

**SZEGEDI SZAKKÉPZÉSI CENTRUM**  
**VASVÁRI PÁL GAZDASÁGI ÉS INFORMATIKAI**  
**TECHNIKUM**

**5 0613 12 03**

**Szoftverfejlesztő és -tesztelő**

**ZÁRÓDOLGOZAT**

**BookWorm**  
Korszerű könyvtár

Készítette:  
**Balogh Norbert**  
**Tápai Nándor**

**Szeged**

**2022**

## Tartalomjegyzék

BEVEZETÉS .....	- 4 -
1. FELADAT MEGHATÁROZÁSA.....	- 5 -
1.1. A rendszer célja .....	- 5 -
1.2. Dokumentáció készítése .....	- 5 -
2. RENDSZERSPECIFIKÁCIÓ .....	- 6 -
2.1. Funkcionális követelmények .....	- 6 -
2.1.1. Asztali alkalmazás – Adminoknak és könyvtárosoknak .....	- 6 -
2.1.2. Mobil alkalmazás – Könyvtárt használóknak .....	- 6 -
2.1.3. Weboldal – mindenkinek .....	- 7 -
2.2. Nem funkcionális követelményei .....	- 7 -
3. ADATBÁZIS .....	- 8 -
3.1. Adatbázis tervezés – (E-K diagram, Bachmann ábra).....	- 9 -
3.2. Adatbázisban táblák felépítése .....	- 10 -
3. FEJLESZTŐI DOKUMENTÁCIÓ.....	- 13 -
4.1. Backend (Balogh Norbert) .....	- 13 -
4.1.1 Technológiai háttér.....	- 13 -
4.1.2. Backend megvalósítása .....	- 15 -
4.1.3. Backend tesztelése.....	- 17 -
4.2. Mobil alkalmazás (Balogh Norbert).....	- 21 -
4.2.1. Technológiai háttér.....	- 21 -
4.2.2 Megvalósítás.....	- 21 -
4.3. Webes alkalmazás .....	- 23 -
4.3.1 Technológiai háttér.....	- 23 -
4.3.2 Megvalósítás.....	- 23 -
4.3.3 Tesztelés .....	- 25 -
4.4. Asztali alkalmazás (Tápai Nándor) .....	- 26 -

4.4.1 Technológiai háttér.....	- 26 -
4.4.2. Megvalósítás.....	- 26 -
4.4.3. Tesztelés .....	- 28 -
5. FELHASZNÁLÓI DOKUMENTÁCIÓ .....	- 30 -
5.1. Asztali alkalmazás (Tápai Nándor) .....	- 30 -
5.2. Webalkalmazás.....	- 32 -
5.3. Mobil applikáció (Balogh Norbert).....	- 33 -
5.4. Docker-compose (Balogh Norbert).....	- 36 -
6. ÖSSZEGZÉS.....	- 37 -
FORRÁSJEGYZÉK.....	- 38 -
MELLÉKLETEK .....	- 39 -
Hallgatói nyilatkozat .....	- 44 -

## BEVEZETÉS

A XXI. század egyik meghatározó eleme a digitalizáció, amit egyre több kis- és nagyvállalkozás vezet be és vált át a papír alapú adminisztrációról. Napjainkban a mobiltelefont az emberek alapvető szükségletüknek tekintik és a mai gyermekek már egészen kis korunktól kezdve használják különböző funkcióit.

Záródolgozatunkban mi is ezt a „társadalmi problémát”, a növekvő munkaerőhiány és a rohamosan digitalizálódó világ adta lehetőséget megragadva hosszas egyeztetéseket követően készítettük el a BookWorm alkalmazást.

Az alap ötletet egy gyermekkori barátom adta, akinek a munkáját nagyban megkönnyítené egy digitális bárholnan bárki által elérhető rendszer. Így neki sem kellene papír alapon rögzítenie a kölcsönzéseket, az aktuálisan visszahozott vagy selejtezett könyveket, mert arra lenne egy központi adatbázissal ellátott rendszer. Így született meg a BookWorm ötlete, ami egy modern könyvtári nyilvántartó és kölcsönzési rendszer, amely valós igényeknek és elvárásoknak megfelelően lett megtervezve és elkészítve.

Az elkészítés során igyekeztünk mind a végfelhasználóknak mind az adminisztrációt végző könyvtárosoknak a lehető legnagyobb segítséget nyújtani a fejlesztéseinkkel.

Kezdetben felmértük, hogy jelenleg miként működik a könyvtári könyvek nyilvántartása, kölcsönzése, visszahozása és a selejtezése.

A rendszer tervezése során felmértük az esetleges új igényeket és igyekeztünk beépíteni azokat, természetesen a régi jól bevált funkciók folyamatok megtartása, esetleges felokosítása mellett.

# 1. FELADAT MEGHATÁROZÁSA

A záródolgozatunk öt részből, komponensből tevődik össze. Az egész rendszer alapját képező adatbázisból, az ehhez kapcsolódó backend és asztali rendszerekből, valamint a kliens (mobil alkalmazás és web alkalmazás) rendszerekből. A kliens oldalak egyenként teljesen más funkciókat látnak el. A weboldalt bárki számára látogatható, azon könyvek, kategóriák tekinthetők meg. Van lehetőség regisztrálni és e-mail küldeni az üzemeltetőknek is. Ezzel szemben a mobil alkalmazás jelenlegi formája kifejezetten azoknak a felhasználóknak készült akik kölcsönözni szeretnének, a kölcsönzéseiket szeretnék menedzselni. Ezeket a backend szolgálja ki.

Ezeket felül a belső üzemeltetésért (pl.: kölcsönzések, könyvtárosok és könyvek kezelése) az asztali alkalmazás felel, amely közvetlenül kapcsolódik az adatbázishoz.

## 1.1. A rendszer célja

A kialakított rendszernek három fő célja van:

1. Csökkenteni a papír alapú, felesleges adminisztrációs munkát
2. Átláthatóbbá és könnyebben kezelhetővé tenni a készletkezelést
3. Könnyíteni a felhasználók, könyvtárosok munkáját (elegendő egy telefonos applikációról leolvasott vonalkód)

Az elkészített szoftverek e két cél megvalósítását végzik. A webes alkalmazás segíti a natúrkozmetikumokról való egyszerű és gyors tájékozódását és azok egyszerű megvásárlását. Míg az asztali alkalmazás segíti az adatbázis kezelés (új elem hozzáadás, elem módosítás, elem törlés) egyszerű és könnyű kivitelezését.

## 1.2. Dokumentáció készítése

A záródolgozatot és a dokumentációt közösen készítettük, amelyhez a Microsoft Office programcsaládot és a Flowchart Maker online diagram készítő szoftverét használtuk. Mindketten főként az általunk készített alkalmazások dokumentációját írtuk meg amelyet a társunk lektorált. Ahol az adott fejezetnél nincs külön feltüntetve a készítő neve, akkor az teljes mértékben közös munka volt.

## **2. RENDSZERSPECIFIKÁCIÓ**

A rendszer megtervezésénél arra törekedtünk, hogy a már jól bevált funkciókat megtartsuk, csak azt modernizált környezetbe helyezzük és felokosítsuk. Így jutottunk el a ténylegesen megvalósítandó projekt tervéhez, miszerint készítünk egy kizárólag a könyvtárból elérhető és használható asztali alkalmazást, amely az adminok és a könyvtárosok által használt funkciókat hivatott ellátni.

Ezen felül készítünk egy mobil alkalmazást, amely első körben csak a „sima felhasználónak” készül. Itt tudják az adataikat módosítani, profilképet feltölteni és követni a könyvtárhasználatukat.

Ezekon felül készítettünk még egy webes felületet, ami főként egy bemutatkozó és regisztrációs felület. Természetesen ezen keresztül a felhasználók kapcsolatba is tudnak lépni velünk e-mail formájában.

Összességében ennek az egésznek a motorja egy szerver oldali alkalmazás és egy relációs (RBDMS) adatbázis.

### **2.1. Funkcionális követelmények**

#### **2.1.1. Asztali alkalmazás – Adminoknak és könyvtárosoknak**

- Bejelentkezés
- Új felhasználó hozzáadása
- Felhasználók keresése
- Felhasználók inaktiválása
- Könyvek hozzáadása, módosítása
- Könyvek keresése
- Kölcsönzés és visszahozás
- Szerzők listázása
- Új szerző hozzáadása, módosítása

#### **2.1.2. Mobil alkalmazás – Könyvtárt használóknak**

- Be és kijelentkezés
- Elfelejtett jelszó
- Jelszó megváltoztatása
- Személyes adatok szerkesztése
- Profilkép módosítása

- Könyvtári könyvek megtekintése
- Korábbi kölcsönzések megtekintése
- Aktuális kölcsönzések megtekintése
- Kapcsolati lehetőségek megtekintése

#### 2.1.3. Weboldal – mindenkinek

- Könyvek megtekintése
- Könyvek megjelenítése kategóriának megfelelően
- Kapcsolatfelvétel e-mail-ben
- Regisztráció

## 2.2. Nem funkcionális követelményei

Legfontosabb nem funkcionális követelménye a rendszernek, hogy stabilan működjön és használható legyen. Az adott funkciókhoz csak a funkcióhoz társított jogkörrel (*ADMIN*, *USER*, *LIBRARIAN*) rendelkező bejelentkezett felhasználó férjen hozzá.

Ebből kifolyólag szükség van jelszavak tárolására is, amelyet egyirányú titkosítással oldottunk meg. A felhasználói élvény és használhatóság miatt a mobil alkalmazáson lehetőség van új jelszó kérésére is, amelyet az adott felhasználó a regisztrált e-mail címére kap meg.

Az alkalmazás komponenseknek – backend kivételével – grafikus felülettel kell rendelkezniük, amelynek felhasználóbarátnak és könnyen tanulhatónak kell lennie, valamint több különböző eszközön (mobil, PC, tablet) is egyaránt esztétikusan kell kinéznie és megfelelően kell tudnia használni a klienseknek. Így tehát a mobil és a webes alkalmazás is reszponzívan készült.

## 2.3. Futtatási követelmények

A programoknak futásához szükséges és elégséges egy általános felhasználói hardverkörnyezet. Minimum Windows 7 operációs rendszer és 1 GB RAM ajánlott a zökkenőmentes futtatáshoz. Az alkalmazás minden komponensének szüksége van stabil internet kapcsolatra. Ennek hiányában a megfelelő működés nem garantált.

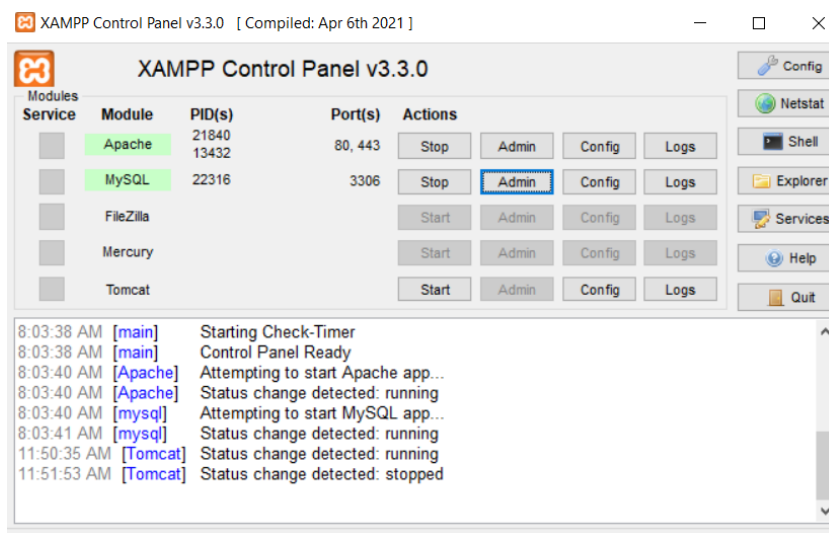
Az adatbázis használatához szükséges a XAMPP nevű program telepítése és beüzemelése vagy a backend részen található Docker-compose indítása, amely platform függetlenül felhúzza az adatbázist a szükséges adatokkal, elindítja és csatlakoztatja a backend alkalmazást az

adatbázishoz és kiülteti a megadott helyi portokra. Ehhez egyedül egy Docker szükséges amelyet a weboldalról könnyedén letölthetünk.

### 3. ADATBÁZIS

A rendszer egy központi adatbázis adataival dolgozik, így mondhatjuk, hogy a stabil jól megtervezett adatbázis a rendszer alappilléreinek tekinthető.

Az adatbázis létrehozásához és használatához 2 különböző szoftvert is használtunk. Az egyik ilyen többek által ismert és szeretett program a **XAMPP**, amely képes Apache webservert és MySQL adatbázis kezelő futtatására, amelyhez egy PHP nyelven készült webes felületet biztosít.



1. kép – XAMPP vezérlő felület

A másik szoftver a **DBeaver**, amely a legtöbb adatbázishoz (SQL és NoSQL egyaránt) képes kapcsolódni. Ezt a szoftvert főként a Docker-ben indított adatbázis kapcsolódásához használtuk.

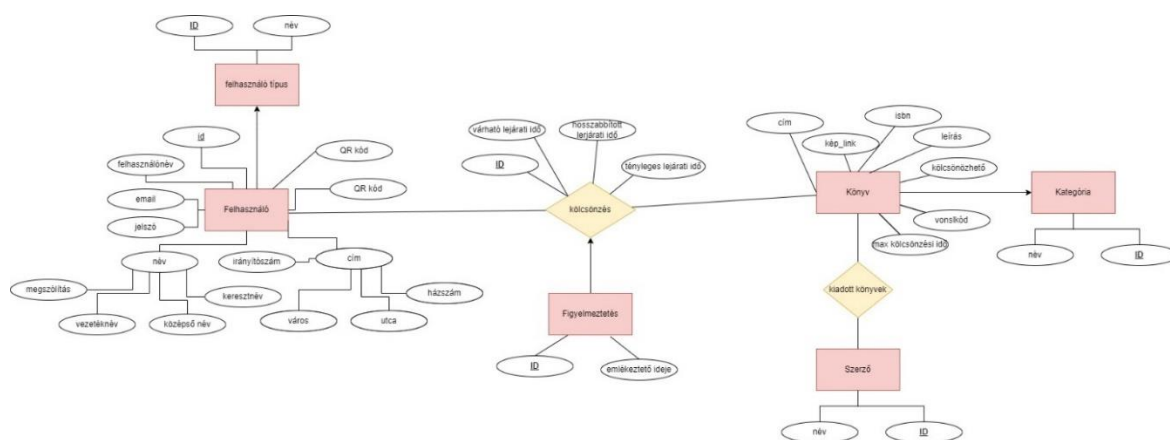


id	name	created_date	last_mod_date	last_mod_user
1	Iskolat	2021-12-20 15:11:17	nbalogh	[NULL]
2	Eletmód, egészségt	2021-12-20 15:11:17	nbalogh	[NULL]
3	Család	2021-12-20 15:11:17	nbalogh	[NULL]
4	Film	2021-12-20 15:11:17	nbalogh	[NULL]
5	Utazás	2021-12-20 15:11:17	nbalogh	[NULL]
6	Jelek	2021-12-20 15:11:17	nbalogh	[NULL]
7	Téka	2021-12-20 15:11:17	nbalogh	[NULL]
8	Egyéb	2021-12-20 15:11:17	nbalogh	[NULL]
9	Pénz, gazdaság, üzlet	2021-12-20 15:11:17	nbalogh	[NULL]
10	Művészet, építészet	2021-12-20 15:11:17	nbalogh	[NULL]
11	Hobbi, szabadidő	2021-12-20 15:11:17	nbalogh	[NULL]
12	Gastronómia	2021-12-20 15:11:17	nbalogh	[NULL]
13	Tartózkodás, segédanyagok	2021-12-20 15:11:17	nbalogh	[NULL]
14	Gyermekek és ifjúság	2021-12-20 15:11:17	nbalogh	[NULL]
15	Történelem	2021-12-20 15:11:17	nbalogh	[NULL]
16	Vallás	2021-12-20 15:11:17	nbalogh	[NULL]
17	Képregény	2021-12-20 15:11:17	nbalogh	[NULL]
18	Hangoskönyvek	2021-12-20 15:11:17	nbalogh	[NULL]

2. kép – DBeaver kezelőfelület

### 3.1. Adatbázis tervezés – (E-K diagram, Bachmann ábra)

A project adatbázisának tervezése során szükség volt egy vázra, amely szemléltette, hogy miként kapcsolódnak az egyes funkciókhoz tartozó adatokat tároló táblák egymáshoz. Az E-K diagram segítségével átláthatóbb az alkalmazás fő célja.

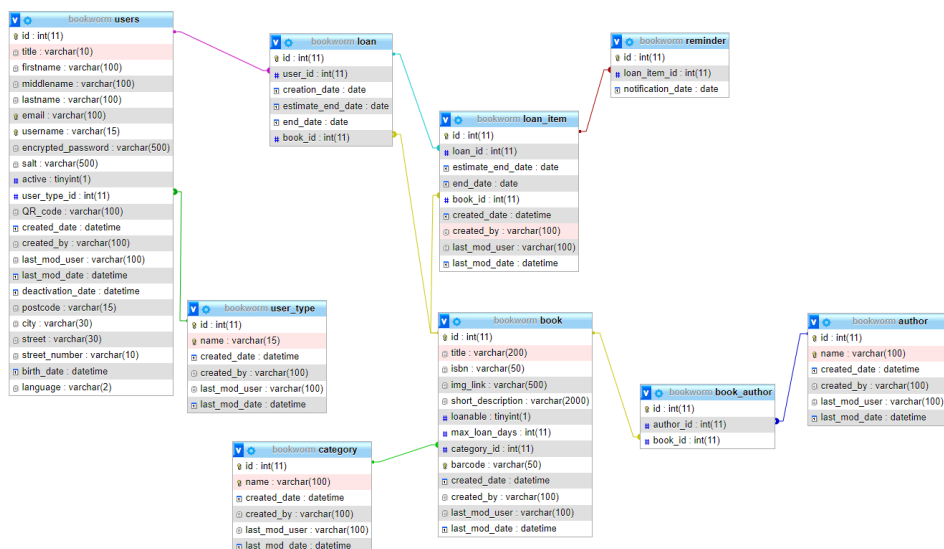


1. ábra - E-K diagram

A projektnek egy 'bookworm' nevű adatbázist készítettünk, ahol az adatbázis tábláinak és oszlopainak elnevezése során igyekeztünk követni az SQL nyelvben használt alapelveket, vagyis mindent angol nyelven a szóösszetételek mentén '\_' használatával hoztunk létre. Az adatbázis kilenc táblát tartalmaz, amelyek kapcsolatban vannak egymással.

Az összes tábla esetén elsődleges kulcsként **id** mező került kialakításra, amely egész szám típusú (*int*) és az adatbázis által automatikusan növelt (*AUTO\_INCREMENT*).

Az adatbázis végleges szerkezetét és tábláinak felépítését és kapcsolatát a következő Bachmann ábra mutatja.



2. ábra - Adatbázis Bachmann ábrája

### 3.2. Adatbázisban táblák felépítése

Az adatbázis a relációs adatbázis alapelveknek megfelelően angol nevezéktannal és alul vonással (    ) a szóösszetételek mentén készült.

A táblák tartalmaznak auditor mezőket, amelyek minden esetben a következők, így ezeket az egyes tábláknál nem tüntetem fel egyesével, csak *auditor\_mezők*-kel jelzem a jelenlétüket.

Az auditor mezők a következők:

*created\_date* – **DATETIME** - A létrehozás ideje

*created\_by* – **VARCHAR(100)** – A létrehozó user/process neve

*last\_mod\_user* – **VARCHAR(100)** – A rekordot utoljára módosító user/process neve

*last\_mod\_date* – **DATETIME** – A dátum amikor a rekordot utoljára módosították

Egyedek – adatbázis táblák:

author, book, book\_author, loan, book\_loan, category, users, user\_type, reminder

Tulajdonságok – adott táblák oszlopai:

AUTHOR( **ID**, name , *auditor\_mezők* )

CATEGORY( **ID**, name , *auditor\_mezők* )

USER\_TYPE( **ID**, name , *auditor\_mezők* )

BOOK( **ID**, title , isbn, img\_link, short\_description, loanable, max\_loan\_days, *category\_id*, barcode, *auditor\_mezők* )

BOOK\_AUTHOR ( **ID**, *book\_id* , *author\_id* ) – Kapcsolótábla a könyvek és a szerzőik között. Megvalósítva a ManyToMany kapcsolatot.

BOOK\_LOAN( **ID**, estimate\_end\_date, end\_date, *book\_id*, extended\_date, *user\_id*, *auditor mezők*)

REMINDER( **ID**, *loan\_id*, notification\_date, *auditor mezők* )

USER(**ID**, title, firstname, lastname, middlename, email, username, phone\_number, birth\_date, language, encrypted\_password, salt, active, *user\_type\_id*, QR\_code, deactivation\_date, postcode, city, street, street\_number, user\_img, *auditor mezők*)

### **Egyedek és tulajdonságaik (adattípusok):**

Az auditor mezők csak a Author (szerző) tábla esetén kerülnek feltüntetésre, a többi esetben csak jelzem a jelenlétüket, de minden esetben ugyanazok.

#### *Author – (szerző)*

<b>ID</b>	szám (INT) – AUTO_INCREMENT – elsődleges kulcs
name	szöveg (VARCHAR(100)) – a szerző neve
created_date	idő (DATETIME) – a létrehozás ideje
created_by	szöveg (VARCHAR(100)) – létrehozó user/process neve
last_mod_user	szöveg (VARCHAR(100)) – utoljára módosító user/process neve
last_mod_date	idő (DATETIME) – utolsó módosítás ideje

#### *Book – (könyv)*

<b>ID</b>	szám (INT) – AUTO_INCREMENT – elsődleges kulcs
title	szöveg (VARCHAR(200)) – a könyv címe
isbn	szöveg (VARCHAR(50)) – a könyv ISBN száma
img_link	szöveg (VARCHAR(500)) – a könyv képének url címe
short_description	szöveg (VARCHAR(2000)) – Rövid leírás
loanable	szám (TINYINT) – megmondja, hogy kölcsönözhető-e a könyv
max_loan_days	SZÁM (INT) – hány napra lehet maximum kikölcsönözni
barcode	szöveg (VARCHAR(50)) – vonalkód
category_id	SZÁM (INT) – idegen kulcs
<i>**auditor mezők</i>	

#### *book\_author - kapcsolótábla*

<b>ID</b>	szám (INT) – AUTO_INCREMENT – elsődleges kulcs
author_id	SZÁM (INT) – idegen kulcs author táblára
book_id	SZÁM (INT) – idegen kulcs book táblára

#### *category – (kategória)*

<b>ID</b>	szám (INT) – AUTO_INCREMENT – elsődleges kulcs
name	szöveg (VARCHAR(100)) – a szerző neve
<i>**auditor mezők</i>	

#### *reminder – emlékeztetők a felhasználóknak*

<b>ID</b>	szám (INT) – AUTO_INCREMENT – elsődleges kulcs
loan_id	SZÁM (INT) – idegen kulcs loan táblára
notification_date	idő (DATETIME) – mikor került kiküldésre ez az emlékeztető

*user\_type – (felhasználó típusok)*

<b>ID</b>	szám (INT) – AUTO_INCREMENT – elsődleges kulcs
name	szöveg (VARCHAR(100)) – a szerző neve
<b>**auditor mezők</b>	

*Book\_loan – (kölcsonzések)*

<b>ID</b>	szám (INT) – AUTO_INCREMENT – elsődleges kulcs
estimate_end_date	dátum (DATE) – kölcsönzés lejáratára, eddig kell visszahozni
end_date	dátum (DATE) – ténylegesen ekkor történt a visszahozatal
book_id	szám (INT) – idegen kulcs a book táblára
extended_date	dátum (DATE) – hosszabbított dátum
user_id	szám (INT) – idegen kulcs a user táblára
<b>**auditor mezők</b>	

*users – (könyv)*

<b>ID</b>	szám (INT) – AUTO_INCREMENT – elsődleges kulcs
title	szöveg (VARCHAR(10)) – a könyv címe
firstname	szöveg (VARCHAR(100)) - keresztnév
lastname	szöveg (VARCHAR(100)) – családnév
middlename	szöveg (VARCHAR(100)) – középső név
email	szöveg (VARCHAR(100)) – e-mail cím
username	szöveg (VARCHAR(15)) – felhasználónév
encrypted_password	szöveg (VARCHAR(500)) – titkosított jelszó
salt	szöveg (VARCHAR(500)) – jelszó titkosításhoz használt
QR_code	szöveg (VARCHAR(100)) – QR kód
active	szám (TINYINT) – aktív vagy nem
user_type_id	szám (TINYINT) – idegen kulcs a user_type táblára
deactivation_date	dátum (DATETIME) – felhasználó inaktíválásának ideje
postcode	szöveg (VARCHAR(15)) – irányítószám
city	szöveg (VARCHAR(30)) – város
street	szöveg (VARCHAR(30)) – utca
street_number	szöveg (VARCHAR(10)) – házszám
birth_date	dátum (DATE) – születési dátum
language	szöveg (VARCHAR(2)) – nyelv pl.: HU
phone_number	szöveg (VARCHAR(15)) – Telefonszám
user_img	fájl (LONGBLOB) – felhasználó által feltöltött kép
<b>**auditor mezők</b>	

### 3. FEJLESZTŐI DOKUMENTÁCIÓ

#### 4.1. Backend (Balogh Norbert)

##### 4.1.1 Technológiai háttér

A backend oldali **RESTful** API-ok elkészítéséhez a JetBrains által készített és supportált *IntelliJ IDEA 2021.3.3 Ultimate Editiont* használtam mint fejlesztőkörnyezetet. Ezen felül a szerver oldali backend alkalmazást **JAVA** nyelv **11-es** verziójával, **Maven** csomagolással és **Spring boot 2.5.5.-ös** keretrendszerrel valósítottam meg.

A project során több Spring által karbantartott és néhány külső dependencyt is használtam, amelyeket Maven segítségével húztam be a projektembe.

```
<!-- DOCUMENT -->
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-boot-starter</artifactId>
  <version>3.0.0</version>
</dependency>

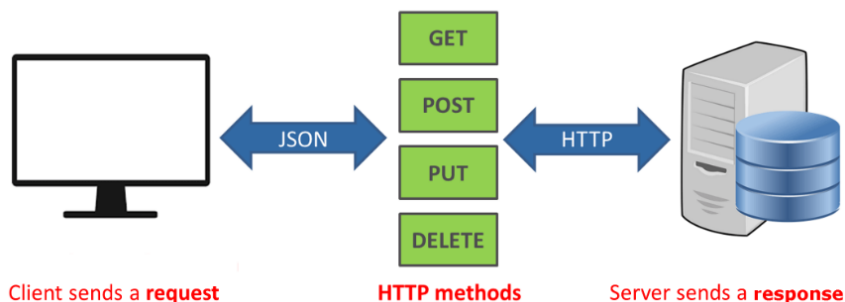
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
```

3. kép – Maven dependency hozzáadása

Ezek közül a legfontosabbakat külön kiemelem.

#### Spring Web:

Tartalmaz egy beépített Apache Tomcatet, amin futtatni lehet az alkalmazást és tartalmazza a **RESTful** és **MVC** project készítéséhez szükséges függőségeket.



3.ábra – REST API működése

## Spring Security:

Authentikációért (*Ki vagy?*) és autorizációért (*Mihez van jogod?*) felelős. Teljes mértékig személyre szabható.

## Spring Data JPA:

Adatbázis (*Persistence*) réteg kezelésért felelős. Hasonló szerepet tölt be mint az **Entity Framework** a .NET-ben készített alkalmazások esetén. A **JPA** - a **Hibernate** egyik legjobb és legelterjedtebb implementációja - is egy **ORM** (*Object-Relational Mapping*).

```
1 package com.library.model.entities;
2
3 import ...
4
5 @Entity
6 @Table(name = "book")
7 @AllArgsConstructor
8 @NoArgsConstructor
9 @Setter
10 @Getter
11 public class Book extends AbstractEntityLogStamp {
12
13     @Size(max = 200)
14     @NotBlank
15     @Column(name = "title", nullable = false, updatable = false, length = 200)
16     private String title;
17
18     private String isbn;
19
20     @Column(name = "img_link")
21     private String imgLink;
22
23     @Column(name = "short_description")
24     private String shortDescription;
25
26     @Column(columnDefinition = "BIT", nullable = false, length = 1)
27     @Type(type = "org.hibernate.type.NumericBooleanType")
28     private boolean loanable;
29
30     @Column(name = "max_loan_days", nullable = false)
31     private Integer maxLoanDays;
32 }
```

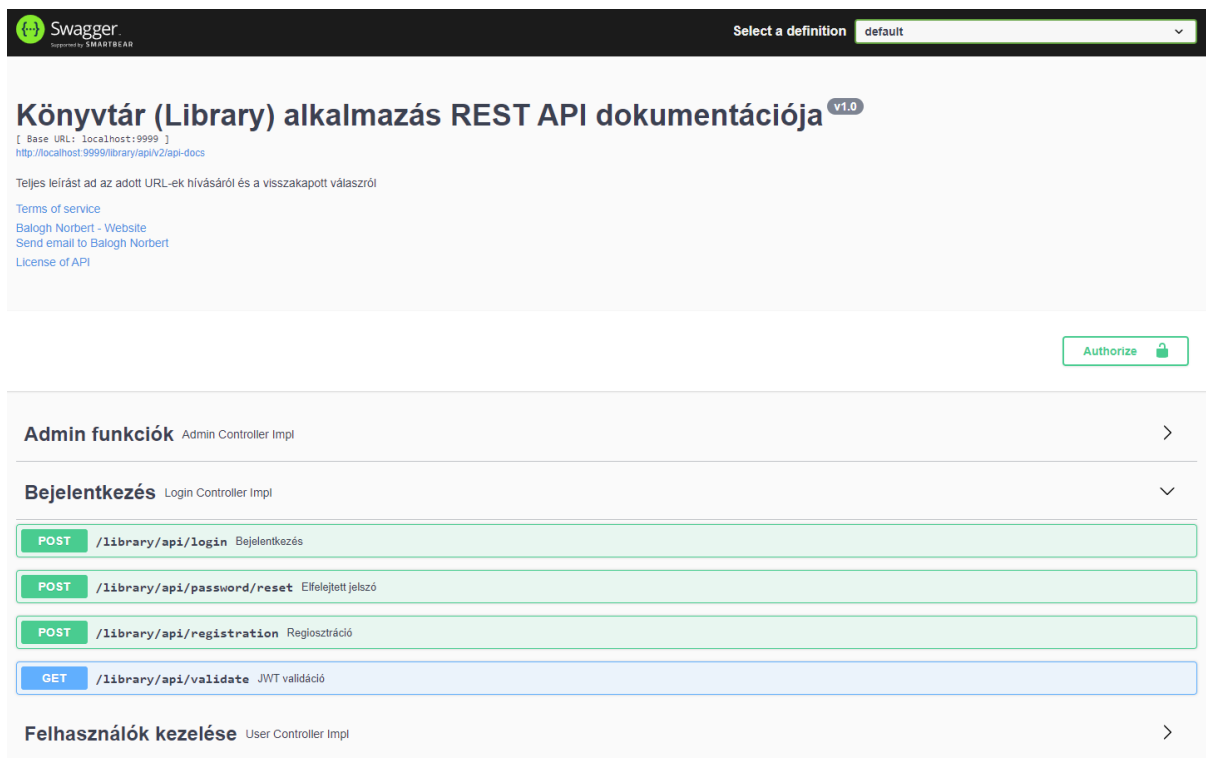
4. kép - Könyv (*Book*) entitás JPA-s annotációkkal

## Thymeleaf - (*Template engine*)

Felülírt megjelenítéshez vagy mint jelen esetben is, dinamikus **HTML** e-mailek készítéséhez használható.

## Spring Mail – e-mail küldés teszi lehetővé

## SpringFox – Swagger UI biztosítása



## 5. kép – Swagger UI – API dokumentáció

## Spring Test

A teszteléshez szükséges eszközöket tartalmazza (**JUnit5** és **Mockito**)

### 4.1.2. Backend megvalósítása

A backend **RESTful** (*REpresentational State Transfer*) API-eket (*Application Programming Interface*) biztosít amelyeket az egyes kliens alkalmazások HTTP-on (*HyperText Transfer Protocol*) keresztül tudnak meghívni. A pontos **URL**-ek (*Uniform Resource Locator*) leírását és a kérés – válasz felépítését Swagger által kigenerált weboldal dokumentálja (*lásd 1. melléklet*).

A backend alkalmazás fő feladata, hogy az adott kliensnek (webalkalmazás vagy mobil applikáció) hozzáférést adjon és alapvető adatkezelési műveleteket (**CRUD** – *Create Read Update Delete*) lásson el a központi adatbázisban tárolt adatokon. Ehhez a standard HTTP metódusokat használtam.

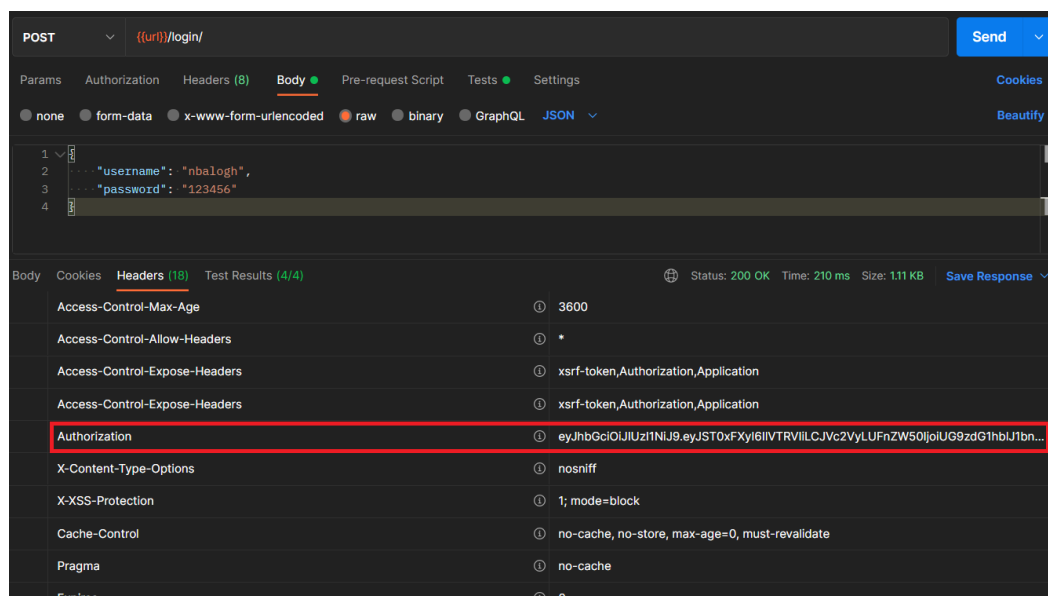
- **GET** - adat lekérdezése
- **POST** – létrehozás, adatot küld a szerver felé
- **PUT** – frissítést, módosítást (update) hajt végre az adaton
- **PATCH** – részlegesen frissíti, módosítja (update) az adatot.

- **DELETE** – törli az adatot az adatbázisból (fizikailag vagy logikailag)

Természetesen vannak publikusan elérhető és autentikációhoz kötött végpontok is, de ezek esetén sincs minden felhasználónak joga mindegyik funkció használatához.

Az autentikációért és az autorizációért a Spring Security felelős, amely megfelelő konfigurálását követően tudja, hogy az adott felhasználó be van-e jelentkezve és az adott végpontot elérheti, van-e joga használni vagy sem.

Ahhoz, hogy a felhasználó elérje a funkciókat szükség van egy publikusan hívható bejelentkezést biztosító **REST API**-ra. Bejelentkezést követően a felhasználó a válasz részeként kap egy a backend által hitelesített **JWT**-t (JSON Web Token) az *Authorization* fejlécben, amelyet a következő hívásokkor (amennyiben az adott végpont megköveteli azt) küldenie kell a kérés fejlécében (*request header*).



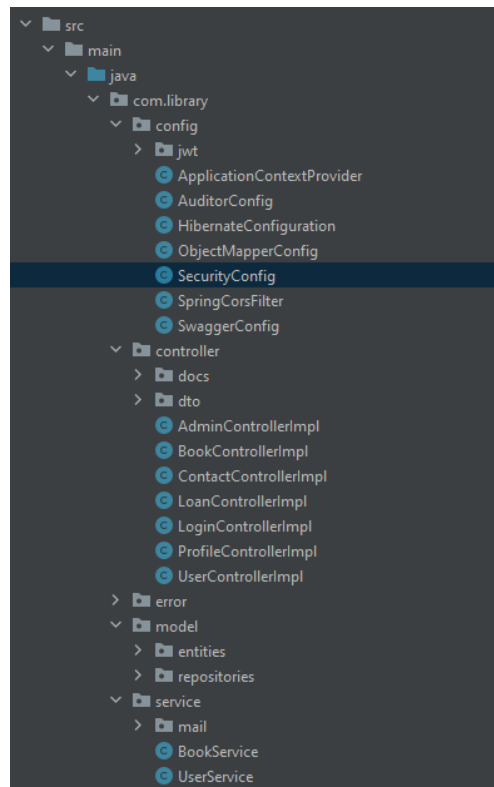
6. kép - JWT a válasz *Authorization* fejlécében

A Spring Security ezt a token-t fogja ellenőrizni. Amennyiben azonosítás szükséges az adott végponthoz, de ezt a kérés nem tartalmazza, vagy nem beazonosítható belőle a felhasználó, akkor a backend a RESTful szabványoknak megfelelően 401-es HTTP státuszkóddal (*UNAUTHORIZED*) tér vissza. Ha a felhasználó beazonosítása sikeres, de nincs joga az adott URL használatához akkor 403-as HTTP státuszkódot kap vissza (*FORBIDDEN*).

A programkódot csomagokra bontva (*package*) helyeztem el, követve az MVC (Model-View-Control) struktúrát. Így elkülönül egymástól az Controller réteg - ami a bejövő hívásokért



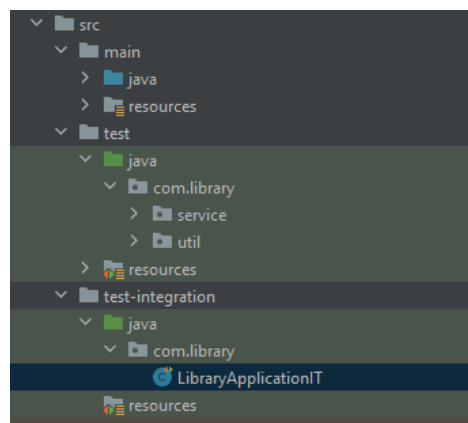
felelős -, a Service réteg ami magát az üzleti logikát tartalmazza és az Model réteg ami az adatbázis entitásokat reprezentálja JAVA nyelven.



7. kép – project struktúra

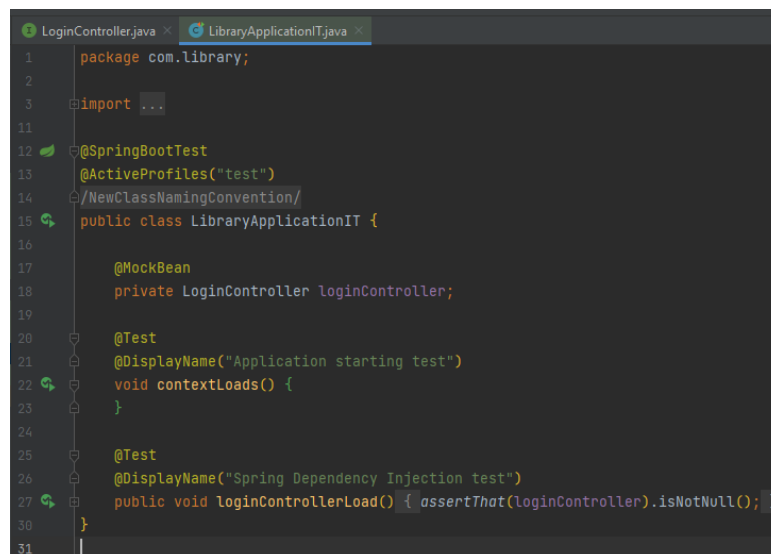
#### 4.1.3. Backend tesztelése

A backend tesztelés során a **JAVA** teszteknel közkeletű **JUnit5** keretrendszert és **Mockito** használtam. A kód teszteléséhez három különböző fajta tesztelést végeztem. Integrációs, egység (*unit*) teszteket, amik a projekt részét képezik, valamint Postman teszteket készítettem a használt végpontokhoz.



8. kép – Tesztek elhelyezkedése a kódbázisban

Az integrációs teszteknel csak azt vizsgáltam, hogy megfelelően elindul-e az alkalmazás és fel tudja-e állítani a kontextust, valamint egy **DI** (Dependency Injection) tesztet végeztem.



```
1 package com.library;
2
3 import ...
4
11
12 @SpringBootTest
13 @ActiveProfiles("test")
14 @NewClassNamingConvention/
15 public class LibraryApplicationIT {
16
17     @MockBean
18     private LoginController loginController;
19
20     @Test
21     @DisplayName("Application starting test")
22     void contextLoads() {
23     }
24
25     @Test
26     @DisplayName("Spring Dependency Injection test")
27     public void loginControllerLoad() { assertThat(loginController).isNotNull(); }
28
29
30
31 }
```

9. kép – Spring integrációs teszt

A tesztelés során főként egységteszteket (*Unit*) készítettem – azok közül néhányat a **TDD** (*Test-Driven Development*) metodika alapján - és ezekkel biztosítottam az üzleti logikát. A szükséges függőségeket a Mockito `@Mock` annotációjával mockoltam ki.

A unit tesztek készítésénél a given-when-then elvnek megfelelően strukturáltam a kódot. A *verify* tesztelési módszerrel és *ArgumentCaptor*-ok segítségével vizsgáltam, hogy biztosan lefut-e az adott metódus és biztosan az adott értékkel és annyszor lett-e meghívva, mint amennyi az elvárt (alap esetben 1).



```
258 @Test
259 void findUserByUserName_withUsername_returnUser() {
260     final var username = "admin";
261     final Predicate<User> adminPred = user -> user.getUsername().equals(username);
262     final var adminUser : User = filterUser(adminPred).get(0);
263
264     when(userRepository.findUserByUsername(username)).thenReturn(Optional.of(adminUser));
265
266     User user = userService.findUserByUserName(username);
267
268     verify(userRepository).findUserByUsername(usernameCaptor.capture());
269     assertTrue(nonNull(user));
270     assertEquals(username, user.getUsername());
271     assertEquals(username, usernameCaptor.getValue());
272     assertTrue(user.getUserType().getUserType().isAdmin());
273 }
```

10. kép – `findByUserName` metódus unit tesztelése

A tesztek minden build folyamatnál lefutnak - de természetesen manuálisan is indíthatóak -, így biztosítják, hogy a kód már működő funkcionalitása nem tört meg.

```
[INFO] Tests run: 16, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.987 s - in com.library.service.UserServiceTest
[INFO] Running com.library.util.PasswordUtilTest
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.186 s - in com.library.util.PasswordUtilTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 18, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 4.552 s
[INFO] Finished at: 2022-04-26T20:40:41+02:00
[INFO] -----
```

11. kép – 18 db sikeresen lefutott unit teszt.

Run: UserServiceTest

Test Results: 753 ms

Tests passed: 16 of 16 tests - 753 ms

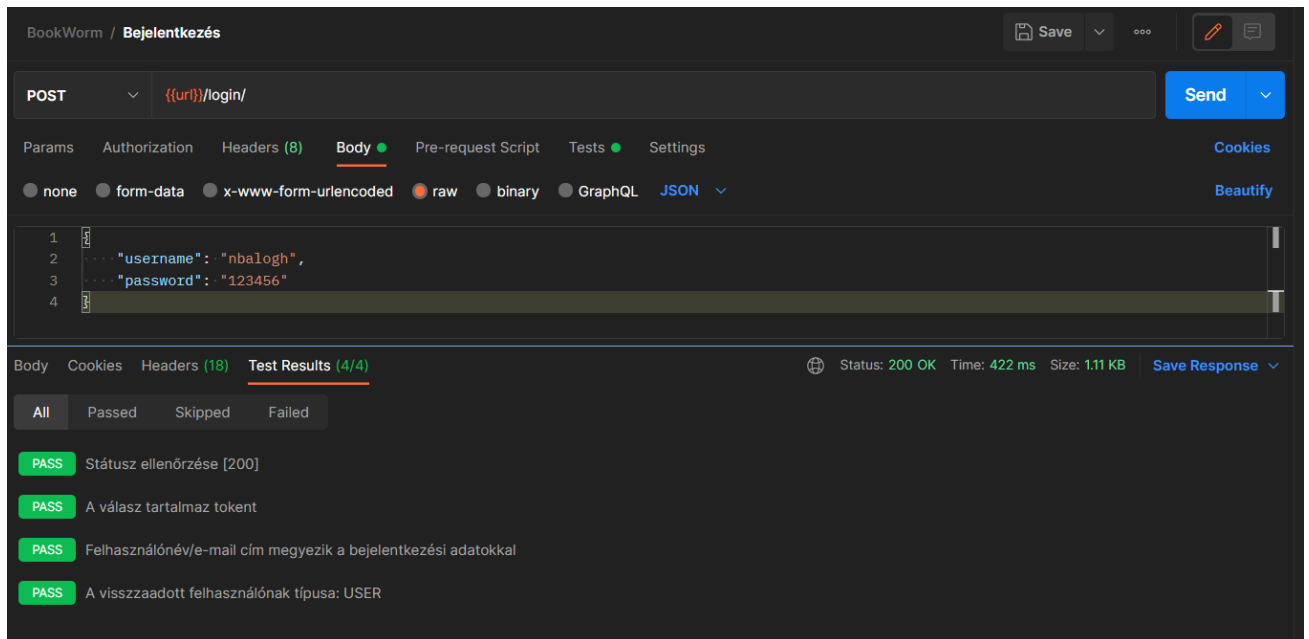
Process finished with exit code 0

12. kép – UserServiceTest osztály tesztjeinek eredménye

Postman tesztelésnél nem csak manuálisan ellenőriztem az adott kérésre érkező választ, hanem kihasználva az eszköz adta lehetőségeket teszt scripteket is készítettem, amelyek ellenőrzik, hogy a válasz az elvárt státuszkóddal, típussal tér-e vissza.

Method	Endpoint	Status	Response
GET	Könyvek listázása	2/0	Success
GET	Könyv keresése Id alapján	3/0	Success
GET	Könyv kategóriák listázása	3/0	Success
GET	Könyvtár könyveinek a száma	3/0	Success
GET	Vonalkód lekérdezése	2/0	Success
POST	Jelszó megváltoztatása	1/0	Success
GET	Felhasználó adatai	5/0	Success
GET	Felhasználó kölcsönzése	2/0	Success
GET	Kölcsönzés by Id	2/0	Success
PUT	Kölcsönzés hosszabítása	1/0	Success
POST	Bejelentkezés	4/0	Success
POST	Efelejtett jelszó	1/0	Success
POST	Kapcsolat - email	2/0	Success

13. kép – Postman collection futási eredménye



14. kép – Bejelentkezés tesztelése Postmanben

A bejelentkezési funkcióhoz készített teszt script a 2-es számú mellékletben található meg. Sikeres bejelentkezést követően a válasz fejlécében érkezik egy JWT, valamint magát a bejelentkezett felhasználó adatait is megkapja a kliens.

A szerver felé küldött kérés struktúrája:

```
{
  "username": "nbalogh",
  "password": "123456"
}
```

A rá érkező válasz struktúrája pedig a következő:

```
{
  "first_name": "Norbert",
  "last_name": "Balogh",
  "email": "balogh.norbert192@gmail.com",
  "username": "nbalogh",
  "user_id": 1,
  "full_name": "Balogh Norbert",
  "user_type": {
    "value": "USER",
    "id": 3
  }
}
```

A vizsgaremek backend/tests/postman mappájában elérhető az exportált Postman collection és a hozzá tartozó környezeti változók is. A kollekció további teszteket is tartalmaz.

## 4.2. Mobil alkalmazás (Balogh Norbert)

### 4.2.1. Technológiai háttér

A mobil alkalmazás fejlesztéséhez a **Visual Studio Code** fejlesztőkörnyezetet használtam, amely hatalmas plugin készletének köszönhetően kiváló eszköznek bizonyult.

Az alkalmazás elkészítéséhez **Ionic** keretrendszert (6.18.1) használtam, amiben **TypeScript** nyelven készítettem a **React** típusú programomat. Maga a keretrendszer lehetőséget ad platform független mobil alkalmazások készítésére. A fejlesztéshez elengedhetetlen a **node** (16.13.1-es verziót használtam) és a **npm** (8.1.2-es verziót használtam) megléte.

### 4.2.2 Megvalósítás

Az alkalmazást strukturáltan építettem fel, hogy könnyen áttekinthető és karbantartható legyen (lásd 3. melléklet). Külön szeparáltan helyezkednek ez az egyes felületek, a kisebb komponensek, amelyek csak egy részét képezik egy felületnek és a service réteg is.

Az alkalmazás útvonal irányítását (rooting) úgy oldottam meg, hogy a kezdőképernyő, ami a bejelentkező felület (/ és /login útvonal) és az elfelejtett jelszó (/forgotten/password) nincs védve. Tehát ezeket bárki elérhető bejelentkezés nélkül is. Az összes többi már védett URL-nek tekintetem. Ezeknél szükséges a bejelentkezés, amennyiben nincs bejelentkezve az adott felhasználó (nincs érvényes JWT elmentve a local storage-ben a böngészőben) akkor az alkalmazás automatikusan visszairányít a bejelentkező oldalra.

```
1 import React, {useContext, useEffect} from "react"
2 import {Redirect, Route} from "react-router";
3 import {AuthContext} from "../context";
4 import {WebAPI} from "../services/WebAPI";
5
6 interface ProtectedRouteProps {
7   component: React.ComponentType<any>;
8   path: string;
9 }
10
11 export const ProtectedRoute: React.FC<ProtectedRouteProps> = ({component: Component, path}) => {
12   const context = useContext(AuthContext);
13
14   useEffect(() => {
15     }, [Component, path]);
16
17   const hasToken = (): boolean => {
18     const token = WebAPI.storageService.getToken();
19     return (token !== null && token !== "");
20   }
21
22   context.isAuthenticated = hasToken();
23   return (
24     <Route path={path}
25       render={props => context.isAuthenticated ? <Component {...props} /> : <Redirect to="/login"/>} />
26   );
27 };
```

15. kép – protected rootért felelős kód

```

36  setupIonicReact();
37
38  const App: React.FC = () => {
39    return (
40      <IonApp>
41        <IonReactRouter>
42          <IonSplitPane contentId="main">
43            <Menu/>
44            <IonRouterOutlet id="main">
45              <Route path="/" exact={true}>
46                <Redirect to="/login"/>
47              </Route>
48              <Route path="/login" exact={true}>
49                <LoginPage/>
50              </Route>
51              <Route path="/forgotten/password" exact={true}>
52                <ForgottenPasswordPage/>
53              </Route>
54              <ProtectedRoute path="/profile" component={ProfilePage}/>
55              <ProtectedRoute path="/modify" component={ModifyPersonalDetailPage}/>
56              <ProtectedRoute path="/password/change" component={PasswordChangePage}/>
57              <ProtectedRoute path="/books" component={BookPage}/>
58              <ProtectedRoute path="/contact" component={ContactPage}/>
59              <ProtectedRoute path="/loans" component={LoanPage}/>
60              <ProtectedRoute path="/history" component={LoanPage}/>
61              <ProtectedRoute path="/book/detail/:id" component={BookDetailPage}/>
62            </IonRouterOutlet>
63          </IonSplitPane>
64        </IonReactRouter>
65      </IonApp>
66    );
67  };
68
69  export default App;

```

16. kép – Routing beállítása

Az applikáció a kapcsolati adatok kivételével mindent a Spring boot backend alkalmazás által biztosított REST API-ok hívásán keresztül olvas, ír az adatbázisba. Ezekért a funkciókért az egyes servicek felelősek ( login.service.ts, book.service.ts, login.service.ts, user.service.ts)

```

passwordReset = async (email: string): Promise<boolean | Error> => {
  const data = {username: email, password: undefined};
  const options = {
    method: 'POST',
    headers: {
      'Content-Type': APPLICATION_JSON
    },
    body: JSON.stringify(data)
  } as RequestInit

  return await fetch(this.baseUrl + '/password/reset', options)
    .then(response => response.status === 200)
    .catch(ex => new Error("Fetch fail!"));
}

private setToken(token: string | undefined) {
  this.storageService.setToken(token);
}
}

```

17. kép – Új jelszó igénylésért felelős metódus

A mobil alkalmazás tesztelése során főként manuális tesztelést alkalmaztam, amihez több böngészőt is használtam.

### 4.3. Webes alkalmazás

#### 4.3.1 Technológiai háttér

A webalkalmazás fejlesztéséhez szintén a **Visual Studio Code** fejlesztőkörnyezetet használtuk, úgy mint a mobil alkalmazás esetén.

Az alkalmazás elkészítéséhez **VueJs** keretrendszert használtunk, amiben **HTML**, **CSS** és **JavaScript** kódokat írtunk. Az általunk készített személyre szabott **CSS** osztályok mellett a Bootstrap által biztosított stílusokat is alkalmaztunk, ezzel garantálva az oldal reszponzivitákat.

Az alkalmazás szintén a JAVA nyelven íródott backenddel kommunikál és rajta keresztül éri el az adatbázisban tárolt könyveket. Ezeket a hívásokat az **axios** függvény segítségével valósítottuk meg.

#### 4.3.2 Megvalósítás

A webalkalmazás fő célja egy publikusan is elérhető felület, ahol a felhasználók megtekinthetik a könyvtárban található könyveket. Az oldalt megvalósítása során a hagyományos és a **SPA** (*Single Page Application*) weboldalak elveit ötvöztük. Lehetőség van navigálni az adott szekciók között, de regisztrálni is.

Igyekeztünk az oldalt teljesen reszponzívra fejleszteni, hogy minden felhasználó eszközön a lehető legjobb élményt nyújtsa az oldal használata. Ehhez a Bootstrap által biztosított osztályokat és általunk készített mediaQueryket is használtunk.



## ÖNNEK AJÁNLJUK

```
174 @media (max-width: 768px) {  
175   .navbar {  
176     opacity: 0.95;  
177   }  
178  
179   .navbar-container input[type="checkbox"],  
180   .navbar-container .hamburger-lines {  
181     display: block;  
182   }  
183  
184   .navbar-container {  
185     display: block;  
186     position: relative;  
187     height: 64px;  
188   }  
189  
190   .navbar-container input[type="checkbox"] {  
191     position: absolute;  
192     display: block;  
193     height: 32px;  
194     width: 38px;  
195     top: 28px;  
196     left: 28px;  
197     z-index: 5;  
198     opacity: 0;  
199   }  
200 }
```



18.-19. kép – Saját készítésű mediaQuery ( bal oldalon) és a megjelenítés 450x720px felbontás esetén

Új felhasználó regisztrációját követően a megadott e-mail címre kiküld a backend egy üdvözlő e-mailt az ideiglenes jelszóval, amellyel be tud lépni a mobil alkalmazásba.

A Rólunk mondták szekció az alkalmazásban ideiglenesen eltárolt 10 db véleményből véletlenszerűen kiválaszt 3 db-ot és azt jeleníti meg.

### Rólunk mondták

**Boldizsár Bonifác**  
★★★★★  
Korszerű, gyorsan tanulható és folyamatosan fejlődő. Szóval egyszerűen csak jó...

**Oszkár**  
★★★★★  
Nálunk már a könyvtárban mindenki ezt használja, mert annyira egyszerű.

**Kiss Balázs**  
★★★★★  
Nemrégiben kezdtem el használni az alkalmazást és nagyon megkönnyítette a könyvtári könyvek kezelését.

20. kép – véletlenszerű vélemények

A főoldalon található kategóriák az backend hívás során az adatbázisból kiolvasott és visszaadott könyvtári könyv típusokból választ ki véletlenszerűen 3db-ot, majd azokra kattintva



át navigál a könyvek oldalra, ahol csak az adott kategóriának megfelelő szűrési feltétellel indít keres a backend felé, majd a visszakapott könyveket jeleníti meg.

A programkód írása során igyekeztünk a komponenseket külön kiszervezni, hogy a későbbiekben újra használhatóak legyenek.

```
<template>
  <section id="showcase" class="">
    <div class="showcase-container">
      <h1 id="home" class="main-title">Aktuális kínálatunk</h1>
    </div>
  </section>

  <div class="container">
    <div class="row">
      <BookComponent v-for="book in books" :key="book.book_id" :book="book"/>
    </div>
  </div>
</template>
```

21. kép – Könyvek listázása saját komponens használatával

```
src > components > BookComponent.vue > {} "BookComponent.vue" > template
1 <template>
2 <router-link :to="{ name: 'Details', params: { id: book.book_id } }">
3   <div class="event-link">
4     <div class="event-card">
5       <div class="card-body">
6         <h3 class="card-title">{{ book.title }}</h3>
7         <div class="status">
8           <span v-if="statusColorR" class="status-icon-r"></span>
9           <span v-else-if="statusColorG" class="status-icon-g"></span>
10          <span>{{ statusColorR ? 'Csak helyben olvasható' : 'Kölcsönözhető' }}</span>
11        </div>
12        
13      </div>
14    </div>
15  </div>
16 </router-link>
17 </template>
```

22. kép – BookComponent.vue felépítése

Az egyes komponensek közötti navigációt a **VueJs** keretrendszer Routere tette lehetővé, ami az index.js fájlban található.

#### 4.3.3 Tesztelés

A weboldal tesztelése főként manuális tesztelésből állt különböző böngészőkben, de készítettünk hozzá néhány unit tesztet is.

A regisztrációs felület kitöltésénél készítettünk több negatív esetet lefedő unit tesztet is, amelyek azt hivatottak ellenőrizni, hogy az adott kötelező mezők kitöltése esetén ténylegesen megjelenik-e a figyelmeztető üzenet.

```
test('A regisztráció kitöltése - hiányzó név', async () => {
  const wrapper = shallowMount(Registration)

  let nameError = wrapper.find('label[id="lastnameError"]')
  expect(nameError.exists()).toBe(false)

  const email = wrapper.find('input[id="lastname"]')
  await email.setValue("");
  await email.trigger('change')

  nameError = wrapper.find('label[id="lastnameError"]')
  expect(nameError.exists()).toBe(true)
  expect(nameError.text()).toBe("A vezetéknév nem megfelelő formátumú!")
})
```

23. kép – Regisztrációs űrlap hiányos kitöltésére szolgáló teszt

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

> filmek@0.1.0 test:unit
> vue-cli-service test:unit

PASS tests/unit/registration.spec.js
PASS tests/unit/home.spec.js
PASS tests/unit/books.spec.js

Test Suites: 3 passed, 3 total
Tests: 11 passed, 11 total
Snapshots: 0 total
Time: 6.093s
Ran all test suites.
```

24.kép - Sikeresen lefutó tesztek

## 4.4. Asztali alkalmazás (Tápai Nándor)

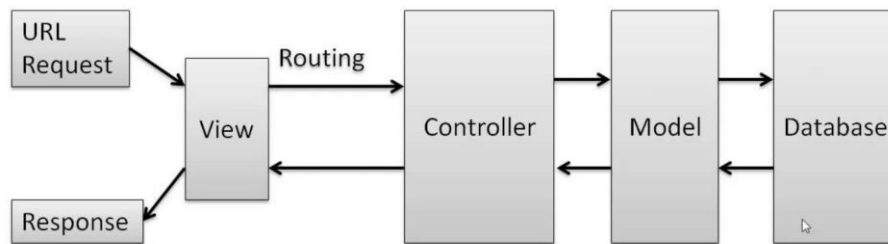
### 4.4.1 Technológiai háttér

Az asztali **WPF** (*Windows Presentation Foundation*) alkalmazás elkészítéséhez a **Microsoft Visual Studio 2019** fejlesztői környezetet és C# nyelvet használtam, amely főként .NET projektekre van specializálva. A **WPF** project felhasználói felületéért (*UI – User Interface*) felelős nyelve a **XAML** (*eXtensible Application Markup Language*) leíró nyelv.

### 4.4.2. Megvalósítás

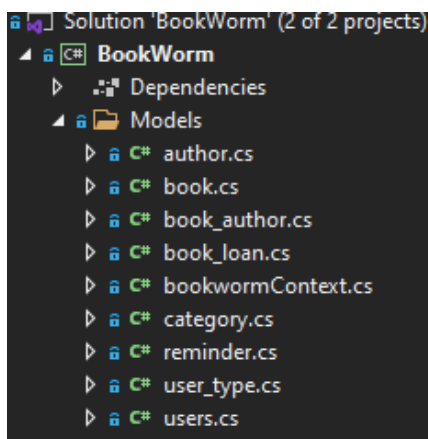
Az alkalmazás tervezésénél figyeltem az OOP (Objektum orientált programozás) paradigmáinak követésére, ezért a megvalósítás során is az MVC modell követésére törekedtem, hogy jól elkülöníthető legyenek egymástól a felületért felelős UI részek, az üzleti logika és az adatbázis entitásokat reprezentáló osztályok.

## MVC Design Pattern – ASP.Net



4. ábra – MVC modell

A modell réteg reprezentálja az adatbázis egyes tábláit **C#** kódnak megfelelően. Itt vannak definiálva az adattípusok, a kulcsok és az adatbázis szinten létrehozott megszorítások is, hogy például az adott mező kötelező-e vagy egy szöveges típusú változó esetén mekkora lehet a maximális mérete.



```
[Index(nameof(loanable), Name = "loanable")]
[Index(nameof(title), Name = "title")]
15 references
public partial class book
{
    1 reference
    public book()
    {
        book_author = new HashSet<book_author>();
        book_loan = new HashSet<book_loan>();
    }

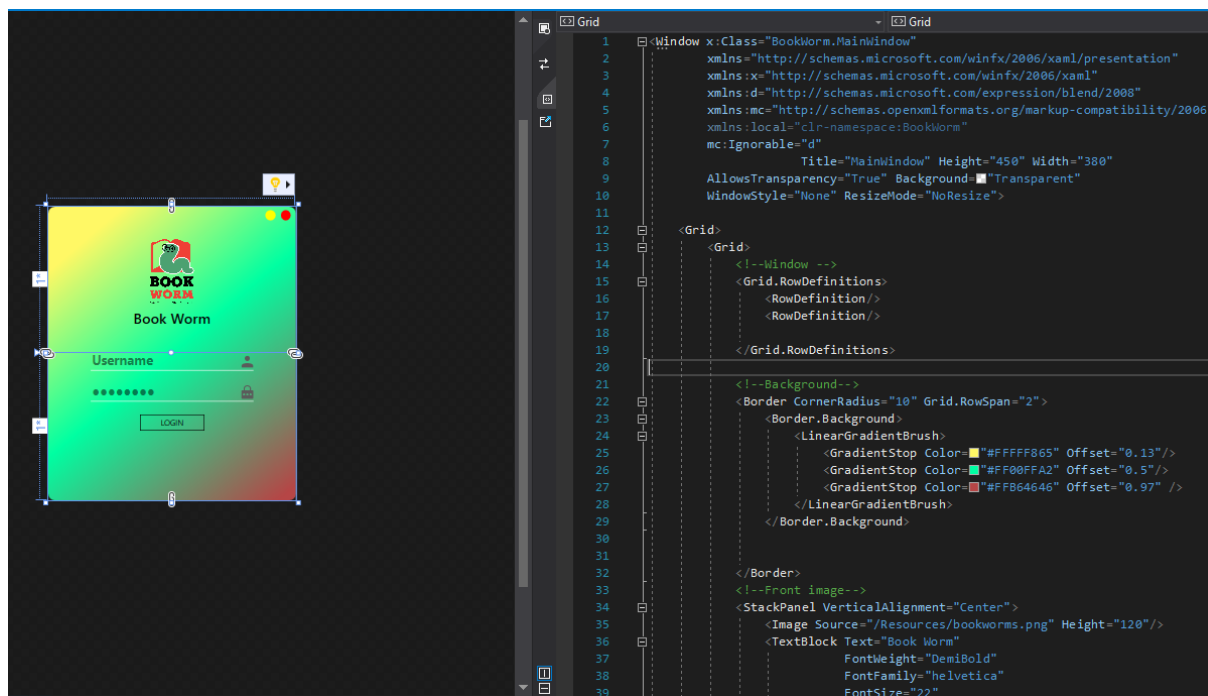
    [Key]
    [Column(TypeName = "int(11)")]
    3 references
    public int id { get; set; }
    [Required]
    [StringLength(200)]
    2 references
    public string title { get; set; }
    [StringLength(50)]
    1 reference
    public string isbn { get; set; }
    [StringLength(500)]
    2 references
    public string img_link { get; set; }
    [StringLength(2000)]
    1 reference
    public string short_description { get; set; }
```

25.-26.. kép – Adatbázis tábláknak megfelelő C# osztályok (bal oldalon). Egy adott osztály szerkezetének részlete (jobb oldalon)

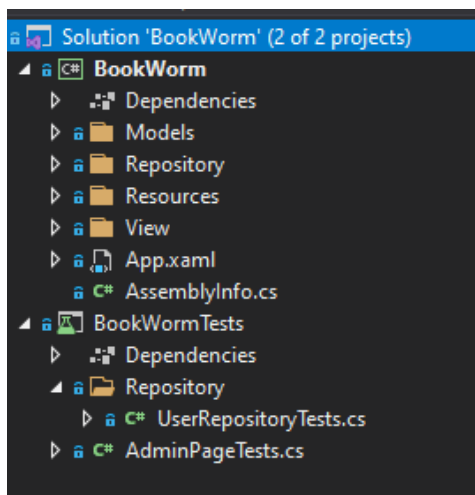
Az adatbázis műveletekért a Repository réteg felelős, amely minden entitásnak tartalmaz egy saját repositoryt. Ezek a repository-k a bookwormContext használatával végznek **CRUD** műveleteket az adott táblákon. A bookwormContext egy DbContext kiterjesztés ami az adatbázis kapcsolatért és az azt érintő műveletekért felelős.

A megjelenítésért a View vagy nézet réteg felelős, amely a **XAML** leíró nyelv segítségével könnyedén formálható. Ebben a mappában található az összes felületi megjelenítésért felelős osztály. A program során több képet is felhasználtam, hogy növeljem a felhasználói élményt.

Ezeket a képeket a Resources mappában helyeztem el, így az alkalmazásom gond nélkül el tudja érni és használni tudja ezeket az erőforrásokat.



27. kép – Bejelentkezési felület grafikusán (bal oldalon) és a XAML kódja (jobb oldalon)



28.kép- Az alkalmazás struktúrája

#### 4.4.3. Tesztelés

Az asztali alkalmazás unit tesztjeit a Visual Stúdió 2019 segítségével készítettem és a beépített Test Explorer segítségével futtattam és ellenőriztem.

A *InsertUserTest* metódussal ellenőriztem, hogy a ténylegesen sikerült-e az adott felhasználó beszúrása.

```
[TestMethod()]
public void InsertUserTest()
{
    UserRepository userRepository = new UserRepository();

    int numberOfUsersInDb = userRepository.CountUser();

    users user = new users();
    user.id = numberOfUsersInDb + 1;
    user.firstname = "Elek";
    user.lastname = "Test";
    user.email = "test@gmail.com";
    user.username = "test_elek";
    user.user_type_id = 3;
    user.created_date = DateTime.Now;
    user.created_by = "test";

    userRepository.Insert(user);

    var newCount = userRepository.CountUser();
    Assert.IsTrue(numberOfUsersInDb < newCount, "Nem sikerült a beszúrás");

    var users = userRepository.GetAllSimpleUser();
    var userList = users.Where(u => u.email.Equals(user.email)).ToList();

    Assert.IsFalse(userList.Count == 0, "Nem szerepel ilyen email címmel user a db-ben");
    Assert.IsTrue(userList[0].username.Equals(user.username), "Nem ugyanaz a usernem");
}
```

29.kép – felhasználó beszúrása teszt

A **Test Explorer** segítségével könnyedén futtathattam a teszteket, ahol gyors visszajelzést is kaptam a sikerességükről.

Test Explorer					
Test	Duration	Traits	Error Message	Test Detail Summary	
BookWormTests (3)	1.1 sec			AdminPageTest	
BookWorm.Repository.Tests (2)	1.1 sec			Source: AdminPageTests.cs line 15	
UserRepositoryTests (2)	1.1 sec			Duration: < 1 ms	
GetAllLibrarianTest	888 ms				
InsertUserTest	171 ms				
BookWorm.Tests (1)	< 1 ms				
AdminPageTests (1)	< 1 ms				
AdminPageTest	< 1 ms				

30.kép – a tesztek eredménye

Az alkalmazás további funkcióit és a felhasználói élményt manuálisan teszteltem végig. A tesztelés során készített jegyzőkönyveket a 5. és a 6. számú melléklet tartalmazza.

Mivel az asztali alkalmazás egy belső használatra szánt kizárólag adminok és könyvtárosok által elérhető funkciókat tartalmaz ezért fontos, hogy a használatát adatbázisban tárolt felhasználónév jelszó sikeres egyezősége esetén kezdheti meg az adott illető.

Az alkalmazás indításánál egy bejelentkezési ablak jelenik meg, ahol a felhasználói jogosultság alapján dől el, hogy beléphet-e az alkalmazásba, vagy sem. Amennyiben a felhasználó, aki belépni kíván, nem rendelkezik admin vagy könyvtáros jogosultsággal, nincs lehetősége az asztali alkalmazásba történő belépésre. A sikeres belépést követően az alkalmazás a jogosultságnak megfelelő felületet jeleníti meg.

## 5. FELHASZNÁLÓI DOKUMENTÁCIÓ

### 5.1. Asztali alkalmazás (Tápai Nándor)

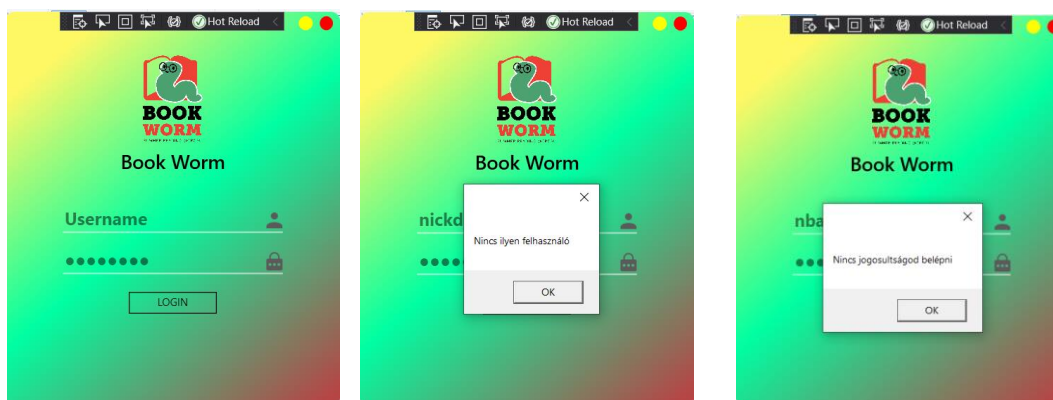
Az asztali alkalmazás .NET keretrendszerben készült, amit az újabb Windows operációs rendszerek (Windows 10 és Windows 11) alapértelmezetten tartalmaznak. Az asztali alkalmazás használatához szükségünk van egy adatbázisra is. Az adatbázis létrehozásához szüksége *bookworm\_db.sql* fájl a *database* mappában található. Ehhez szükséges egy MySQL kiszolgálóval (pl.: XAMPP) rendelkezni és importálni az adatbázist létrehozó scriptet. *Egyéb megoldásként a 5.4-es pontban leírt docker compose használata*

A program *desktop\_application/BookWorm* mappában elhelyezett *setup.exe* fájlal telepíthető, majd telepítést követően a **BookWorm.exe** fájlal indítható. Természetesen a projekt a Visual Stúdió fejlesztőkörnyezetből is elindítható.

A program indulását követően egy bejelentkező ablak jelenik meg. Innen csak sikeres bejelentkezést követően tudunk továbblépni a könyvtárosok/adminok. Az alkalmazás az adott felhasználó jogának megfelelő felületet fogja megnyitni.

Sikeres bejelentkezést követően az alkalmazás eltárol néhány adatot a memóriájában az aktuálisan bejelentkezett felhasználóról. Így egy könyvtáros később sem fogja elérni az admin funkciókat.

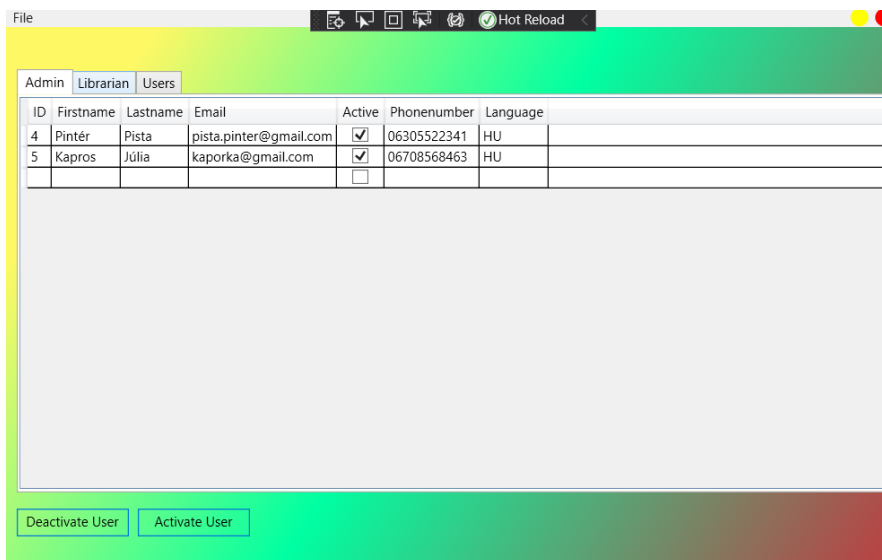
Téves felhasználónév, jelszó páros esetén a bejelentkezés sikertelen és az alkalmazás hibüzenet dob.



31. – 33. kép – Bejelentkező felület

Ha admin jelentkezik be, akkor az alkalmazás egyből az admin oldalt jeleníti meg. Itt az adminoknak lehetőségük van minden típusú felhasználó között keresni (*admin, könyvtáros, „egyszerű felhasználó”*). Ezeken a fülökön az adminok képesek inaktíválni az egyes felhasználókat. Innentől kezdve ők nem tudnak belépni az alkalmazásokba. Az adminok a

könyvtárosoknak készült funkciókat is elérik és bármikor át tudnak lépni a File menüpont alatt a könyvtárosok oldalára.



34. kép – admin felület

Az adminnal ellentétben a könyvtárosként belépett felhasználók csak sima felhasználókat tudnak felvenni a rendszerbe. Csökkentve ezzel a hibázási lehetőséget.

Ha egy könyvtáros lép be, akkor egyből a főmenübe kerül (nem tudja elérni az admin oldalt) és a következő funkciók közül választhat.



35. kép – Könyvtáros menü



## 5.2. Webalkalmazás

A weboldal lehetőséget ad a felhasználóknak, hogy megtekintsék a könyvtárban lévő könyveket, azokra kattintva részletesebb információt kapjanak róluk.



36.kép – webalkalmazás navigációs menü

A rólunk mondták funkció jelenleg még csak szemléltető jelleggel előre felvette véleményekből véletlenszerűen jelenít meg hármat. Későbbiekben ezt szeretnék majd lecserélni valós adatbázisban tárolt véleményekre.



37.kép – Vélemények megjelenítése

Az Önnek ajánljuk szekcióban az adatbázisban tárolt kategóriák közül jelenítünk meg véletlenszerűen hármat, majd, ha a felhasználó a neki szimpatikusra kattint akkor az adott típusnak megfelelő könyveket listázza ki a könyvtárból.



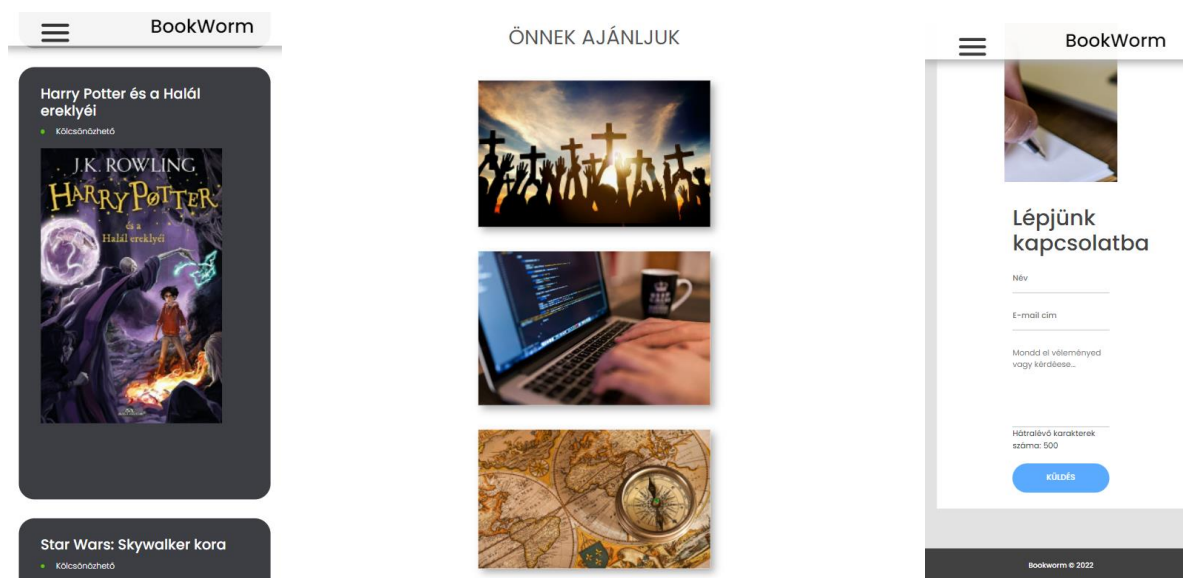
## ÖNNEK AJÁNLJUK



38. kép – A felhasználónak ajánlott kategóriák

Az oldalon lehetőség van regisztrálni a rendszerbe *(amennyiben még az adott e-mail cím nem foglalt)* és a regisztrációról egy megerősítő e-mail megy ki, amely tartalmazza az ideiglenesen generált jelszót is, amivel már bele lehet lépni a mobil applikációba.

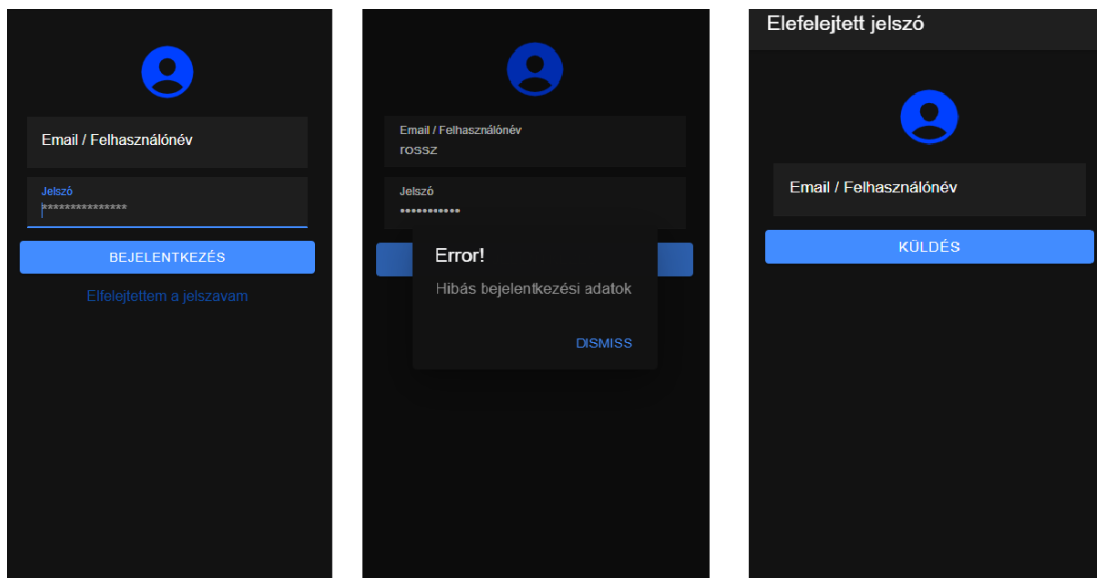
Az alkalmazás elkészítésénél elsődleges szempont volt az esztétikus megjelenés és a reszponzív megjelenítés, hogy minden kliens eszközön a lehető legjobb élményt kapja a felhasználó.



39-40-41. kép - Reszponzív megjelenés szemléltetése

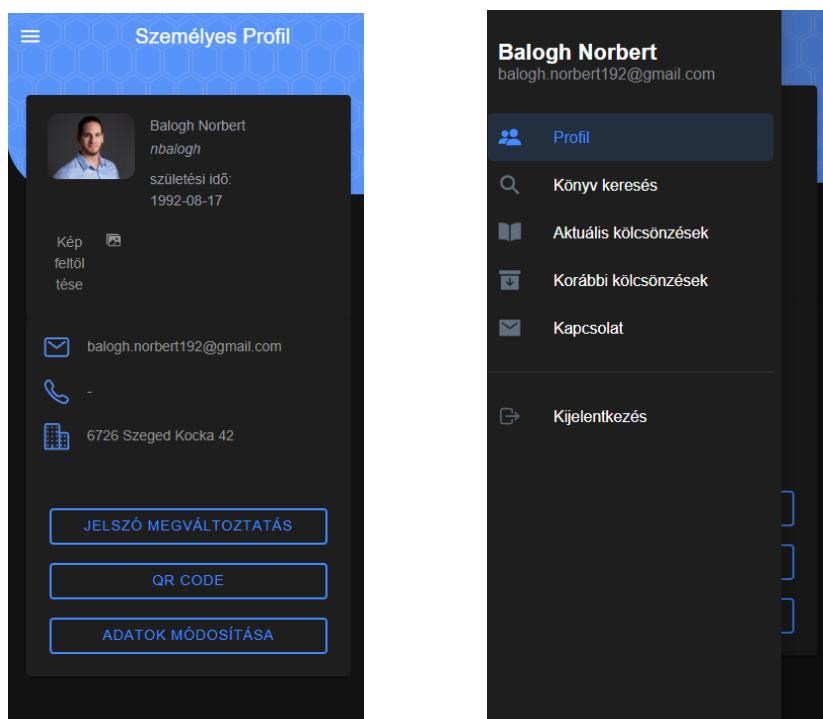
### 5.3. Mobil applikáció (Balogh Norbert)

Az applikáció célja, hogy a könyvát használó felhasználók követni tudják kölcsönzéseiket, - aktuálisan mi van náluk, mikor jár le, korábban miket kölcsönöztek ki – valamint a könyv visszavitele során segítséget nyújtson a könyvtárosnak. Meg tudják még tekinteni a könyvtárban szereplő könyveket és van lehetőségük saját adataik megtekintésére és módosítására is.




42-43-44. kép – Mobil applikáció bejelentkező és elfelejtett jelszó felülete

Az applikáció csak bejelentkezést követően használható. Ha valaki elfelejtette a jelszavát, akkor van lehetősége egy új automatikusan generált jelszó kérésére, amit e-mailben ki is küld neki a rendszer a megadott e-mail címre. Természetesen belépést követően ezt van lehetősége lecserélni.



45-46.kép – Profil oldal és a menü




Aktuális jelszó

Új jelszó

Megerősítő jelszó

VISSZA CSERE



Vezetéknév  
Balogh

Keresztnév  
Norbert

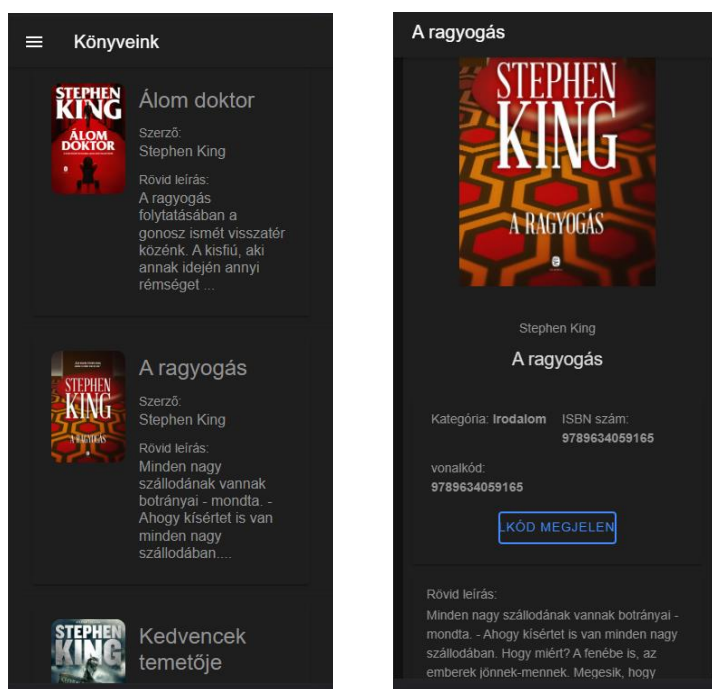
E-mail  
balogh.norbert192@gmail.com

Telefonszám

VISSZA MENTÉS

47-48. kép – Jelszó csere és adatmódosítás

A könyvek listázásánál paginget, azaz lapozást használtam, ahol egyszerre csak az első 10 elem jelenik meg, minimális információkkal. A további könyveket a görgetés hatására kéri le az alkalmazás a backendtől. Amennyiben a felhasználó rákattint az egyik könyvre akkor az alkalmazás betölti a könyv részletes információit.

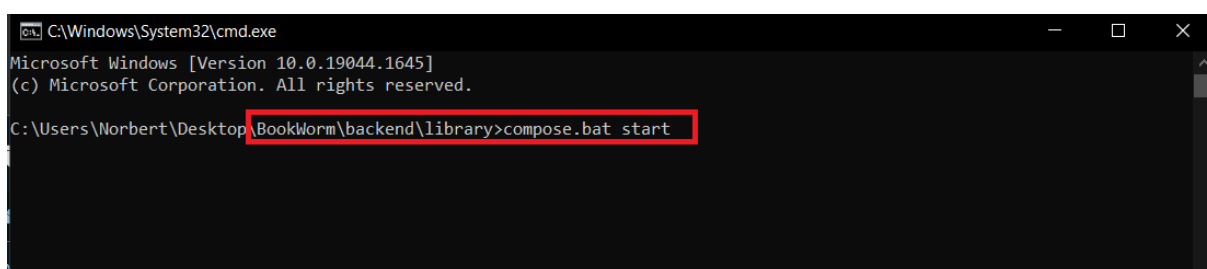


49-50.kép – Könyvek megjelenítése

## 5.4. Docker-compose (Balogh Norbert)

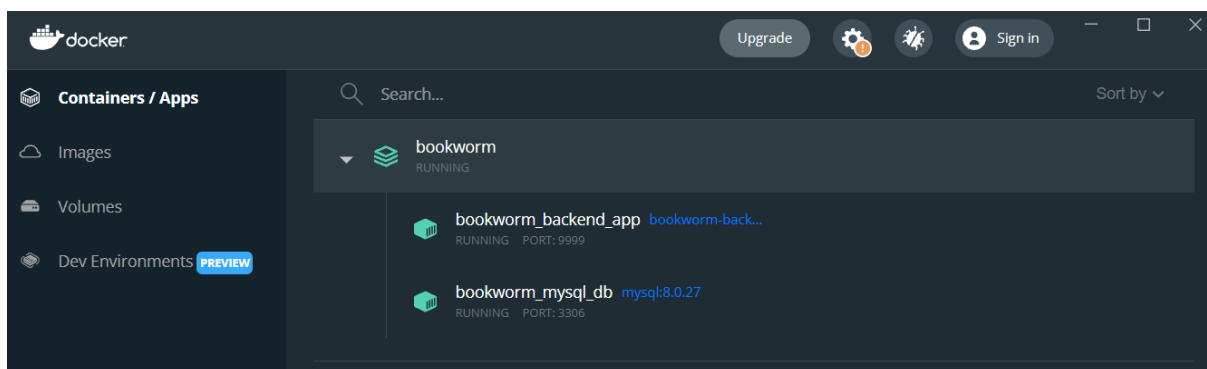
Az adatbázis és a backend szerver elindítható egy docker konténer használatával is. Ebben az esetben nincs szükség sem a JAVA nyelven készült Spring boot alkalmazás számára sem az adatbáziskezelő réteg számára külön futtató környezetre, ezeket a Docker biztosítja.

A számítógépünkön csak egy Docker host megléte szükséges (*Windows esetén Docker Desktop*), amit könnyedén telepíteni tudunk a <https://docs.docker.com/get-docker/> oldalról. Ezt követően csak arra van szükségünk, hogy parancssorban elindítsuk a backend/library mappában lévő compose.bat fájlunkat start paraméterrel.



*x. kép – compose.bat futtatása*

Ezt követően amikor a compose futása véget ért, akkor a Docker Desktopban megjelenik az adatbázis és a backend applikáció is, amelyek pár másodperc múlva már használatra készen állnak.



*x.kép – Docker Desktopban futó alkalmazás és adatbázis*

## 6. ÖSSZEGZÉS

Vizsgamunkánk célja egy modern könyvtári nyilvántartó rendszer megtervezése és annak megvalósítása volt, amely több különböző komponensből tevődik össze.

A webes alkalmazás betekintés nyújt a felhasználóknak a könyvtári könyvek listájára és kapcsolatot tud létesíteni a felhasználó az üzemeltetőkkel, jelen esetben velünk. A tervek szerint ez a felület még rengeteg fejlesztésre szorul. A jövőben szeretnénk elérni, hogy ne csak regisztrálni lehessen a weboldalon, hanem minden olyan funkciót (bejelentkezés, adatok módosítása, korábbi és aktuális kölcsönzések megtekintése) és még számos más új funkciót (például a könyv előjegyzést) is használni tudjanak a felhasználók, amit a mobil alkalmazás már biztosít számukra.

Úgy érezzük, hogy az alkalmazást akkor lehetne teljesen késznek tekinteni, amennyiben weben és mobilon is közel ugyanazok a funkciók zavartalanul elérhetőek. Terveink között szerepel, hogy XY felhasználó - aki már letöltötte a mobil alkalmazást- bemegy a könyvtárba egy könyvet kölcsönözni és elegendő felmutatni a telefonját, mert az ott látható QR kódot az könyvtáros egy vonalkód olvasóval vagy akár a mobil alkalmazás könyvtáros funkciójával beolvassa és már intézi is a könyv kölcsönzését.

Összességében úgy éreztük, hogy a project során jól sikerült összedolgozni, hol személyesen pair-programing keretei között, hol online. A kezdeti problémák ellenére sikerült egy működőképes rendszer összeraknunk. A projectfeladat megvalósítása során a legnagyobb problémát az idő okozta, főként, hogy mi a három fős csapatokkal szemben emberhátrányból indultunk, aminek sokszor – főként a projekttől független iskolai beadandók kapcsán – is éreztük a hatását.

## FORRÁSJEGYZÉK

<https://docs.docker.com/get-docker/>  
<https://hevodata.com/learn/rest-api-best-practices/>  
<https://www.baeldung.com/spring-rest-docs>  
<https://learning.postman.com/docs/getting-started/introduction/>  
<https://www.youtube.com/watch?v=pn7IPCuliZs>  
<https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>  
<https://ionicframework.com/docs/>  
<https://www.typescriptlang.org/docs/handbook/intro.html>  
<https://vuejs.org/guide/introduction.html>  
<https://css-tricks.com/>  
<https://www.tutorialsteacher.com/mvc/mvc-model>

# MELLÉKLETEK

**Bejelentkezés** Login Controller Impl

**POST** /library/api/login Bejelentkezés

JWT-t ad vissza a response AUTHORIZATION headerben

Parameters

Try it out

Name	Description
<b>login</b> <small>* required</small> object (body)	login  Example Value   Model <pre>{  "password": "string",  "username": "string"}</pre> <div>Parameter content type application/json</div>

Responses

Response content type \*/\*

Code	Description
200	OK  Example Value   Model <pre>{  "email": "string",  "firstname": "string",  "full_name": "string",  "lastname": "string",  "middle_name": "string",  "phone": "string",  "user_id": 0,  "user_type": {    "admin": true,    "id": 0,    "librarian": true,    "user": true,    "value": "string"  },  "username": "string"}</pre>
201	Created
401	Unauthorized
403	Forbidden

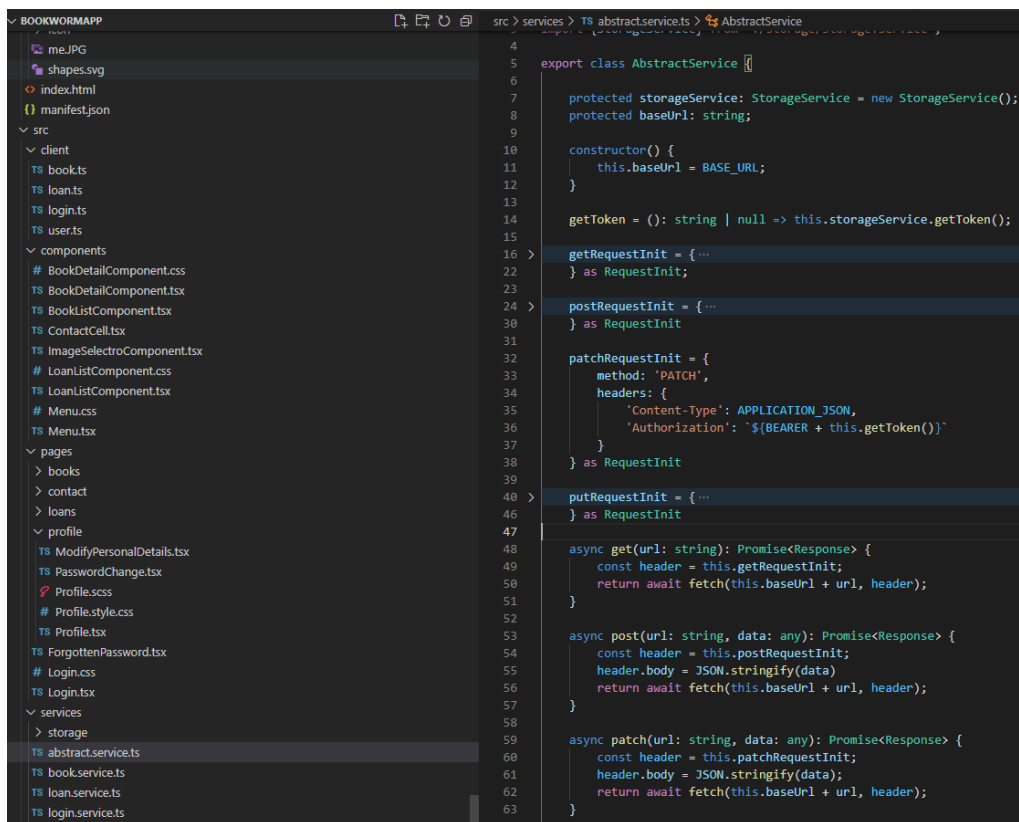
1. számú melléklet: Bejelentkezéshez használt API Swagger dokumentációja

```

1  const actual_status_code = pm.response.code;
2  const json_data = pm.response.json();
3
4  if(actual_status_code == 200){
5      pm.test("Státusz ellenőrzése [200]",
6          () => pm.expect(actual_status_code).to.be.eql(200)
7      );
8
9      const auth_header = pm.response.headers.get('Authorization');
10     pm.test(["A válasz tartalmaz token",
11         () => pm.expect(auth_header).to.be.a('string').and.not.be.empty
12     ]);
13
14     pm.environment.set('token', `${auth_header}`);
15     const has_username = pm.expect(pm.response.json()).to.have.property('username');
16     if(has_username){
17         pm.test(
18             "Felhasználónév/e-mail cím megegyezik a bejelentkezési adatokkal",
19             () => pm.expect(pm.environment.get('username')).to.be.oneOf([json_data.username, json_data.email])
20         );
21         const user_type = json_data.user_type;
22         pm.test("A visszaadott felhasználónak típusa: " + user_type.value,
23             () => pm.expect(user_type).to.be.an('object').is.not.null
24         );
25     }
26 }else if(actual_status_code == 401){
27     pm.test("Elvárt hibaüzenet ellenőrzése",
28         () => pm.expect(json_data.message).to.be.a('string').include('Unauthorized')
29     );
30 }else if(actual_status_code == 403){
31     pm.test("Elvárt hibaüzenet ellenőrzése",
32         () => pm.expect(json_data.message).to.be.a('string').include('User is blocked')
33     );
34 }

```

2. számú melléklet: Bejelentkezés végpont Postman tesztscriptje



3. számú melléklet: A mobil alkalmazás struktúrája



Üdvözöljük a BookWorm rendszerében! [Küldés](#) [Beérkező levelek x](#)



info.bookworm.library@gmail.com  
címezett: én ▾

23:26 (0 perccel ezelőtt) ☆ ↶ ⋮



Kedves Balogh Norbert!

Üdvözöljük a BookWorm rendszerében! Jó szorakozást kívánunk!

**Az Ön ideiglenesen generált jelszava: goK3lkrKCuUCRcqp**

Kérjük a következő belépés után változtassa meg!

Szívélyes üdvözléttel,  
A BookWorm csapata

8. Számú melléklet – Regisztrációs e-mailről készült képernyőkép

TESZTELÉSI JEGYZŐKÖNYV	
A tesztet leírása és célja:	Asztali alkalmazás backend oldali tesztelése
A tesztelt folyamat / funkció leírása:	Az <u>AuthorRepository</u> -ban található <u>InsertAuthor</u> módszer tesztelése, írók feltöltése adatbázisba.
A tesztelés előfeltételei:	Megfelelő adatbázis kommunikáció. Helyesen megírt <u>InsertAuthor</u> módszer.
A tesztelés dátuma és ideje:	2022.04.19
A tesztadatok típusa:	<u>InsertAuthor</u> ; publikus módszer.
A tesztet végző személy(ek):	Tápai Nándor
A tesztelt rendszer beállításai:	PC-n futó Windows környezet
A tesztet elvárt eredménye:	A módszer hívását követően a feltöltés sikeresen megtörténik
A tesztelés eredménye:	<input checked="" type="checkbox"/> megfelelt <input type="checkbox"/> nem felelt meg <input type="checkbox"/> megfelelt megjegyzésekkel
Megjegyzések:	
A tesztelési jegyzőkönyvet készítette (név, aláírás):	Tápai Nándor

9. Számú melléklet – *InsertAuthor* funkció tesztelési jegyzőkönyve

TESZTELÉSI JEGYZŐKÖNYV	
A tesztet leírása és célja:	Asztali alkalmazás frontend oldali tesztelése
A tesztelt folyamat / funkció leírása:	AddUser oldalon található <u>Upload gomb</u> <u>userAddButton click</u> esemény tesztelése
A tesztelés előfeltételei:	Megfelelő adatbázis kommunikáció. Megfelelő frontend backend kommunikáció. Helyesen megírt <u>Insert</u> metódus.
A tesztelés dátuma és ideje:	2022.04.19
A tesztadatok típusa:	<u>userAddButton: RoutedEvent</u> <u>Insert metódus: Publikus metódus</u>
A tesztet végző személy(ek):	Tápai Nándor
A tesztelt rendszer beállításai:	PC-n futó Windows környezet
A tesztet elvárt eredménye:	A gomb megnyomását követően lekérdezi a mezők adatait és sikeres behelyettesítést követően megtörténik a feltöltés.
A tesztelés eredménye:	<input checked="" type="checkbox"/> megfelelt <input type="checkbox"/> nem felelt meg <input type="checkbox"/> megfelelt megjegyzésekkel
Megjegyzések:	
A tesztelési jegyzőkönyvet készítette (név, aláírás):	Tápai Nándor

6.számú melléklet - UserAddButton funkció tesztelési jegyzőkönyve

## Hallgatói nyilatkozat

Alulírottak, **Balogh Norbert** és **Tápai Nándor** a Szegedi Szakképzési Centrum Vasvári Pál Gazdasági és Informatikai Technikum hallgatója kijelentem, hogy a **BookWorm korszerű könyvtár** című záródolgozat a saját munkánk.

Kelt: Szeged, 2022. április 25.

.....  
**Balogh Norbert**

.....  
**Tápai Nándor**