósítani. Ez mellett megtanultam azt is, hogy sok esetben nem szabad megrögzötten ragaszkodni az egyes tervekhez. Sokszor ugyanis el kell engedni pár dolgot, hogy más megoldás felé orientálódjunk. Azt is megtanultam, hogy bizony sok esetben kell újratervezni. Egy-egy feladatot talán nem is lehet megvalósítani, vagy még nincs annyi tapasztalatom, hogy sikerüljön. Ezzel egy kis türelmet is tanultam.

A különböző technológiák nagyban segítették a szakmai fejlődésem. Nem is kérdés, hogy mindenképp szeretném majd továbbfejleszteni az alkalmazást. Ahogyan írtam a dokumentációt, rengeteg továbbfejlesztési lehetőség megfogalmazódott bennem. Az elsődleges terveim közé tartozik például az is, hogy a könyveket kategorizálni tudjuk majd. Most még csak amiatt nincs benne, mert úgy érzem, hogy nincs annyi könyv még a szervezet tulajdonában, hogy ez elengedhetetlen legyen. Emellett még esetleg kommentelési lehetőséget is lehetne biztosítani a könyvekhez. Ez például akkor lehetne jó, ha valaki nem tudja, hogy egy könyvet ki szeretne-e venni, vagy nem, akkor a kommenteket olvasva talán könnyebb dönteni. Ez mellett még hasznos lehet majd egy emlékeztető email is. Mivel bejelentkezéskor úgyis elkérjük az email címet, így tulajdonképpen semmi akadálya. Meg tudjuk szabni, hogy egy könyv meddig lehet valakinél, és ha ez a dátum közeledik, akkor küldünk egy emailt. De akár arra is adhatunk majd lehetőséget, hogy a felhasználó meghosszabbítsa a kölcsönzést. Továbbá, ha már nagyobb lesz az igény, akkor felvehetünk egy újabb szerepet is a könyvtárosnak. Így az admin feladatait megtartjuk, de a könyvtáros esetében sokkal kevesebb jogosultsága lesz. Mondjuk csak annyi, hogy könyveket vesznek fel, és adjon ki. De a könyvek szerkesztését lehet, hogy megtartom csak az adminnak. Ezen kívül szeretnék még egyszerű weboldalt is készíteni a szervezetnek. Majd arról átnavigálni a könyvtár webalkalmazásra. És még egy olyan ambícióm is van, hogy egy androidos appot is készítek hozzá. Erre nem lenne szükség, de én szeretnék az android fejlesztéssel is megismerkedni.

Felhasznált irodalom, források

1. ReactJs - <https://reactjs.org/> (Utolsó megtekintés: 2022.12.08.)
2. Node.js - <https://nodejs.org/en/> (Utolsó megtekintés: 2022.12.08.)
3. Visual Studio Code - <https://code.visualstudio.com/> (Utolsó megtekintés: 2022.12.08.)
4. Spring Boot - <https://spring.io/projects/spring-boot> (Utolsó megtekintés: 2022.12.08.)
5. Java - <https://www.java.com/en/> (Utolsó megtekintés: 2022.12.08.)
6. Apache Maven - <https://maven.apache.org/> (Utolsó megtekintés: 2022.12.08.)
7. IntelliJ IDEA - <https://www.jetbrains.com/idea/> (Utolsó megtekintés: 2022.12.08.)
8. Swagger - <https://swagger.io/> (Utolsó megtekintés: 2022.12.08.)
9. Swagger OpenAPI - <https://swagger.io/specification/> (Utolsó megtekintés: 2022.12.08.)
10. Git - <https://git-scm.com/> (Utolsó megtekintés: 2022.12.08.)
11. Github - <https://github.com/> (Utolsó megtekintés: 2022.12.08.)
12. 10 Features You Should Consider for Your Next Web App -
<https://www.devwerkz.com/blog/10-features-you-should-consider-for-your-next-web-app>
- (Utolsó megtekintés: 2022.11.08.)
13. Diagrams.net - <https://app.diagrams.net/> (Utolsó megtekintés: 2022.12.08.)
14. Java Programming Masterclass updated to Java 17 -
<https://www.udemy.com/course/java-the-complete-java-developer-course/> - (Utolsó megtekintés: 2022.12.08.)
15. Spring Boot Tutorial | Full Course [2022] [NEW] -
<https://www.youtube.com/watch?v=9SGDpanrc8U> - (Utolsó megtekintés: 2022.12.08.)
16. React - The Complete Guide (incl Hooks, React Router, Redux) -
<https://www.udemy.com/course/react-the-complete-guide-incl-redux/> - (Utolsó megtekintés: 2022.12.08.)
17. Disz - <https://www.facebook.com/doroszloiifjusagiszervezet> (Utolsó megtekintés: 2022.12.08.)
18. Kód elérhetősége: <https://github.com/hdotty/fullstack> (Utolsó megtekintés: 2022.12.10.)

Nyilatkozat

Alulírott Hencsár Dorottya programtervező informatikus alapszakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztési Tanszékén készítettem, Programtervező informatikus Bsc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel. Tudomásul veszem, hogy szakdolgozatomat / diplomamunkámat a Szegedi Tudományegyetem Diplomamunka Repozitóriumban tárolja.

Dátum

2022.12.10.

Aláírás

Köszönetnyilvánítás

Szeretnék köszönetet nyilvánítani a témavezetőmnek, Dr. Bilicki Vilmosnak, a sok türelemért, és a sok jó tanácsért. Hogy jó kommunikációs csatornát épített ki számunkra, és hogy hétről hétre ellenőrizte a munkánkat, hogy a legjobban sikerüljön. A kedves, biztató szavakat, amelyek megkönnyítették a munkánkat.

Elektronikus mellékletek

A könyvtár adminisztrációs webalkalmazás kódjának online elérése:

<https://github.com/hdotty/fullstack>