

# Könyvtér

# Fejlesztői dokumentáció

2022

Hermányi Gergely Barnabás

Szivák Gergő

# Tartalomjegyzék

<b>Ismertető.....</b>	<b>3</b>
Cél.....	3
Felhasznált technológiák.....	3
Fejlesztői környezet.....	4
Kódolási konvenciók.....	4
Munkamegosztás.....	4
<b>Adatbázis.....</b>	<b>5</b>
Tervezés és Szerkezet.....	5
Adatbázis létrehozása.....	6
<b>REST API.....</b>	<b>8</b>
UML terv.....	8
Általános működés.....	8
Végpontok.....	9
Indítás.....	11
Osztályok.....	12
BaseController.....	12
AccountController.....	12
AdminController.....	13
AdvertisementController.....	14
AuthController.....	15
BookController.....	16
GenreController.....	17
Modellek.....	17
<b>Asztali.....</b>	<b>18</b>
UML terv.....	18
Általános működés.....	18
Indítás.....	18
Osztályok.....	19
Konyvter_desktop.....	19
MainController.....	19
RestController.....	19
GuiController.....	21

AdvertisementModel.....	23
UserModel.....	23
ViewModel.....	23
RESTModel.....	23
<b>Frontend.....</b>	<b>26</b>
Általános működés.....	26
Indítás.....	27
Felépítés.....	27
Komponensek.....	29
app komponens.....	29
advertisement komponens.....	30
main komponens.....	30
myadvertisements komponens.....	31
newadvertisement komponens.....	32
register komponens.....	32
settings komponens.....	33
updateadvertisement komponens.....	33
shared könyvtár.....	34
<b>Tesztelés.....</b>	<b>35</b>
REST API.....	35
Asztali alkalmazás.....	36
Web.....	38

# Ismertető

## Cél

Célunk egy online marketplace (piactér) létrehozása könyveknek. Ahol a felhasználók autentikáció után adhatják, vehetik, cserélhetik egymás között új és használt könyveiket. Természetesen a könyveket kategorizáljuk és a hirdetés során bevitt adatok alapján részletes szűrési, keresési módokat biztosítunk. A program a felhasználók számára webes Single-page application formájában érhető el, egyaránt rendelkezik asztali és mobil nézetrel.

Emellett rendelkezésre áll még egy asztali alkalmazás is amely kizárólag adminisztratív (admin) feladatokat lát el, mint például jelentett hirdetés törlése, felhasználó törlése és a hirdetésekhez, felhasználókhoz kapcsolódó adatok megjelenítése, szűrése.

A backend-et Laravel php keretrendszer, az adatbázist MariaDB szerver biztosítja.

## Felhasznált technológiák

### Backend

Laravel API (PHP keretrendszer)

- laravel-sanctum (Autentikációs kiegészítő csomag)

Mariadb server (Adatbázis kiszolgáló)

### Frontend

Angular (TypeScript-alapú) 13.0.4

- routing module
- guard module
- ngx-pagination module
- Bootstrap 5 CSS

### Asztali alkalmazás

Apache NetBeans 12.6

Java 17

gson 2.8.2

# Fejlesztői környezet

- XAMPP Version: 8.1.1
  - Adatbázis kezelése a mysql klienssel.
- Visual Studio Code
- Laravel Framework 8.82.0
- Composer version 2.2.3
- PHP 8.1.1
- Angular 13.0.4
- npm 6.14.12
- node v14.16.1
- Apache NetBeans 12.6
- Java 17

## Kódolási konvenciók

A kódot git verziókezelővel használjuk.

### Alapkönyvtárak

- api
- database
- desktop
- doc
- web

Az api backend, a web a webes frontend.

## Munkamegosztás

Szivák Gergő

- Backend (REST API, adatbázis) és tesztelése
- Frontend

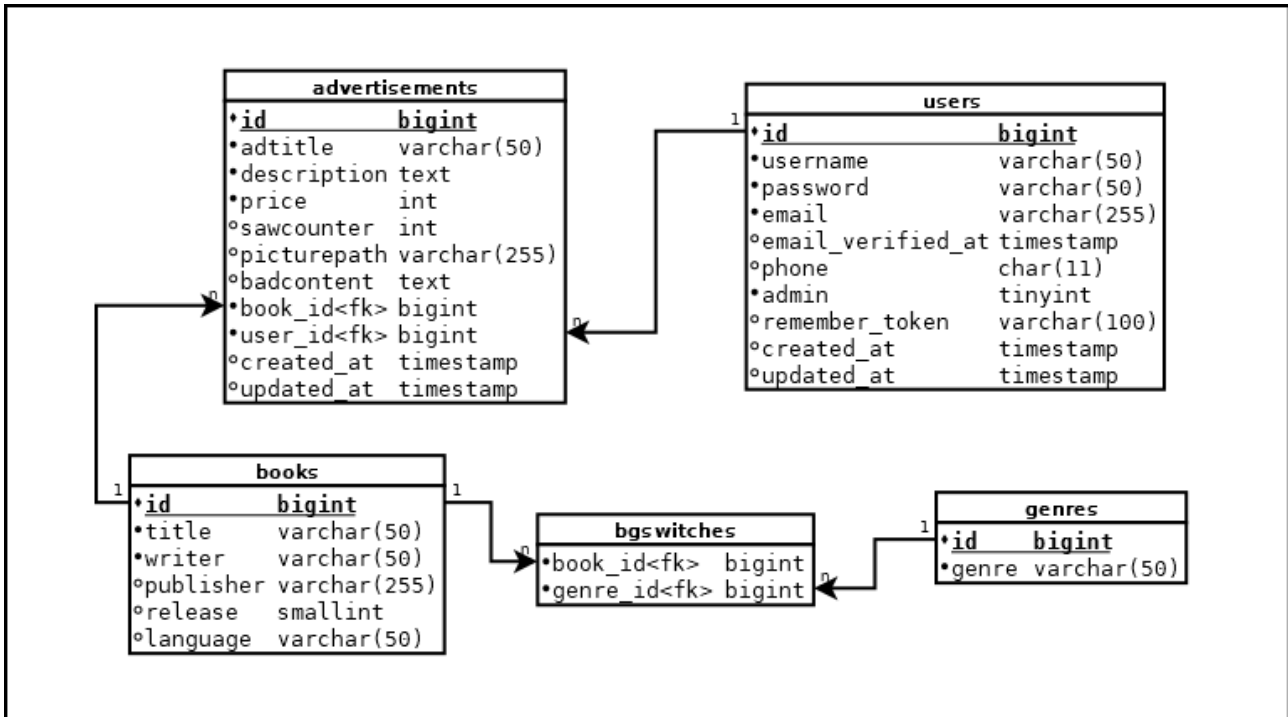
Hermányi Gergely Barnabás

- Asztali alkalmazás és tesztelése
- Frontend tesztelése

# Adatbázis

## Tervezés és Szerkezet

Adatbázis terv:



### users:

- username: felhasználónév
- password: jelszó
- email: e-mail cím
- email\_verified\_at: e-mail igazolási időpontja
- phone: telefonszám
- admin: boolean admin vagy felhasználói
- remember\_token: token
- created\_at: létrehozás időpontja
- updated\_at: módosítás időpontja

### advertisements:

- adtitle: hirdetés címe
- description: leírás
- price: ár
- sawcounter: megtekintések száma
- picturepath: a kép elérési útvonala
- badcontent: jelentés szövege, ha üres nincs jelentve

- book\_id: külsőkulcs(books tábla)
- user\_id: külsőkulcs(users tábla)
- created\_at: létrehozás időpontja
- updated\_at: módosítás időpontja

#### **books:**

- title: könyv címe
- writer: író
- publisher: kiadó
- release: megjelenés
- language: nyelv

#### **bgswitches:** (kapcsolótábla)

- book\_id: külsőkulcs(books tábla)
- genre\_id: külsőkulcs(genres tábla)

#### **genres:**

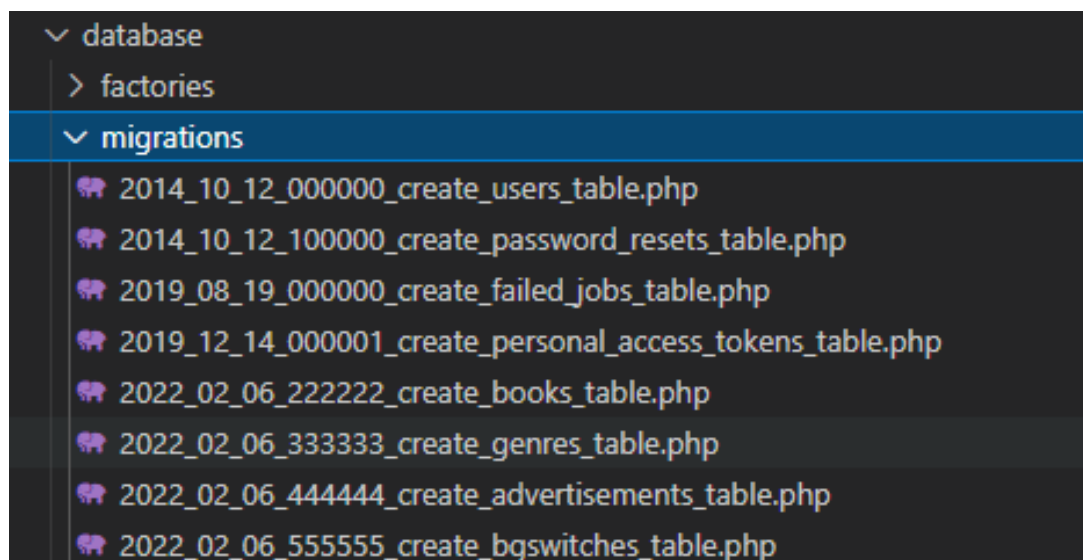
- genre: műfaj

## Adatbázis létrehozása

Az adatbázis Laravel használatával jött létre, ehhez 3 dolgot kellett tenni:

1. php artisan make:migration create\_tablanev\_table

ezzel a paranccsal létrehozhatók a következő migrációs fájlok:

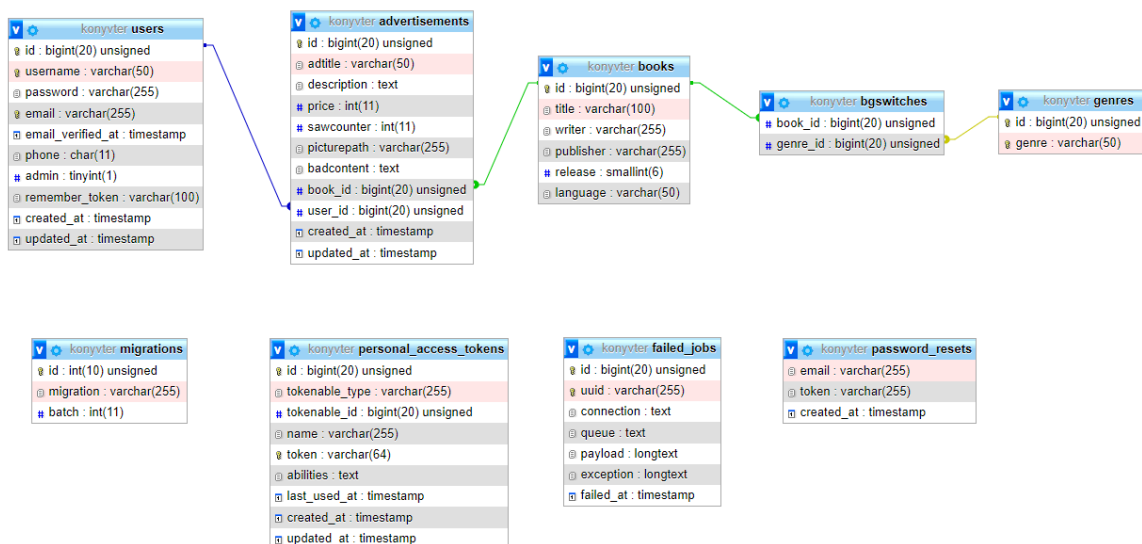


## 2. a tábla mezők hozzáadása a migrációs fájlukhoz

```
Schema::create('users', function (Blueprint $table) {
    $table->id();
    $table->string('username', 50)->unique();
    $table->string('password');
    $table->string('email')->unique();
    $table->timestamp('email_verified_at')->nullable();
    $table->char('phone', 11)->nullable();
    $table->boolean('admin')->default(false);
    $table->rememberToken();
    $table->timestamps();
});
```

## 3. php artisan migrate utasítás futtatásával létrejön az adatbázis

Adatbázis végeredmény a Laravel saját tábláival együtt:



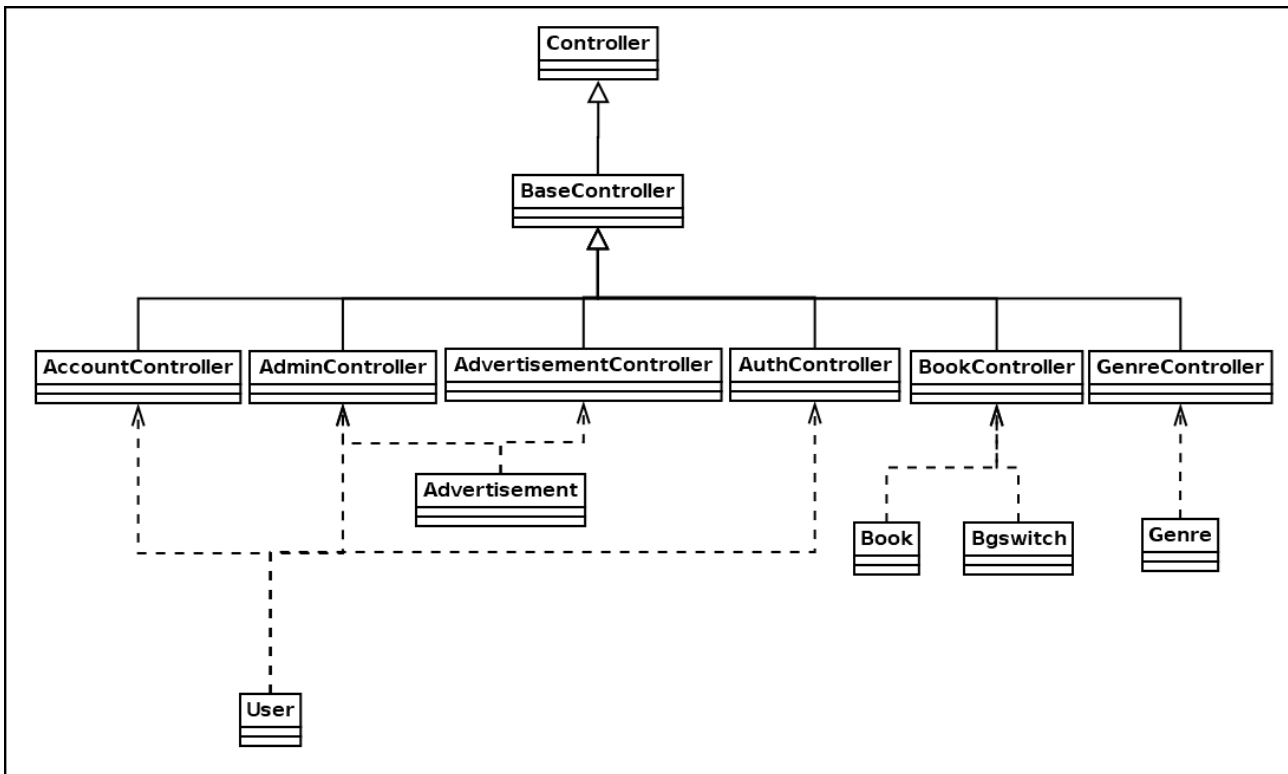
### Laravel saját táblái:

- migrations
- personal\_access\_tokens
- failed\_jobs
- password\_resets



# REST API

## UML terv



## Általános működés

A REST API http kéréseket fogad, melyek tartalmazzák a műveletekhez szükséges megfelelő adatokat. A kényes műveletek végpontjai védettek, autentikáció szükséges a használatukhoz. Ilyenek a kijelentkezés, hirdetés feladása, módosítása, törlése stb... Ugyanakkor a regisztrációhoz, bejelentkezéshez, kereséshez vagy a hirdetés jelentéséhez nem kell autentikáció.

Az adatokat Json vagy Multipart formátumban fogadja és feldolgozza. A vezérlést kontrollerek valósítják meg, minden adatkezelési csoportnak külön kontrollere van, itt történik az adatfeldolgozás.

A kontrollerek modellekkel vannak kapcsolatban amelyek az adatkezelésért felelősek, itt történik az adatok adatbázisból kiolvasása, illetve a az adatok kiírása adatbázisba.

A modellek adatbázis táblákkal vannak kapcsolatban, melyek az adatok tárolásáért felelősek. Az adatbázis táblák adatait a modellek kezelik.

## Végpontok

Végpont	Metódus	Hitelesítés	Leírás
/api/register	POST	nem	regisztráció
/api/login	POST	nem	bejelentkezés
/api/logout	POST	igen	kijelentkezés
/api/admin/reportedads	GET	igen & admin	jelentett hirdetések
/api/admin/users	GET	igen & admin	összes felhasználó
/api/admin/reportedads/{adtitle}	GET	igen & admin	jelentett hirdetések keresés
/api/admin/users/{username}	GET	igen & admin	felhasználók keresés
/api/admin/reportedads/remove/{id}	PUT	igen & admin	jelentett hirdetés visszavonása
/api/web/genres	GET	nem	összes műfaj
/api/web/books	GET	nem	összes könyv
/api/web/books	POST	igen	új könyv
/api/web/books/{id}	GET	nem	adott könyv betöltése
/api/web/books/{id}	PUT	igen	könyv módosítása
/api/web/books/{id}	DELETE	igen	könyv törlése
/api/account/{id?}	GET	nem	publikus fiókadatok
/api/account/{id?}	PUT	igen/user vagy admin	fiók módosítása
/api/account/{id?}	DELETE	igen/user vagy admin	fiók törlése
/api/web/advertisements	GET	nem	összes hirdetés
/api/web/advertisements/my	GET	igen	hirdetésim
/api/web/advertisements	POST	igen	új hirdetés
/api/web/advertisements/{id}	GET	nem	adott hirdetés betöltése
/api/web/advertisements/{id}	POST/ _method:PUT	igen	hirdetés módosítása

Végpont	Metódus	Hitelesítés	Leírás
/api/web/advertisements/report/{id}	PUT	nem	hirdetés jelentése
/api/web/advertisements/{id}	DELETE	igen/user vagy admin	hirdetés törlése
/api/web/advertisements/filter	POST	nem	keresés és szűrés a hirdetések között

### Megjegyzés a végpontokhoz:

a képek az alábbi végponton érhetők el:

/storage/images/{username}/{advertisementId}/{imagename}

/api/web/advertisements/{id} | POST -> ezen az útvonalon a megfelelő működés érdekében fel kell venni a következő paramétert: "name": "\_method", "value": "PUT", ugyanis a multipart csak a GET és POST metódust támogatja

# Indítás

- 1) composer install
- 2) importtal:
  - 1) használja konyvter/database/exports/-ban lévő konyvter.sql export fájlt az adatbázis létrehozásához
  - 2) konyvter/database/exports/-ban lévő storage könyvtár áthelyezésére a konyvter/api/konyvter/-mappába, az ott lévő storage könyvtárat nyugodtan írja felül
  - 3) .env example másolása, a másolat átnevezése .env névre majd az adatbázis beállítása
- 3) import nélkül:
  - 1) üres adatbázis létrehozása szabadon választott néven
  - 2) .env example másolása, a másolat átnevezése .env névre majd az adatbázis beállítása
  - 3) php artisan migrate
- 4) php artisan key:generate
- 5) php artisan storage:link
- 6) php artisan serve

## Megjegyzés az indításhoz

php artisan migrate parancsra a táblákon kívül a következők jönnek létre:

Létrejön egy admin felhasználó

Létrejön egy TesztElek nevű felhasználó és a hozzá tartozó hirdetések, a hirdetésekhez tartozó könyvek, valamint a könyvekhez tartozó műfajok kapcsolói.

A TesztElekhez tartozó storage könyvtárában lévő első 3 kép GitHub-ra is felkerül.

A műfajok tábla teljesen feltöltve jön létre REST API-n keresztül nem módosítható, mivel a weboldal kinézetét nagyban befolyásolja

# Osztályok

## **BaseController**

Feladata a műveletek válaszainak küldése. Sikeres művelet esetén az adatot és üzenetet visszaküldi, sikertelen művelet esetén hibaüzenetet küld.

sendResponse()

- bejövő paraméterek: \$result, \$message ( A felhasználónak szánt saját üzenet )
- kimenő adatok: Response ( saját üzenet ).

sendError()

- bejövő paraméterek: \$error ( A php által generált hibaüzenet ), \$errorMessages ( Saját hibaüzenet ), \$code ( A válaszban küldendő http kód ).
- kimenő adatok: Response ( hibaüzenet és a hiba kódja ).

## **AccountController**

Feladata a felhasználói fiókhoz kapcsolódó műveletek kezelése.

show()

- bejövő paraméterek: id (lehet null, ha null sanctum alapján azonosít)
- kimenő adatok: felhasználó nyilvános adatai + üzenet

update()

- bejövő paraméterek: id (lehet null, ha null sanctum alapján azonosít), \$request

Az adatokat validálja majd frissíti az adatbázis megfelelő sorát

- kimenő adatok: felhasználónév + üzenet

delete()

- bejövő paraméterek: id (lehet null, ha null sanctum alapján azonosít)

Törli a felhasználót az adatbázisból

- kimenő adatok: [ ] + üzenet

## ***AdminController***

Feladata az admin jogosultsággal elérhető műveletek kezelése, sanctum alapján azonosítja az admint:

```
auth( "sanctum" )->user()->admin
```

reportedads()

- bejövő paraméterek: nincs

megkeresi az összes jelentett hirdetést

- kimenő adatok: jelentett hirdetések + üzenet

user()

- bejövő paraméterek: nincs

megkeresi az összes felhasználót

- kimenő adatok: felhasználók + üzenet

searchads()

- bejövő paraméterek: hirdetés címe

cím alapján keres a jelentett hirdetések között

- kimenő adatok: keresési találatok + üzenet

searchuser()

- bejövő paraméterek: felhasználónév

felhasználónév alapján keres a felhasználók között

- kimenő adatok: keresési találatok + üzenet

removeReport()

- bejövő paraméterek: hirdetés \$id

hirdetés id alapján visszavonja a jelentést

- kimenő adatok: null + üzenet

## **AdvertisementController**

Feladata a hirdetéssel kapcsolatos műveletek kezelése

index()

- bejövő paraméterek: nincs

összegyűjti a hirdetéseket és időrendi sorrendben visszaadja

- kimenő adatok: hirdetések + üzenet

create()

- bejövő paraméterek: \$request

Validál, utána létrehozza a hirdetést és eltárolja a képet

- kimenő adatok: létrehozott hirdetés + üzenet

show()

- bejövő paraméterek: \$id

id alapján megkeres egy hirdetést

- kimenő adatok: hirdetés + üzenet

update()

- bejövő paraméterek: \$request

Validál, utána frissíti a hirdetést és ha érkezik kép eltárolja

- kimenő adatok: frissített hirdetés + üzenet

delete()

- bejövő paraméterek: \$id

id alapján megkeres egy hirdetést és törli

- kimenő adatok: [ ] + üzenet

reportAd()

- bejövő paraméterek: \$id, \$request

id alapján megkeres egy hirdetést, validál és frissíti a hirdetés badcontent mezőjét a jelentés szövegével

- kimenő adatok: jelentett hirdetés + üzenet

getMyAds()

- bejövő paraméterek: nincs

sanctum alapján összegyűjti a bejelentkezett felhasználó hirdetéseit

- kimenő adatok: hirdetéseim + üzenet

filter()

- bejövő paraméterek: \$request

validál, de nincs elvárt paraméter majd rákeres a beállított paraméterekre, de csak azokra, majd nézi a találatok „metszetét” és így szűkíti a keresést

az alábbiakra lehet rákeresni:

- hirdetés címe
- könyv címe
- maximum ár
- író
- műfaj

- kimenő adatok: keresési találatok + üzenet

## ***AuthController***

Feladata egy új felhasználó felvétele, felhasználók autentikációja, felhasználók kijelentkeztetése..

register()

- bejövő paraméterek: \$request ( az regisztrációhoz szükséges adatok a kérésben, username, email, password, confirm\_password, phone(nem kötelező) )

Az adatokat validálja, majd sikeres érvényesítés után bejegyzí az users adatbázis tábla megfelelő mezőibe (a jelszavakat titkosítva).

- kimenő adatok: üzenet

login()

Feladata a felhasználó azonosítása név és jelszó alapján. Sikeres autentikáció esetén generál egy tokent a felhasználó számára és bejegyzí a personal\_access\_tokens adatbázis tábla megfelelő mezőjébe, majd átadja a BaseController sendResponse() metódusának a saját üzenettel együtt.

- bejövő paraméterek: \$request ( a bejelentkezéshez szükséges adatok a kérésben, username, password ).

- kimenő adatok: token + üzenet

logout()

Feladata a felhasználó kijelentkeztetése és a token törlése az adatbázis táblából.

- bejövő paraméterek: \$request ( a kijelentkezéshez szükséges adatok a kérésben, token )

- kimenő adatok: üzenet



## **BookController**

index()

- bejövő paraméterek: nincs

meghívja a `getBookData()` függvényt ami visszatér a könyvekkel

- kimenő adatok: összes könyv + üzenet

create()

- bejövő paraméterek: `$request`

Validál, utána ellenőrzi létezik-e már a könyv `checkExist()` függvénnyel és ha nem, létrehozza, valamint a kapcsolatokat is kiépíti a műfajokkal a kapcsolótáblán keresztül

- kimenő adatok: létrehozott könyv + üzenet

show()

- bejövő paraméterek: `$id`

`id` alapján megkeres egy könyvet `getBookData()` függvénnyel

- kimenő adatok: könyv + üzenet

update()

- bejövő paraméterek: `$request`, `$id`

`id` alapján megkeresi a könyvet, validál, utána ellenőrzi létezik-e már a könyv `checkExist()` függvénnyel, valamint ellenőrzi, hogy használatban van-e más felhasználónál és ha nem, frissíti, ha igen létrehozza, valamint a kapcsolatokat is frissíti a műfajokkal a kapcsolótáblán keresztül

- kimenő adatok: frissített könyv + üzenet

delete()

- bejövő paraméterek: `$id`

`id` alapján megkeres egy könyvet és törli

- kimenő adatok: `[ ]` + üzenet

## **GenreController**

index()

- bejövő paraméterek: nincs

visszatér az összes műfajjal

- kimenő adatok: összes műfaj + üzenet

Több metódus nincsen ugyanis a felhasználó nem adhat hozzá műfajt az adatbázishoz, csak a meglévőkből választhat, ugyanis a műfajok száma nagyban befolyásolja a weboldal kinézetét.

## **Modellek**

- **Advertisement:**

Az advertisements adatbázis táblával van kapcsolatban az adatokat onnan olvassa és oda írja.

- **Bgswitch:**

A bgswitches adatbázis táblával van kapcsolatban az adatokat onnan olvassa és oda írja.

- **Book:**

A books adatbázis táblával van kapcsolatban az adatokat onnan olvassa és oda írja.

- **Genre:**

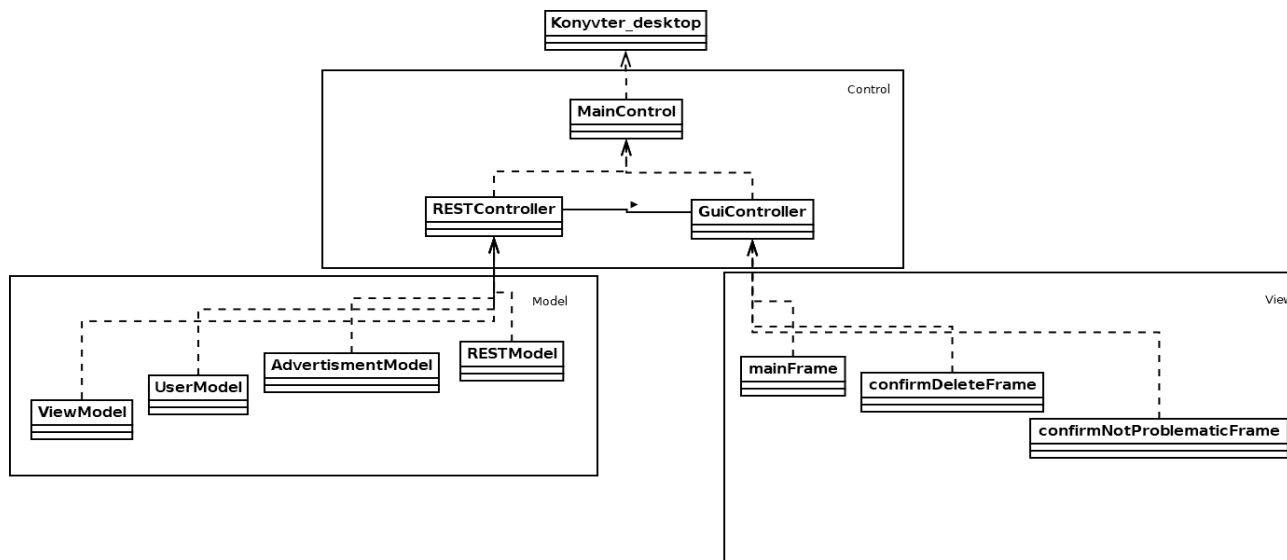
Az genres adatbázis táblával van kapcsolatban az adatokat onnan olvassa és oda írja.

- **User:**

Az users adatbázis táblával van kapcsolatban az adatokat onnan olvassa és oda írja.

# Asztali

## UML terv



## Általános működés

Az asztali alkalmazás publikus terjesztése nem történik meg, ezért nincsen bejelentkező ablak hanem egyből belép az alkalmazás admin felhasználóval és a hozzá tartozó jelszóval. Bejelentkezéskor token generálódik, kijelentkezéskor ez a token törlődik.

A program felhasználók és a jelentett hirdetések szükséges adatait jeleníti meg. A felhasználók kapcsolattartás és szükség esetén felhasználó törlése céljából, a hirdetések pedig jelentés indokának ellenőrzése miatt, elbírálás után pedig a hirdetés vagy jelentés törlése miatt jelennek meg.

Az adatokat a Backend biztosítja és REST API kérésekkel szerezzük meg, minden funkció működéséhez szükséges a kapcsolat a Backendel, különben nem fog működni az alkalmazás egyik lényeges funkciója sem.

## Indítás

- starter.bat fájl futtatása

### Megjegyzés az indításhoz

Az alkalmazás megfelelő futásához nem szükséges, de az adatok beöltődéséhez és az azokkal való műveletekhez esszenciális a már működő, és futó Backend.

A hirdetések megnyitásánál pedig a futó Frontend

# Osztályok

## ***Konyvter\_desktop***

Ez a fő osztály az alkalmazásban

Feladata: Beállítja az alkalmazás stílusát, valamint megpéldányosítja a MainControllert

## ***MainController***

Feladata: Megpéldányosítja a RESTControllert, valamint a GuiControllert és átadjuk neki paraméterként a RESTController egy új példányát

## ***RESTController***

Feladata: A REST API-val kommunikáló Modeltől átveszi vagy átadja neki a megfelelő adatokat és továbbítja a GuiControllernek vagy a RESTModelnek.

Osztályváltozók:

- RESTModel restMdl
- String token
- search\_text
- id

RESTController()

Megpéldányosítja a RESTModel osztályt

Elmenti a token változóba a sikeres belépés után kapott token

inicializálja a search\_text változót üresen

inicializálja az id változót üresen

getToken()

Megszerzi a RESTModeltől a belépés után kapott token

- kimenő adatok: token

Logout()

Meghívja a RESTModel Logout függvényét és átadja neki a token

getUsers()

Meghívja a RESTModel Users nevű függvényét és átadja neki a token és a search\_text változót

- kimenő adatok: A felhasználók bizonyos adatai

getAdvertisements()

Meghívja a RESTModel Advertisements nevű függvényét és átadja neki a token és a search\_text változót

- kimenő adatok: A hirdetések bizonyos adatai

DeleteUser()

Meghívja a RESTModel DeleteUsers nevű függvényét és átadja neki a token és az id változót

DeleteAdvertisement()

Meghívja a RESTModel DeleteAdvertisement nevű függvényét és átadja neki a token és az id változót

ValidAdvertisement()

Meghívja a RESTModel ValidAdvertisement nevű függvényét és átadja neki a token és az id változót

setId()

- bejövő paraméter: id( lehet null )

Beállítja, hogy az osztályváltozó id egyenlő a paraméterként érkezett id-val

getLoginMessage()

Eltárolja egy változóban a RESTModelben keletkező bejelentkezési üzenetet

- kimenő adatok: Bejelentkezéskor keletkező üzenet

getDeleteAdvertisementMessage()

Eltárolja egy változóban a RESTModelben hirdetés törlésekor keletkező üzenetet

- kimenő adatok: Hirdetés törlésekor keletkező üzenet

getDeleteUserMessage()

Eltárolja egy változóban a RESTModelben felhasználó törlésekor keletkező üzenetet

- kimenő adatok: Felhasználó törlésekor keletkező üzenet

getValidAdvertisementMessage()

Eltárolja egy változóban a RESTModelben egy hirdetés jelentésének megszüntetésénél keletkező üzenetet

- kimenő adatok: Hirdetés jelentésének megszüntetésénél törlésekor keletkező üzenet

## **GuiController**

Feladata: Az ablakban minden funkció működjön és minden szükséges adat megjelenjen

Osztályváltozók:

- mainFrame mainFrm
- confirmDeleteFrame deleteFrm
- confirmNotProblematicFrame problemFrm
- ViewModel viewMdl
- Vector<Vector<Object>> tableData
- RESTController restCtr

GuiController()

- bejövő paraméterek: restCtr

Beállítja, hogy az osztályváltozó restCtr egyenlő a paraméterként érkezett restCtr-vel

Megpéldányosítja a ViewModel osztályt

Elindítja a program elindulásához nélkülözhetetlen függvényeket

ActionListeners()

Ráköti az ActionListener-eket a szükséges komponensekre

initFrames()

Meghívja a függvényt ami beállítja a fő ablaknak a tulajdonságait, megpéldányosítja a többi ablakot, valamint elindítja az adatok betöltését a táblákba

initMainFrameProperties()

Beállítja a fő ablaknak a tulajdonságait

initConfirmDeleteFrame()

Beállítja a törlési ablaknak a tulajdonságait

disposeConfrimDeleteFrame()

Felszámolja a törlési ablakot

initConfirmProblematicFrame()

Beállítja a jelentés visszavonására vonatkozó ablak tulajdonságait

disposeConfirmProblematicFrame()

Felszámolja a jelentés visszavonására vonatkozó ablakot

getLoginStatus()

Megvizsgálja, hogy a bejelentkezésnél kapott üzenet üres e

reConnect()

Újraindítja a beélesi folyamatot és elindítja az adatok táblába töltését

initTables()

Feltölti a táblázatokat adatokkal

ThreadStarter()

Elindítja az új szálakat

search()

Megszerzi a keresési mezőből a szöveget és elindítja az adatok táblába töltését, a keresés eredményével

exit()

Kitörli a tokent és bezárja a programot

delete()

Megvizsgálja, hogy melyik tábla van megnyitva és attól függően elindítja a megfelelő törlési függvényt

deleteUser()

Kitörli a kiválasztott felhasználót, megjeleníti a visszatérő üzenetet, aztán utána újra elindítja az adatok táblába töltését

deleteAdvertisement()

Kitörli a kiválasztott hirdetést, megjeleníti a visszatérő üzenetet, aztán utána újra elindítja az adatok táblába töltését

validAdvertisement()

Törli a jelentést a hirdetésből megjeleníti a visszatérő üzenetet, aztán utána újra elindítja az adatok táblába töltését

openAdvertisement()

Megnyit egy hirdetést a webes felületen (Szükséges a Frontendnek futni a háttérben)

refresh()

Láthatóvá teszi a Frissítés gombot

clearStatusLabel()

Kiüríti a statusLbl-t

timer()

(Külön szálon fut) 30 másodpercenként újra lekéri és betölti az adatokat a táblákba

buttonsAvailability()

Ha a hirdetések tábla van megnyitva akkor megjelenít gombokat

### ***AdvertisementModel***

Feladat: Sablonként funkcionál a RESTModel osztály számára

### ***UserModel***

Feladat: Sablonként funkcionál a RESTModel osztály számára

### ***ViewModel***

Feladat: Ebben az osztályban vannak tárolva a táblák oszlopnevei

getUserColumnNames()

Egy tömbbe gyűjtjük az oszlopneveket

- kimenő adatok: Oszlopnevek

getAdvertisementColumnNames()

Egy tömbbe gyűjtjük az oszlopneveket

- kimenő adatok: Oszlopnevek

### ***RESTModel***

Feladat: Kommunikálás a Backendel, adatok fogadása és küldése

Osztályváltozók

- String loginMessage
- String deleteUserMessage
- String deleteAdvertisementMessage
- String validAdvertisementMessage
- Integer loginResponseCode
- Integer logoutResponseCode
- Integer usersResponseCode
- Integer advertisementResponseCode



- Integer deleteUserResponseCode
- Integer deleteAdvertisementResponseCode
- Integer validAdvertisementResponseCode

Login()

Megpróbálja lefuttatni a TryLogin függvényt

- kimenő adatok: token

tryLogin()

Bejelentkezik az alkalmazásba és kiolvassa a tokent

- kimenő adatok: token

Logout()

- bejövő paraméterek: token

Megpróbálja lefuttatni a TryLogout függvényt

tryLogout()

- bejövő paraméterek: token

Visszaadja a tokent a Backendnek

Users()

- bejövő paraméterek: token, search\_text

Megpróbálja lefuttatni a tryUsers függvényt

- kimenő adatok: Felhasználók adatai

tryUsers()

- bejövő paraméterek: token, search\_text

Lefuttatja a kérést Backendnek és kiolvassa a felhasználók adatait

- kimenő adatok: Felhasználók adatai

Advertisements()

- bejövő paraméterek: token, search\_text

Megpróbálja lefuttatni a tryAdvertisements függvényt

- kimenő adatok: Felhasználók adatai

tryAdvertisements()

- bejövő paraméterek: token, search\_text

Lefuttatja a kérést Backendnek és kiolvassa a hirdetések adatait

- kimenő adatok: Felhasználók adatai

DeleteUsers()

- bejövő paraméterek: token, id

Megpróbálja lefuttatni a tryDeleteUsers függvényt

tryDeleteUsers()

- bejövő paraméterek: token, id

Elküldi az id-t a Backendnek és törli a felhasználót és a hozzá tartozó hirdetéseket

DeleteAdvertisements()

- bejövő paraméterek: token, id

Megpróbálja lefuttatni a tryDeleteAdvertisements függvényt

tryDeleteAdvertisements()

- bejövő paraméterek: token, id

Elküldi az id-t a Backendnek és törli a hirdetést

ValidAdvertisements()

- bejövő paraméterek: token, id

Megpróbálja lefuttatni a tryValidAdvertisements függvényt

tryValidAdvertisements()

- bejövő paraméterek: token, id

Elküldi az id-t a Backendnek és törli a jelentett státuszt a hirdetésből

getLoginMessage()

Visszatér a bejelentkezésakor érkező üzenettel

- kimenő adatok: Bejelentkezésakor érkező üzenet

getDeleteUserMessage()

Visszatér a felhasználó törlésekor érkező üzenettel

- kimenő adatok: Felhasználó törlésekor érkező üzenet

getDeleteAdvertisementMessage()

Visszatér a hirdetés törlésekor érkező üzenettel

- kimenő adatok: Hirdetés törlésekor érkező üzenet

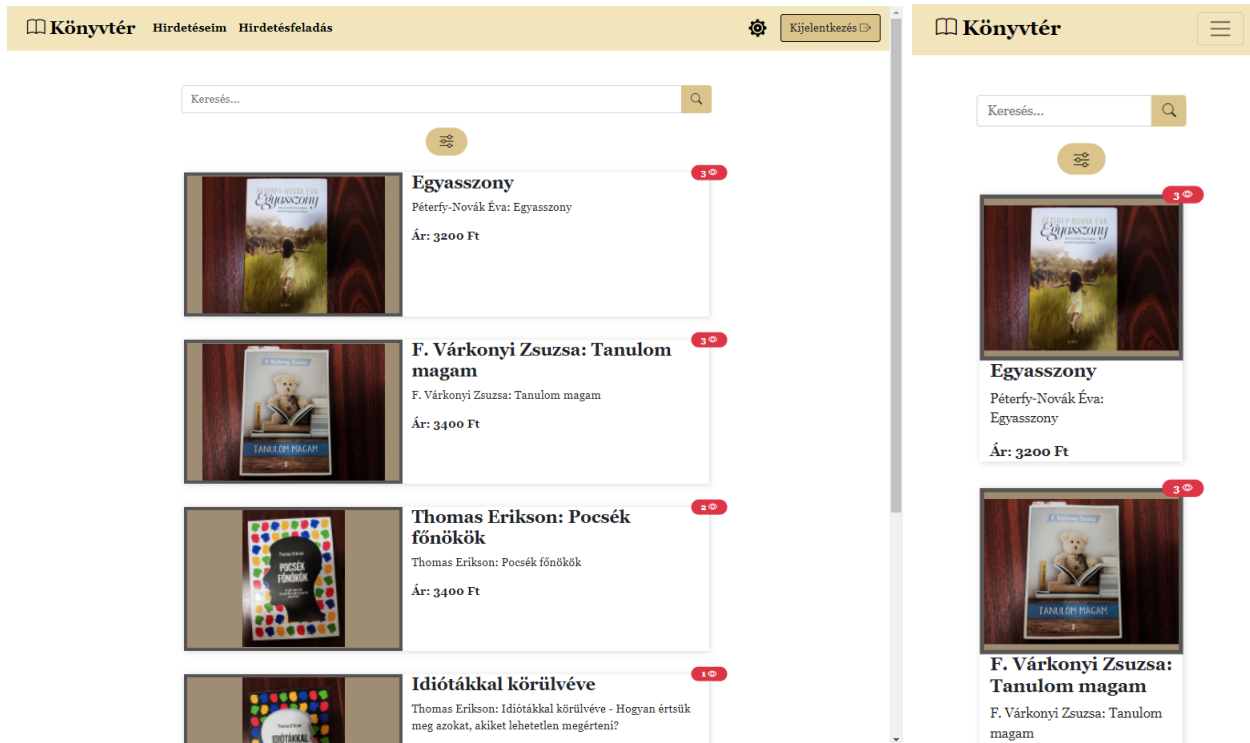
getvalidAdvertisementMessage()

Visszatér a hirdetés jelentése eltávolításakor érkező üzenettel

- kimenő adatok: Hirdetés jelentése eltávolításakor érkező üzenet

# Frontend

## Főoldal asztali és mobil nézet



## Általános működés

A teljes Frontend Angular (TypeScript-alapú) keretrendszer használatával készült, ennek megfelelően Single-page application, olyan webalkalmazás, amely interakcióba lép a felhasználóval azáltal, hogy dinamikusan átírja az aktuális weboldalt a REST API-ról származó új adatokkal, ahelyett hogy teljesen új oldalakat kellene betölteni. Emellett az oldal teljes mértékben reszponzív és mobil eszközről történő használata során egy mobil applikáció érzetét kelti, ezt Bootstrap 5 használatával és rengeteg saját CSS-el sikerült elérni.

# Indítás

- 1) npm install
- 2) ng serve -o

## Megjegyzés az indításhoz

A képek betöltéséhez szükség van a könyvter/database/exports/-ban lévő storage könyvtár áthelyezésére a könyvter/api/könyvter/-ba, az ott lévő storage könyvtárat nyugodtan írja felül. Valamint ne a REST API-nál ne futassa a php artisan migrate parancsot, hanem használja könyvter/database/exports/-ban lévő adatbázis export fájlt.

## Felépítés

```
| -node_modules/  
`-src/  
    | -app/  
    |     | -advertisement/  
    |     | -main/  
    |     | -myadvertisements/  
    |     | -newadvertisement/  
    |     | -register/  
    |     | -settings/  
    |     | -shared/  
    |     | -updateadvertisement/  
    |     | -app-routing.modules.ts  
    |     | -app.components.css  
    |     | -app.component.html  
    |     | -app.component.spec.ts  
    |     | -app.components.ts  
    |     | -app.modules.ts  
    `index.html
```

- advertisement/ - saját komponens
- main/ - saját komponens
- myadvertisements/ - saját komponens
- newadvertisement/ - saját komponens
- register/ - saját komponens
- settings/ - saját komponens
- shared/ - auth.service és auth.guard
- updateadvertisement/ - saját komponens
- app-routing.modules.ts - Az útválasztási bejegyzések
- app.components.css - Az alkalmazás CSS beállításai
- app.components.html - Az alkalmazás fő nézet oldala
- app.component.spec.ts - Script a teszteléshez
- app.components.ts - Fő script komponens
- app.modules.ts - Modulok betöltése
- index.html - Az alkalmazás induló HTML állománya

# Komponensek

## ***app komponens***

app.component.html:

- nav
- modal
- router-outlet
- footer

app-routing.module.ts

- Az útválasztási bejegyzések

app.component.css

- saját css az app komponensre vonatkozóan

app.component.spec.ts

- tesztek

app.component.ts

- ngOnInit() Lifecycle hook amelyet azután hívnak meg, hogy az Angular inicializálta egy direktíva összes adatahoz kötött tulajdonságát.
- login() meghívja az auth.service login metódusát és átadja a bejelentkezési adatokat, sikeres bejelentkezés esetén pedig eltárolja a kapott tokent a lokális tárolóba
- regpage() átnavigál a register komponensre
- isLoggedIn() meghívja az auth.service isLoggedIn metódusát és visszatér, hogy be van e lépve a felhasználó
- logout() meghívja az auth.service logout metódusát és átnavigál a fő komponensre

app.module.ts

- Modulok betöltése

## ***advertisement komponens***

advertisement.component.html:

- egy hirdetés megjelenítése + jelentő modal

advertisement.component.css

- saját css az advertisement komponensre vonatkozóan

advertisement.component.spec.ts

- tesztek

advertisement.component.ts

- ngOnInit() Lifecycle hook amelyet azután hívnak meg, hogy az Angular inicializálta egy direktíva összes adathoz kötött tulajdonságát.
- getAdData() megszerzi a hirdetés adatait REST API-n keresztül és meghívja a getBookData() és getUserData() metódust
- getBookData() megszerzi a könyv adatait REST API-n keresztül
- getUserData() megszerzi a felhasználó adatait REST API-n keresztül
- report() jelenti a hirdetést, elküldi a jelentés szövegét a REST API-nak

## ***main komponens***

(Főoldal)

main.component.html:

- kereső
- szűrő
- hirdetések betöltése (saját cardokba)
- pagination

main.component.css

- saját css a main komponensre vonatkozóan

main.component.spec.ts

- tesztek

main.component.ts

- ngOnInit() Lifecycle hook amelyet azután hívnak meg, hogy az Angular inicializálta egy direktíva összes adathoz kötött tulajdonságát.

- `getGenres()` megszerzi a műfajokat REST API-n keresztül
- `getAds()` megszerzi a hirdetések adatait REST API-n keresztül, majd végig megy az adatokon egy `foreach` ciklussal és cikluson belül hívogatja a `getBook()` metódust
- `getBook()` megszerzi egy könyv adatait REST API-n keresztül és betölti egy `Object`-be
- `getBookTitle()` megszerzi egy adott könyv íróját és címét id alapján.
- `FilterShow()` a szűrő láthatóságát szabályozza
- `navigateAd()` átnavigál a hirdetésre
- `search()` a keresést és szűrést bonyolítja le és a találatok betöltését szabályozza

## ***myadvertisements komponens***

`myadvertisements.component.html`:

- a bejelentkezett felhasználó hirdetései betöltése (saját cardokba)
- lehetőség a hirdetés szerkesztésére, törlésére

`myadvertisements.component.css`

- saját `css` a `myadvertisements` komponensre vonatkozóan

`myadvertisements.component.spec.ts`

- tesztek

`myadvertisements.component.ts`

- `ngOnInit()` Lifecycle hook amelyet azután hívnak meg, hogy az Angular inicializálta egy direktíva összes adatahoz kötött tulajdonságát.
- `getMyAds()` megszerzi a felhasználó saját hirdetései adatait REST API-n keresztül, majd végig megy az adatokon egy `foreach` ciklussal és cikluson belül hívogatja a `getBook()` metódust
- `getBook()` megszerzi egy könyv adatait REST API-n keresztül és betölti egy `Object`-be
- `getBookTitle()` megszerzi egy adott könyv íróját és címét id alapján.
- `navigateAd()` átnavigál a hirdetésre
- `updateAd()` átnavigál a hirdetés szerkesztő oldalára
- `delete()` törli a hirdetést REST API-n keresztül



## ***newadvertisement komponens***

newadvertisement.component.html:

- form validálással és fájlfeltöltéssel
- műfajokhoz generált checkboxok

newadvertisement.component.css

- saját css a newadvertisement komponensre vonatkozóan

newadvertisement.component.spec.ts

- tesztek

newadvertisement.component.ts

- ngOnInit() Lifecycle hook amelyet azután hívnak meg, hogy az Angular inicializálta egy direktíva összes adathoz kötött tulajdonságát.
- NewAdvertisement() létrehoz egy hirdetést newAd() és newBook() metódusok segítségével, majd átnavigál a hirdetéseim komponensre
- checkboxController() kattintásra beteszi vagy kiveszi a beérkező műfajt a genreList listába
- handleFileInput() eltárolja a beérkező fájlt, file változóba
- getGenres() megszerzi a műfajokat REST API-n keresztül
- newBook() létrehoz egy könyvet REST API-n keresztül
- newAd() létrehoz egy hirdetést REST API-n keresztül (Multipart)

## ***register komponens***

register.component.html:

- form validálással

register.component.css

- saját css a register komponensre vonatkozóan

register.component.spec.ts

- tesztek

register.component.ts

- ngOnInit() Lifecycle hook amelyet azután hívnak meg, hogy az Angular inicializálta egy direktíva összes adathoz kötött tulajdonságát.

- register() meghívja az auth.service register metódusát és átadja a regisztrációs adatokat

## ***settings komponens***

settings.component.html:

- form validálással
- confirm modal a fiók törléséhez

settings.component.css

- saját css a settings komponensre vonatkozóan

settings.component.spec.ts

- tesztek

settings.component.ts

- ngOnInit() Lifecycle hook amelyet azután hívnak meg, hogy az Angular inicializálta egy direktíva összes adathoz kötött tulajdonságát.
- GetUserData() megszerzi a felhasználó adatait REST API-n keresztül és betölti a módosító formba
- update() módosítja a felhasználó adatait REST API-n keresztül
- deleteAccount() törli a felhasználói fiókot és minden hozzá tartozó adatot REST API-n keresztül

## ***updateadvertisement komponens***

updateadvertisement.component.html:

- form validálással és fájlfeltöltéssel, a formba már előre betöltődnek az aktuális adatok
- műfajokhoz generált checkboxok, a checkboxokba már előre betöltődnek az aktuális adatok

updateadvertisement.component.css

- saját css az updateadvertisement komponensre vonatkozóan

updateadvertisement.component.spec.ts

- tesztek

updateadvertisement.component.ts

- `ngOnInit()` Lifecycle hook amelyet azután hívnak meg, hogy az Angular inicializálta egy direktíva összes adathoz kötött tulajdonságát.
- `checkboxController()` kattintásra beteszi vagy kiveszi a beérkező műfajt a `genreList` listába
- `checker()` bepipálja a a korábban már hozzáadott műfajokat
- `handleFileInput()` eltárolja a beérkező fájlt, `file` változóba
- `getGenres()` megszerzi a műfajokat REST API-n keresztül
- `getAdData()` megszerzi a hirdetés adatait REST API-n keresztül és meghívja a `getBookData()` metódust
- `getBookData()` megszerzi a könyv adatait REST API-n keresztül
- `updateAdvertisement()` meghívja az `updateBook()` és `updateAd()` metódust és sikeres módosítás esetén átnavigál a hirdetéseim komponensre
- `updateBook()` módosítja a könyv adatait REST API-n keresztül
- `updateAd()` módosítja a hirdetés adatait REST API-n keresztül

## ***shared könyvtár***

auth.guard.ts

Útvonalak frontendes védelmére használjuk

- `canActivate()` meghívja az `auth.service isLoggedIn()` metódusát, ha `true`-val tér vissza visszatér vele, ha `false`-al átnavigál a főoldalra és visszaadja a `false` értéket

auth.service.ts

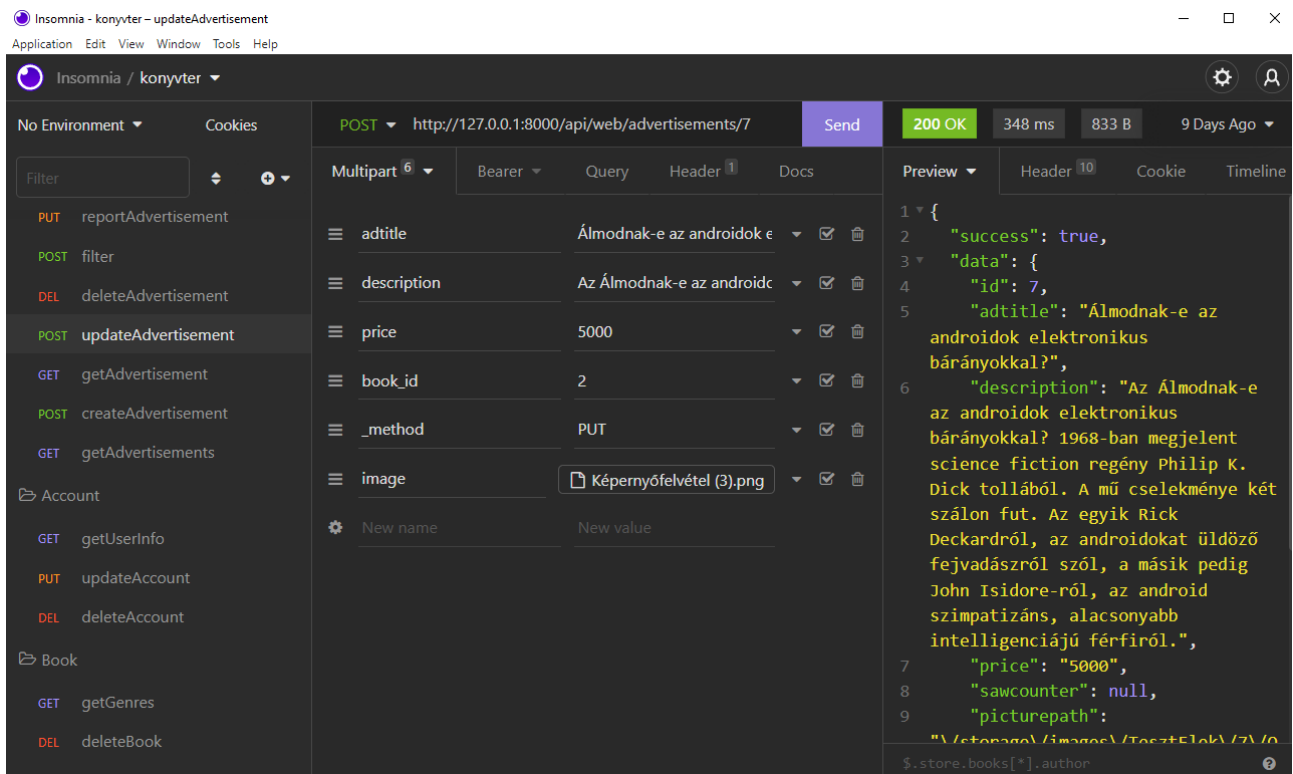
Az autentikációval kapcsolatos metódusokat tartalmazza

- `register()` elküldi a regisztrációs adatokat a REST API-nak és visszatér a válasszal
- `login()` elküldi a bejelentkezési adatokat a REST API-nak és visszatér a válasszal
- `isLoggedIn()` megnézi, hogy be van-e jelentkezve a felhasználó az által, hogy a lokális tárolóban van-e token, ha igen vissza tér vele, ha nem `false` értékkel tér vissza
- `logout()` törli a tokent a lokális tárolóból és REST API-n keresztül is

# Tesztelés

## REST API

### Insomnia - v2022.2.1



A REST API tesztelése Insomnia használatával történt és minden útvonalhoz egy teszt készült. Összesen 25 teszt készült el ami 5 mappába van rendezve:

- **Advertisement:**  
Itt találhatóak a hirdetések útvonalaihoz kapcsolódó tesztek
- **Account:**  
Itt találhatóak a fiókok útvonalaihoz kapcsolódó tesztek
- **Book:**  
Itt találhatóak a könyvek útvonalaihoz kapcsolódó tesztek
- **Admin:**  
Itt találhatóak az admin útvonalaihoz kapcsolódó tesztek
- **Autentikáció:**  
Itt találhatóak az autentikáció útvonalaihoz kapcsolódó tesztek

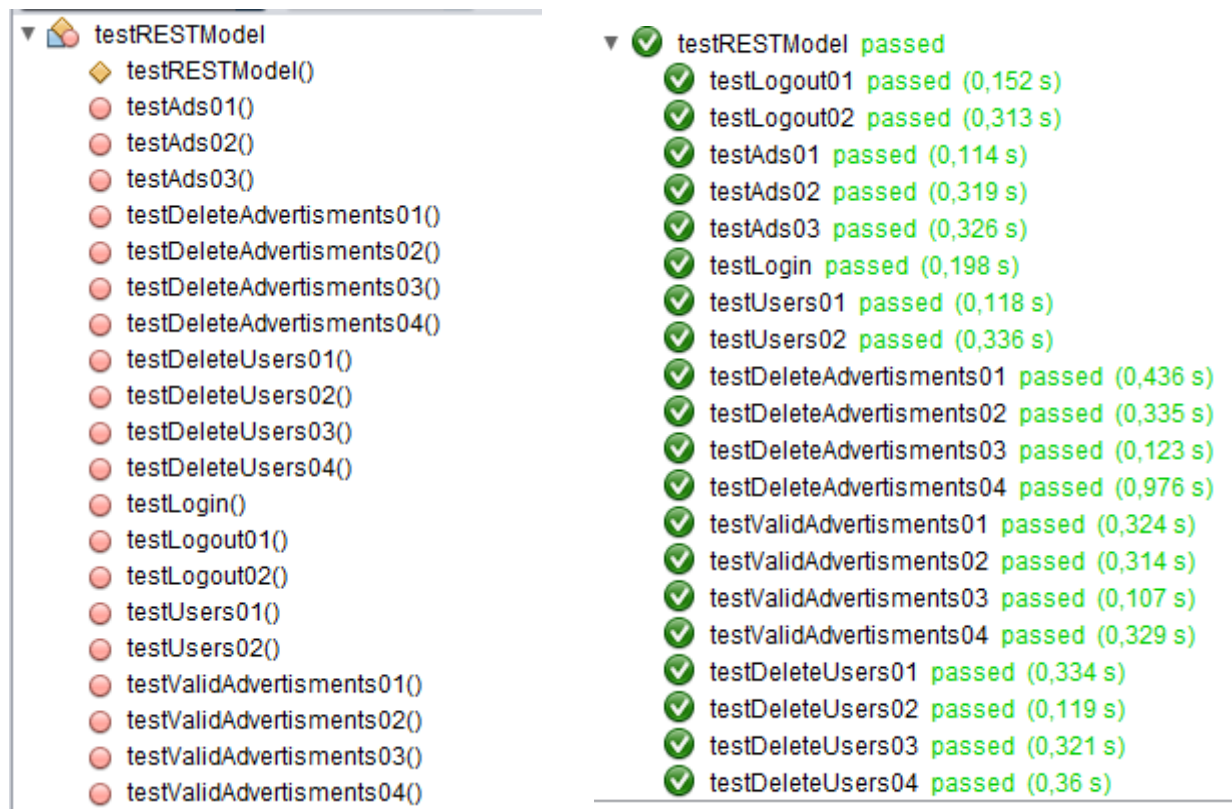
Az Insomnia export fájllai megtalálhatóak a könyvter/api/tests/ mappában.

# Asztali alkalmazás

Apache Netbeans IDE 12.6

A teszteléshez szükséges:

- gson-2.8.2
- junit-jupiter-api-5.8.2
- junit-platform-console-standalone-1.8.2



Az asztali alkalmazás tesztelése Apache Netbeans IDE 12.6 verziójával készült, minden útvonal amelyről kap vagy amelyre küld adatot a Backend felé tesztelésre került, különböző bemeneti paraméterektől függ az adott eredmény ami a visszatérő válasz kódja alapszik között.

A teszteléshez szükséges újra migrálni a backendet, mivel a felhasználó törlése csak így működik biztosan

A tesztek megtalálhatóak a desktop/Konyvter\_desktop/test mappában.

## Egy teszt mélyebb elemzése

```
public void testLogout01() {  
    restMdl = new RESTModel();  
    String token = "";  
    restMdl.Logout(token);  
  
    Integer expected = 500;  
    Integer actual = restMdl.logoutResponseCode;  
  
    assertEquals(expected, actual);  
}
```

A tesztek mindegyike kézzel írott teszt, valamint mindegyik teszt a backendel kommunikáló függvényeket teszteli különböző bemeneti paraméterekkel.

A beérkező paraméterek milyensége határozza meg hogy a teszt milyen kóddal tér vissza.

Ha a jelenlegi tesztnél nem lenne üres a token változó akkor nem 500 hanem 200 kóddal térne vissza, vagy ha nem helyes adattal adnánk át a Logout függvénynek a token változót akkor szintén 500-as válasz kódot kapnánk vissza.

# Web

A web Black Box és White Box módszerrel lett letesztelve. Az angular alapvetően generál minden egyes komponenshez teszt(eket), ami mindig egy spec.ts végű fájlban található.

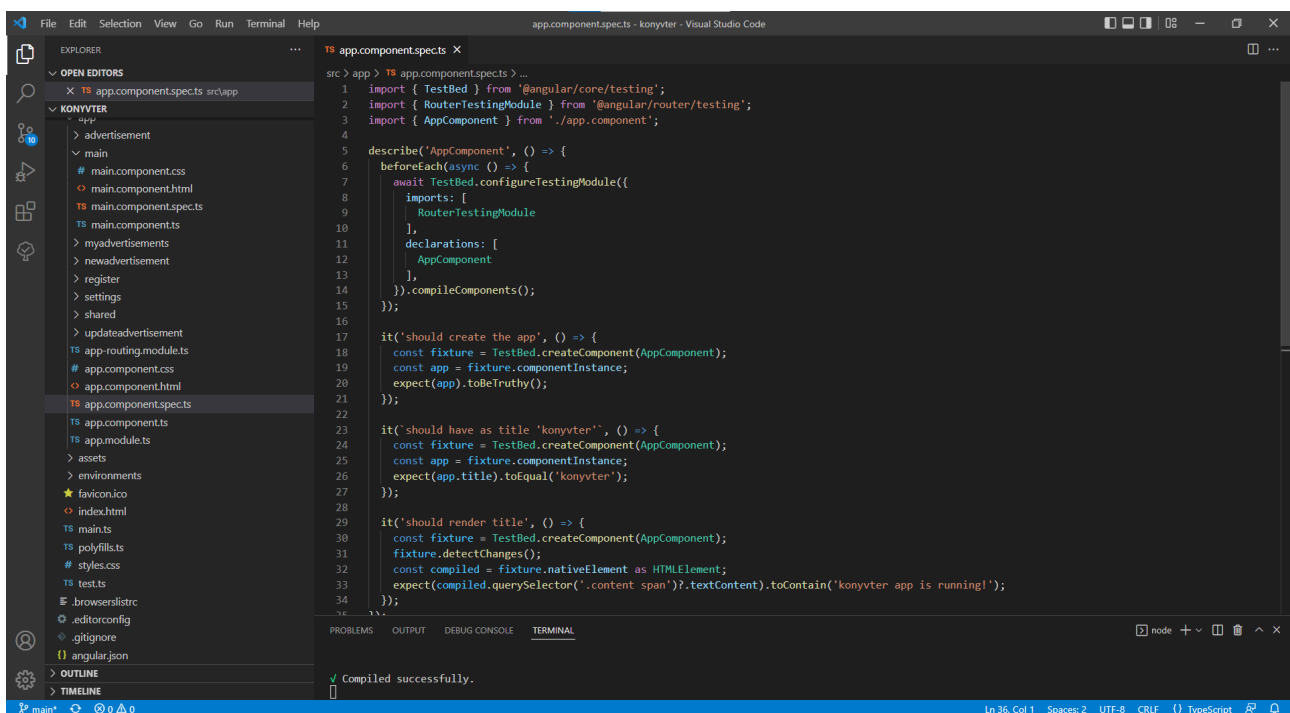
## A White Box

A *White Box* vagy magyarul *fehér doboz* tesztelési típus a programozási tudással rendelkező tesztelőknek kedvez hiszen így a kóddal együtt tudja megtekinteni a jelen esetben elkészült Frontendet, a helytelen szintaxisban meglévő hibákat, valamint a feleslegesen bonyolított vagy logikailag nem megfelelő kódrészeket könnyebben észrevehetjük.

## A Black Box

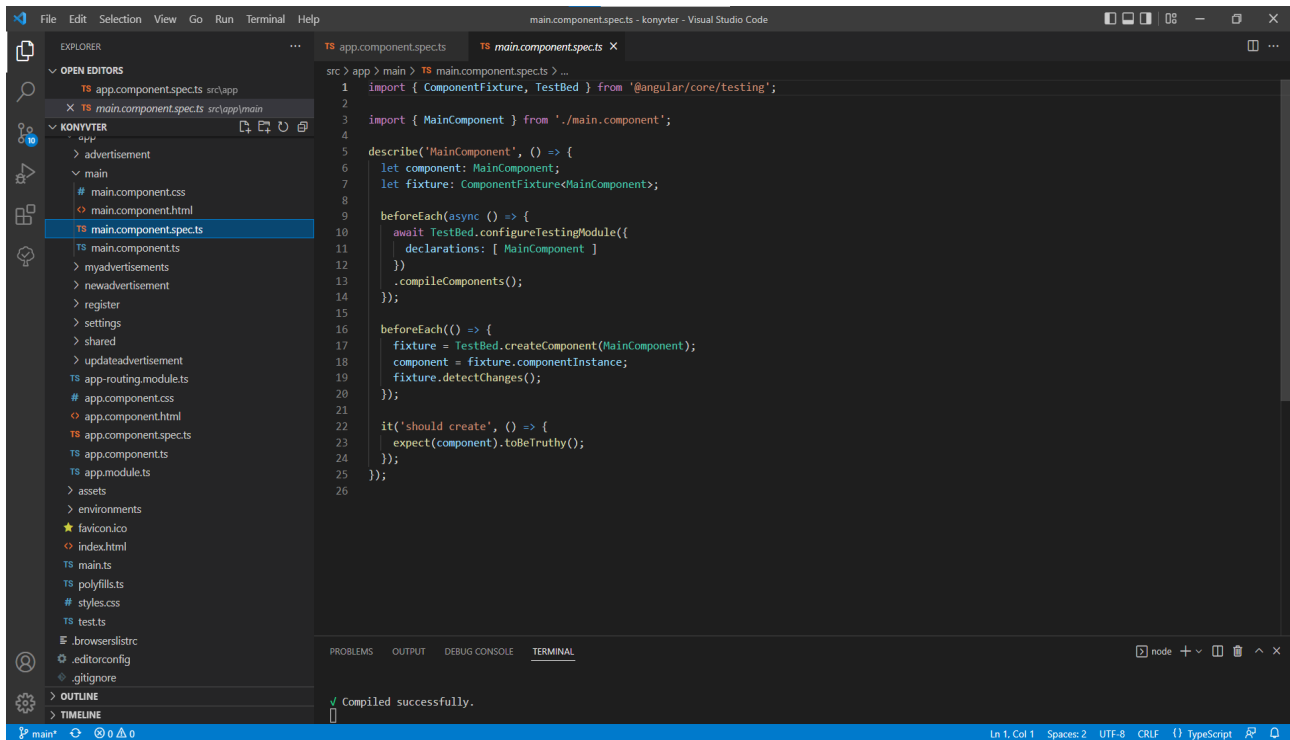
A *Black Box* vagy magyarul *fekete doboz* tesztelés a nevéből adódóan a forráskód nélküli tesztelésre szokták mondani, könnyebb ezzel a módszerrel nagyobb projekteket tesztelni, mivel nem kell a kódot is figyelni tesztelés közben, mivel nem szükséges programozási ismeret így szélesebb körben végezhető mint a White Box tesztmódszer.

Az app componentben több darab teszt is generálódik



```
src > app > TS app.component.spec.ts > ...
1 import { TestBed } from '@angular/core/testing';
2 import { RouterTestingModule } from '@angular/router/testing';
3 import { AppComponent } from './app.component';
4
5 describe('AppComponent', () => {
6   beforeEach(async () => {
7     await TestBed.configureTestingModule({
8       imports: [
9         RouterTestingModule
10      ],
11      declarations: [
12        AppComponent
13      ],
14    }).compileComponents();
15  });
16
17  it('should create the app', () => {
18    const fixture = TestBed.createComponent(AppComponent);
19    const app = fixture.componentInstance;
20    expect(app).toBeTruthy();
21  });
22
23  it('should have as title \'konyvter\'', () => {
24    const fixture = TestBed.createComponent(AppComponent);
25    const app = fixture.componentInstance;
26    expect(app.title).toEqual('konyvter');
27  });
28
29  it('should render title', () => {
30    const fixture = TestBed.createComponent(AppComponent);
31    fixture.detectChanges();
32    const compiled = fixture.nativeElement as HTMLElement;
33    expect(compiled.querySelector('.content span')?.textContent).toContain('konyvter app is running!');
34  });
35
36  ...
37
38  PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
39  ✓ Compiled successfully.
```

A többi componentben mindig csak egy darab teszt generálódik



The screenshot shows the Visual Studio Code interface with a file explorer on the left and a code editor in the center. The file explorer shows a project structure with a 'main' directory containing 'main.component.spec.ts'. The code editor displays the content of 'main.component.spec.ts', which is a TypeScript test file. The file starts with imports for 'ComponentFixture' and 'TestBed' from '@angular/core/testing', and 'MainComponent' from './main.component'. It then defines a 'describe' block for 'MainComponent' with a 'beforeEach' hook that configures the testing module and compiles the components. A 'beforeEach' hook is also defined to create the component and detect changes. Finally, an 'it' block is defined to test that the component is created successfully. The terminal at the bottom shows the message 'Compiled successfully.'

```
src > app > main > TS main.component.spec.ts > ...
1 import { ComponentFixture, TestBed } from '@angular/core/testing';
2
3 import { MainComponent } from './main.component';
4
5 describe('MainComponent', () => {
6   let component: MainComponent;
7   let fixture: ComponentFixture<MainComponent>;
8
9   beforeEach(async () => {
10     await TestBed.configureTestingModule({
11       declarations: [ MainComponent ]
12     })
13     .compileComponents();
14   });
15
16   beforeEach(() => {
17     fixture = TestBed.createComponent(MainComponent);
18     component = fixture.componentInstance;
19     fixture.detectChanges();
20   });
21
22   it('should create', () => {
23     expect(component).toBeTruthy();
24   });
25 });
26
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

node + - - ^ x

✓ Compiled successfully.

Ln 1, Col 1 Spaces: 2 UTF-8 CRLF {} TypeScript