Szoftverarchitektúrák Házi Feladat
Közösségi író-olvasó oldal
Dokumentáció

Készítették:
Buczny Dominik Hanich Péter
Németh Gergő Olivér Olchváry Ambrus

Konzulens:
Gazdi László

2024/2025/1

Tartalomjegyzék

1. A rendszer célja, funkciói, környezete.

1.1. Feladatkiírás

A feladat egy író-olvasó oldal elkészítése. Az oldalon a regisztrált felhasználók olvashatják egymás megosztott történeteit, azokhoz megjegyzéseket, kritikákat fűzhetnek. A történeteket legyen lehetőség gyűjteményekbe, a fejezeteket regényekbe szervezni. A feltöltött történetek minden esetben moderátori ellenőrzésen esnek át, csak ez után érhetőek el publikusan. A moderálás eredményéről a felhasználót mindenképpen értesíteni kell. A történeteket el lehet látni jellemzőkkel, illetve meg lehet jelölni a kategóriájukat, a benne szereplő karaktereket, valamint figyelmeztetéseket és korhatárt lehet rájuk beállítani. Ezen kívül a regisztrált felhasználóknak lehetőségük van egymással privát üzenetben kommunikálni. A rendszerhez webes és mobilos kliens készítése is szükséges. A részletes követelmények a *Specifikáció* dokumentumban találhatók. Mi a feladatkiírástól némileg eltérő nevezéktant használtunk: a továbbiekban a *történet* helyett a *mű* szót használjuk.

1.2. A rendszer által biztosított funkciók

A specifikáció alapján a platform a következő főbb funkciókat hivatott biztosítani. (Az egyes funkciók különböző szintű jogosultságokhoz kötöttek lehetnek.):

- Regisztráció
- Bejelentkezés
- Művekkel kapcsolatos funkciók:
  - Történetek böngészése
  - Történetek keresése
  - Történetek olvasása
  - Történetek létrehozása
  - Történetek szerkesztése, fejezetekre osztása
  - Történetek moderálása
- Gyűjteményekkel kapcsolatos funkciók:
  - Gyűjtemények böngészése
  - Gyűjtemények keresése
  - Gyűjtemények megtekintése
  - Gyűjtemények létrehozása
- Hozzászólás írása Művekhez vgy Gyűjteményekhez
- Művek, Gyűjtemények, Hozzászólások kedvelése
- Privát üzenetek küldése

A rendszer az alábbi szerepköröket különbözteti meg: látogató, regisztrált felhasználó, moderátor, adminisztrátor. A látogatók csak a megosztott tartalmakat tekinthetik meg, a regisztrált felhasználók létrehozhatnak saját tartalmat, kommentelhetnek, kedvelhetnek és üzeneteket küldhetnek. A moderátorok a moderálási jogosultságokkal rendelkeznek, az adminisztrátorok pedig a teljes rendszer felett rendelkeznek, kezelik a jogosultságokat.

1.3. A rendszer környezete

1.3.1. Backend

A backend a Laravel PHP keretrendszerrel [3] készült, mely egy MVC (Model-View-Controller) architektúrát követ. A megvalósítás során a Laravel 11-es verzióját használtuk. A backend szolgáltatásokat REST API-n keresztül érhetik el a kliensek, így lazán csatoltak a rendszerek, könnyű bővíteni, karbantartani őket.

1.3.2. Web kliens

1.3.3. Mobil kliens

A mobil kliens platformspecifikus, android operációs rendszerre készült kotlin nyelven Android Studioban. A felhasználói felület normál méretű mobiltelefonra lett optimalizálva, egyéb méretű eszközökön (pl. okosórán) nem lett tesztelve. A mobil kliens használatához internetelérés szükséges, hogy az alkalmazás elérje a backend szolgáltatásokat.

2. Megvalósítás

A szoftver három fő komponensből áll össze: a backend, a webes kliens és a mobil kliens. A hárokomponens klasszikus kliens-szerver architektúrát valósít meg: a backend a szerveroldali logikát, a webes és a mobilos kliens felhasználói felületet. A backend és a kliensek közötti kommunikáció http protokollon keresztül történik, a backend REST API-t biztosít a kliensek számára. Az egyes komponensek fejlesztését, külön külön végeztük, így a fejlesztőcsapat tagjainak jól elkülönülő feladata és felelősségi körük volt.

2.1. Architektúra

A szoftver egészében és komponenseiben is rétzegzett architektúrát valósít mmeg.

2.1.1. Backend architektúra

A Laravel MVC architektúrájának kifejtése...., magas szintű kép

2.1.2. Mobilos kliens

A mobilos kliens 2 fő rétegre osztható:

1. Adat réteg (Data Layer)
   - Adatlekérdezési réteg (HTTP kliens): REST API hívások implementációja, JSON objektumok kotlin osztályokra való leképezése.
   - Adatelérési réteg (Repository): hálózati kommunikáció és hibakezelés és egységesített kezelése, magasabb szintű kódban könnyebben használható.
2. UI réteg (UI Layer)
   - Állapotkezelés (View Model)
   - Megjelenítés (View)

2.2. Backend megvalósítása

2.2.1. Adathozzáférési réteg - DAL

Eloquent ORM: A backenden, a Laravel keretrendszerben az adathozzáférési réteget az Eloquent nevezetű ORM (Object Relational Mapper) biztosítja. Az Eloquent segítségével a PHP objektumokat és az adatbázis táblákat lehet egymáshoz kapcsolni. Az Eloquent ORM segítségével lehetőség van migrációk írására, modellek, seeder-ek, factory-k, stb. létrehozására.

Migrációk: A migrációs fájlok segítségével a táblák struktúráját lehet definiálni. Ez különösen hasznos a hordozhatóság szempontjából, mivel a migrációk segítségével a fejlesztők könnyen telepíthetik az alkalmazást a különböző adatbáziskezelőket használó környezetekbe. Mi a backend fejlesztése során a MySQL adatbázist használtunk.

Külön kiemelendő, hogy a Laravelben lehetőség nyílik UUID-k haználatára integer ID-k helyett amit ki is használtunk. Ennek előnye, hogy ez egy 36 hosszú alkalmazás szinten egyedi szöveges azonosítót generál minden rekordhoz, így kevésbé kikövetkezethető az adatbázis tartalma, mely nagyobb biztonságot nyújt. A globális egyediség az ütközések elkerülése végett is előnyös.

Modellek: A modellek az táblákat reprezentálják az alkalmazásban objektum orientált módon, ahol maga az osztály a tábla, az osztály példányok pedig a tábla sorai. A modellek segítségével lehetőség van az adatok lekérdezésére, frissítésére, törlésére, valamint az adatok közötti kapcsolatok definiálására, így az 1:1 kapcsolatok, 1:N kapcsolatok, N:M kapcsolatok is könnyen definiálhatóak.

Ezen kívűl a Like és Comment modelleknél a beépített Morph relációkat is használtuk, melyek segítségével egy táblához több másik tábla is kapcsolódhat (polymorphic relationship). Ehhez a Laravel minden táblához egy *_type* és egy *_id* mezőt ad hozzá (pl: `likeable_type` és `likeable_id`), melyek segítségével azonosítani lehet a kapcsolódó táblát és azon belül az egyedi azonosítót.

Az alkalmazásunkban található modellek áttekintő ábrája a 1. ábrán látható.

1. ábra. Az alkalmazás modelljei

Factory-k: A factory-k segítségével lehetőség van tesztadatok generálására az adatbázisba. A factory-k segítségével lehetőség van a modellekhez kapcsolódó adatok generálására, amelyeket a tesztelés során lehet használni. Ehhez a Laravel a Faker nevű könyvtárat használja, amely segítségével különböző típusú hamis, de struktúrális tekintetben helytálló adatokat lehet generálni.

Seeder-ek: A factory osztályok segítségével generált adatokat a seeder-ek segítségével lehet az adatbázisba betölteni, meghatározott mennyiségben és kapcsolatokkal. Külön jelentősége volt a `PermissionSeeder`-nek mely az alkalmazás jogosultságait definiálja.

2.2.2. Üzleti logika réteg - BLL

Az üzleti logikai réteg a Laravel keretrendszerben a vezérlőkön (Controller) és a hozzájuk kapcsolódó komponenseken keresztül valósul meg. Ez a réteg felel az alkalmazás alapvető működési logikájáért, amely az adatbázis-interakciók és a felhasználói kérések között helyezkedik el.

Vezérlők (Controllers) A vezérlők a Laravel MVC (Model-View-Controller) architektúrájának központi elemei, amelyek a beérkező HTTP-kéréseket kezelik és a megfelelő válaszokat generálják. A projektben mindegyik modellre készítettünk egy *<modell neve>Controller* névkonvenciójú vezérlő osztályt. A vezérlők a Laravel keretrendszerben a app/Http/Controllers könyvtárban találhatók. Minden vezérlő osztály a Laravel saját Controller bázisosztályból származik, amely alapvető funkciókat biztosít. A vezérlők metódusai a különböző CRUD HTTP-kérésekhez (GET, POST, PUT, DELETE) kapcsolódnak, és az útvonalak (routes) segítségével érhetők el.

Útvonalak és REST API (Routing and REST API) Az útvonalak határozzák meg, hogy a beérkező HTTP-kérések mely vezérlők mely metódusait hívják meg. A Laravelben az útvonalak a routes könyvtárban találhatók, és különböző fájlokban definiálhatók. Az alkalmazásunk útvonalai az api.php fájlban találhatóak, és alább látható belőle egy kódrészlet.

```
// api.php
Route::group(['middleware' => 'auth:sanctum'], function () {
Route::get('/user', function (Request $request) {
    return $request->user();
});
    Route::post('/works', [WorkController::class, 'store'])
                        ->can('create', Work::class);
    Route::put('/works/{work}', [WorkController::class, 'update'])
                        ->can('update', 'work');
    Route::delete('/works/{work}', [WorkController::class, 'destroy'])
                        ->can('delete', 'work');
});
Route::get('/works', [WorkController::class, 'index'])
                        ->can('view', Work::class);
Route::get('/works/{work}', [WorkController::class, 'show'])
                        ->can('view', Work::class);
```

E példában az útvonalak a WorkController különböző metódusaira irányítják a kéréseket, és az **auth:sanctum** middleware biztosítja, hogy csak hitelesített felhasználók férhessenek hozzá ezekhez az útvonalakhoz.

A REST API (Representational State Transfer Application Programming Interface) egy architektúrális stílus, amely lehetővé teszi az erőforrások HTTP protokollon keresztüli elérését és manipulálását. A Laravelben az útvonalak és vezérlők segítségével valósítható meg a REST API-k kialakítása.

Kérések (Requests) A Laravel Request osztálya biztosítja, hogy objektumorientált módon lehessen kezelni, az alkalmazás által aktuálisan kezelt HTTP-kérelmeket, valamint a kéréssel együtt elküldött cookie-k és fájlok lekérdezését. A HTTP kérések példányainak kiosztását DI (Dependency Injection) segítségével oldja meg a Laravel.

Válaszok (Responses) A Response példányok használata, lehetővé teszi a válasz HTTP státuszkódjának és fejlécének testreszabását. Minden modellhez tartozó Response példány a Laravel saját Response osztályából örököl, amely számos módszert biztosít a HTTP-válaszok testreszabására.

Autómatikus értesítés moderáció eredményéről A Laravel egyszerű módot biztosít az e-mailek küldésére mindenféle gond és nehézség nélkül a Mailable osztály használatával. A Laravel Mail API a népszerű Swiftmailer könyvtárra épült. Mi úgy konfiguráltuk, hogy SMTP-t használjon, de sok egyéb konfigurációval is lehet használni (Pl.: Mailgun, Amazon SES). A Mail konfigurációkat a backend projekt gyökérkönyvtárában megtalálható .env fájlban kell beállítani az alábbi módon:

```
MAIL_MAILER=smtp
MAIL_HOST=mailpit
MAIL_PORT=1025
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null
MAIL_FROM_ADDRESS="hello@example.com"
MAIL_FROM_NAME="${APP_NAME}"
```

Az alábbi parancs kiadásával tudjuk létrehozni a saját Mailable osztályunk és a hozzá tartozó email-levél sablon fájlunkat.

$ php artisan make:mail ModerationMail –markdown=emails.moderations

A ModerationMailable osztályból biztosítom a sablon számára a paramétereket, melyeknek értékeit megszeretnénk jeleníteni az email-ben. Ilyen paraméterek jelenleg a **work**, ami az adott munka objektumát reprezentálja, és a címét és az azonosítóját kérdezem le a sablonban. Valamint a **creator**, ami pedig az a User példány, amelyik létrehozta a munkát.

Miután elkészült a saját Mailable osztályunk és a hozzátartozó markdown sablont is megformáztuk, akkor már csak a kiküldés implementálása hiányzik. Erre használható a Laravel Mail Facade. Rengeteg metódussal rendelkezik, de a legfontosabb a **to** metódus. Ez a metódus határozza meg, hogy az email hova kerül elküldésre. Átadhatjuk a felhasználó email címét paraméterként, és a Laravel elküldi az emailt a megadott email címre. A következő fontos metódus a **send** metódus. Ez a metódus lehetővé teszi számunkra, hogy megadjunk egy Mailable üzenet példányt, tehát itt adtuk meg a korábban létrehozott Mailable osztályt. A levél elküldését a Work modell osztályának updating metódusában implementáltuk, mivel moderálást csakis munkákra lehet kérni, és amikor a ModerationStatus enumjának értéke APPROVED vagy EDIT_REQUESTED értékek valamelyikét veszi fel, akkor kell kiküldenünk az emailt.

A Laravel megkönnyíti az emailek tesztelését azáltal, hogy third-party könyvtárakat szolgáltat, mint például a Laravel mailpit, vagy a Mailtrap. Ezek a szolgáltatások egy hamis

SMTP szervert biztosítanak, ahonnan tesztelhetjük és ellenőrizhetjük, hogy az emailek helyesen vannake formázva. Értelemszerűen meg kell változtatni a mail konfigurációkat az .env fájlban, hogy valamely hamis SMTP szerver hitelesítő adatait használja. A Mailtrap használatával például vizuálisan láthatjuk, hogyan van formázva az email és emiatt döntöttünk mi is e mellett.

Authentikáció: Az authentikációra a Laravel Breeze [2] csomagot használtuk, mely egy minimális authentikációs rendszert biztosít. Az authentikációhoz szükséges routokat, controllereket, view-kat és middleware-eket a csomag automatikusan generálja. Az authentikációhoz szükséges adatbázis táblákat is a csomag generálja, így a felhasználók regisztrációja, bejelentkezése, jelszó visszaállítása, stb. egyszerűen megvalósítható.

A Breeze alapvetően Session alapú CSRF és CORS védelemmel rendelkező authentikációt biztosít a frontendek számára. Azonban maga a Breeze egy másik csomagra a Laravel Sanctumra [5] épít. Ennek előnye, hogy a Sanctum biztosít API Token alapú authentikációt is melyet a mobil kliens így ki tudott használni.

Authorizáció: Az authorizációra a Spatie cég által fejlesztett és karbantartott Laravel Permission csomagot [6] használtuk. A csomag segítségével lehetőség van jogosultságok (permissions) definiálására, azokhoz szerepkörök (roles) rendelésére, valamint a jogosultságok és szerepkörök alapján az authorizáció megvalósítására. A csomag a Laravel Policy-ket használja az authorizáció megvalósítására, amelyek segítségével a jogosultságokat és szerepköröket a modellekhez lehet rendelni.

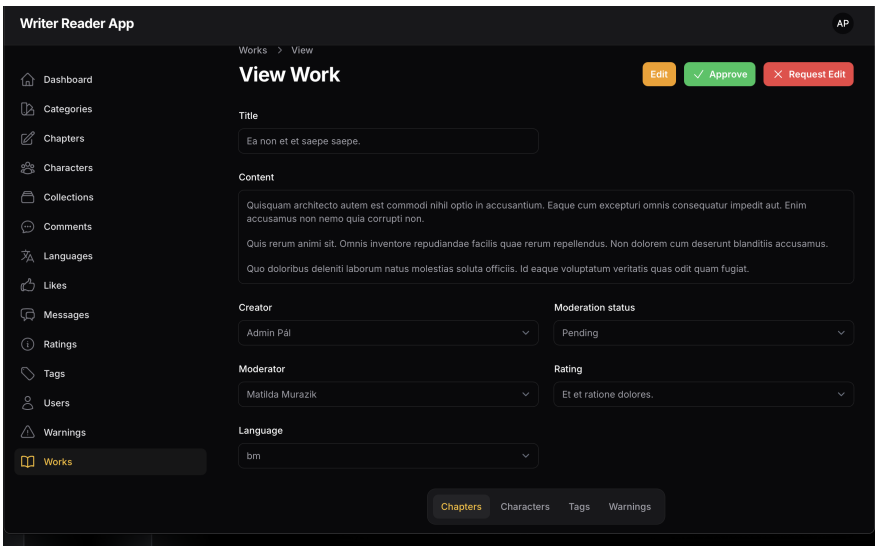
Ennek előnye hogy ellenőrzéskor csak a jogosultság meglétét kell ellenőrizni, azonban a jogosultságok egységesen kioszthatók role-ok segítségével. Az alkalmazás 3 szerepkört valósít meg, melyek a `Registered`, `Moderator` és `Admin`. Az admin az alkalmazás ServiceProvider-ében minden ellenőrzésen automatikusan átengedésre kerül, így nem kell az össze jogosultságot hozzárendelni.

2.2.3. Tesztelés

Teszteltünk, PEST, mekkora fedettség, mi lett tesztelve és miért

2.2.4. Admin felület

Bár nem külön alkalmazás réteg de megemlítendő, hogy az admin felület megvalósítására a FilamentPHP csomagot [1] használtuk a backend oldalon. Előnye, hogy a modell osztályokhoz definiálhatóak úgynevezett filament resource-ok melyek az admin felületen megjelenő mezőket és azok viselkedését definiálják. A csomag segítségével lehetőség van a modellek szerkesztésére, törlésére, listázására, stb. Az admin felület a 2. ábrán látható.



2. ábra. FilamentPHP admin felület

Az admin felülethez csak moderátor és admin jogosultságú felhasználó férhet hozzá. Azon belül az egyes modellek-hez való hozzáférést a policy osztályok szabályozzák de felülírható külön a filament resource-okon keresztül is.

Ezen kívűl az adminfelületen került megvalósításra a művek jóváhagyása is melyekre a moderátoroknak van jogosultságuk. A jóváhagyás után a művek publikussá válnak és megjelennek a felhasználók számára is, valamint automatikusan email értesítés küldődik a mű létrehozójának.

2.3. Mobil kliens megvaklósítása

Ebben a fejezetben a mobil kliens implementációjának részleteit mutatjuk be. Sajnos előre nem látható és technikai nehézségek miatt a megszabott határidőig nem sikerült a specifikációban előírt minden funkciót elkészíteni, itt csak a működő komponensek kerülnek bemutatásra.

A programban a következő főbb funkciók érhetők el:

- Regisztráció
- Bejelentkezés
- Művek böngészése
- Művek részleteinek megtekintése
- Művek olvasása
- Kedvelés küldése
- Hozzászólás írása
- korábbi privát üzenetek megtekintése

A mobilé architektúrája rétegekre bontható, ezek kifejtésében bemutatjuk nagyvonalakban a kód felépítését és a használt technológiákat. a használt androidos technológiákat.

2.3.1. Adatlekérdezés (HTTP kliens)

A mobil kliens a szerverrel való hálózati kommunikációt a Retrofit könyvtár segítségével valósítja meg. A Retrofit egy nyílt forráskódú Android és Java HTTP-kliens, amely a REST API-k hívását teszi lehetővé.

REST API interfész Az API-hívások kezelésére a `WriterReaderApi` interfész definiálja a szerver által támogatott végpontokat. Az egyes metódusok Retrofit-es annotációkkal vannak ellátva, amelyek a megfelelő HTTP metódusokat és a végpontokat definiálják. Az interfész nem tartalmazza az összes szerver által biztosított végpontot, hanem csak amelyek a mobil klens eddigi funkcióihoz szükségesek:

- **Művek kezelése:**
  - `getWorks`: Az összes mű lekérdezése.
  - `getWork`: Egy adott mű részleteinek lekérdezése azonosító alapján.
- **Felhasználók kezelése:**
  - `getUsers`: Az elérhető felhasználók listázása.
  - `getUser`: Egy adott felhasználó adatainak lekérdezése azonosító alapján.
  - `getCurrentUser`: Az aktuálisan bejelentkezett felhasználó adatainak lekérdezése token alapján.
- **Hozzászólások és kedvelések kezelése:**
  - `postComment`: Hozzászólás küldése.
  - `postLike`: Kedvelés hozzáadása.

  - `deleteLike`: Kedvelés törlése.

- **Hitelesítés:**
  - `register`: Új felhasználó regisztrációja.
  - `login`: Bejelentkezés a rendszerbe.
  - `logout`: Kijelentkezés az aktuális munkamenetből.

A HTTP kommunikációban JSON objektumokat használunk, melyeket kotlin osztályokra képződnek le A JSON objektumok és Kotlin osztályok közti átalakítást a **Moshi** könyvtár segítségével végezzük el.

Retrofit konfiguráció A REST API végpontok eléréséhez egy Retrofit klienst kell létrehozni, itt konfiguráljuk például az időtúllépési értékeket, a moshi JSON adaptert és az alap URL-t a szerverhez, ami jelenleg `http://10.0.2.2`, ez egy alias és az android emulátoron a host gép címét jelenti. A retrofit kliens a `WriterReaderApplication` osztályban az alkalmazáés indulásakor jön létre.

2.3.2. Adatelérés (Repository)

Az alkalmazás az adatelérésre egy a hálózati kommunikáció feletti absztrakciós réteget használ (**data access layer**), ami lényegeében egy `ApiMAnager` nevű wrapper osztály a Retrofit kliens és a hálózati hívások köré. Megkönnyíti a magasabb szintű kódból való használatot. AZ adatelérési réteg két fontosabb feladata az API-hívások kezelése és a hibakezelés. Az osztáy suspend fun metóduasi gondoskodnak arról, hogy a hálózati hívások csak corutineokon belül történjenek, így ne blokkolják a UI szálat,

Hibakezelés A válaszok kezeléséhez a `Retrofit Response` osztályt használja, amely tartalmazza a HTTP státuszkódot és a szerver által küldött adatokat. Minden API-hívás esetén figyelembe vesszük a lehetséges hibákat, mint például hálózati időtúllépést vagy nem várt szerverhibákat, hogy megfelelő visszajelzést adhassunk a felhasználónak. Az `ApiMAnager` metódusai onSuccess és onError callback függvényeket várnak paraméterként, amelyek a hálózati hívások végrehajtásakor hívódnak meg.

Az `ApiMAnager` osztályt az alkalmazás indulásakor pédányosítjuk így az egész alkalmazásban elérhető lesz.

2.3.3. UI réteg (UI Layer)

A alkalmazás egyes képernyőit MVI (Model-View-Intent) architektúra szerint valósítottuk meg. Ennek az architektúrának 4 komponense van

- **Model**
- **Intent**
- **View**
- **View-Model**

Model A Model az alkalmazás állapotának (State) egy reprezentációja. Ez a komponens tartalmazza a képernyőhöz szükséges összes adatot és a felhasználói interakciók eredményét. Az állapot megváltozása új state objektum létrehozásával történik.

Intent Az Intent-ek képviselik a felhasználói interakciókat és a képernyő eseményeit (például: egy gomb lenyomása). Az Intent-ek explicit módon jelzik a ViewModel-nek, hogy milyen műveleteket kell végrehajtania.

View A View a Jetpack Compose deklaratív UI komponensei által megvalósított felhasználói felület. Az composable elemek kizárólag a State alapján frissülnek. A View nem tartalmaz logikát, hanem kizárólag az állapot megjelenítéséért felelős.

View-Model A ViewModel felel az Intent-ek feldolgozásáért és az állapot frissítéséért. Ez a komponens biztosítja a State folytonosságát és kezeli az üzleti logikát, például hálózati kérések indítását. A ViewModel figyeli az Intent-eket, végrehajtja a szükséges műveleteket, és az új állapotot a View felé közvetíti.

2.3.4. Grafikus felhasználói felület

3. Telepítési lerírás

3.1. Backend telepítése

3.1.1. Fejlesztői környezet telepítése

A backend fejlesztői környezet telepítéséhez szükséges a PHP és a Composer csomag kezelő telepítése. Az alkalmazás egyéb függőségei (adatbázis, stb) Docker kontérekekben futnak, így a Docker telepítése szükséges. A Docker telepítéséhez a `https://docs.docker.com/get-docker/` címről tölthető le a telepítő. A Composer telepítéséhez a `https://getcomposer.org/download/` címről tölthető le a telepítő. A Composer telepítése után a backend mappában a következő parancsot kell futtatni: `composer install`. Ezzel a Composer telepíti a Laravel függőségeket.

A backend mappában a `.env.example` fájlt le kell másolni a `.env`-be. Ebben a fájlban kell beállítani az alkalmazás konfigurációját (adatbázis, frontend_url, stb).

Ezután a `docker-compose.yaml` fájl segítségével indítható a backend. Ennek megkönnyítésére mi a Laravel Sail csomagot használtuk mely egy CLI a különböző konténerek kezelésére (lásd [4]). Indítás után szükséges az alkalmazás egyedi kulcsának generálása (`php artisan key:generate`, `sail` esetén a `php` kulcsszó minden további parancsban helyettesítendő `sail`-el), az adatbázis migrálása (`php artisan migrate`) és a jogosultságok inicializálása (`php artisan db:seed -class=PermissionSeeder`). Ezek után a backend elérhető a `http://localhost:8000` címen.

3.1.2. Éles környezet telepítése

Az éles környezetben való telepítéshez lásd a Laravel hivatalos dokumentációját: `https://laravel.com/docs/11.x/deployment`. Ezen kívűl az email küldés funkcióhoz szükséges egy SMTP szerver és annak konfigurációjának beállítása a `.env` fájlban.

3.2. Android kliens telepítése

A projekt továbbfejlesztéséhez vagy szerkesztéséhez Android stúdió telepítése szükséges. Az Android Studio letölthető a `https://developer.android.com/studio` címről. Az android studió tartalmazza az Android SDK-t ami a kód fordításához szükséges. A kód fordításához szükséges minimális SDK verziószáma: 24. Az alkalmazás futtatásához szükséges egy Android eszköz vagy emulátor. Emulátor szintén telepíthető az Android studióban.

Ha kiadásra szánjuk az alkalmazást, akkor le kell fordítani a brojektet és APK filet kell generálni. Az APK-t aláírással kell ellátni, majd feltölthető a Google Play áruházba. Innen a felhasználók letölthetik és telepíthetik az alkalmazást.

3.3. Webes frontend telepítése

A buildeléshez és fejlesztői módban való futtatáshoz npm szükséges. A webes gyökérmappában (writer-reader-web-client) konzolon kiadva az `npm install` parancsot, beszerzi a szükséges JavaScript csomagokat. Fejlesztői környezetben, a `npm run dev` parancsot kiadva elindíthatjuk, jelenleg a localhost 3000-es portján. A production verziót buildelni a `npm run build`-et használva tudjuk a writer-reader-web-client/dist alkönyvtárba. Ezután hostolható egy szerveren. `https://vite.dev/guide/static-deploy.html`

4. A program készítése során felhasznált eszközök

4.1. Backend szoftverek

- Laravel 11, PHP 8.2, Composer
- Package-k:
  - FilamentPHP: admin felület,
  - Laravel-Permissions: jogosultság kezelés,
  - Laravel Breeze: authentikáció,
  - Laravel Sail: Docker konténerek kezelése
  - Pest: PHP tesztelés
- Docker (Desktop): konténerizáció
- MySQL: adatbázis (konténerizálva)
- MailPit: mail küldés tesztelése (konténerizálva)
- PhpStorm IDE: fejlesztői környezet

4.2. Webes szoftverek

- Vite 5.4 + React 18.3 (JSX)
- npm: JavaScript csomagkezelő
- Package-k:
  - React Router: navigáció
  - MUI: UI React komponensek
  - Axios: kérések
  - js-cookie: sütik kezelése
  - mui-markdown: markdown megjelenítése
  - react-md-editor: markdown szerkesztő
  - ESLint: kód analízis
- Firefox, Chrome, MS Edge: böngészők
- Visual Studio Code: fejlesztői környezet

5. Összefoglalás

6. Továbbfejlesztési lehetőségek

Hivatkozások

[1] Filament. Filament. https://filamentphp.com/docs.

[2] Laravel. Laravel breeze. https://laravel.com/docs/11.x/starter-kits#laravel-breeze.

[3] Laravel. Laravel documentation. https://laravel.com/docs/11.x.

[4] Laravel. Laravel sail. https://laravel.com/docs/11.x/sail.

[5] Laravel. Laravel sanctum. https://laravel.com/docs/11.x/sanctum.

[6] Spatie. Laravel permissions. https://spatie.be/docs/laravel-permission/v6/introduction.