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

2.2. Backend megvalósítása

2.2.1. Adathozzáférési réteg - DAL

Eloquent ORM + MySQL adatbázis, lehet írni migrációkról, modellek, seeder-ek, factory-k, stb

2.2.2. Üzleti logika réteg - BLL

Controllerek, Routing + Rest API, Auth middleware, Mail értesítés, Request validáció, Data Transfer Object-ek (Resource osztályok),...

2.2.3. Tesztelés

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

2.2.4. Admin felület

nem külön réteg de jó ha itt van, kép jöhet

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 (Data Query Layer)

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 (Data Access Layer)

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)

3. Telepítési lerírás

4. A program készítése során felhasznált eszközök

4.1. Backend szoftverek

lásd üzi...

5. Összefoglalás

6. Továbbfejlesztési lehetőségek