XIX. reál- és humántudományi Erdélyi Tudományos Diákköri Konferencia (ETDK)
Kolozsvár, 2016. május 19–22.

Legendárium Navigátor:
szoftverrendszer a Székelyföldi Legendárium számára

Szerzők:
Máté Attila Barna
Babeş-Bolyai Tudományegyetem, Kolozsvár, Matematika és Informatika Kar, informatika szak, III. év
Nagy Roland
Babeş-Bolyai Tudományegyetem, Kolozsvár, Matematika és Informatika Kar, informatika szak, III. év

Témavezetők:
dr. Simon Károly, egyetemi adjunktus,
Babeş-Bolyai Tudományegyetem, Matematika és Informatika Kar
Kelemen Bálint, szoftverfejlesztő, Codespring
Szécsi Zsolt, szoftverfejlesztő, Codespring

Kivonat

A dolgozatban bemutatott LegendáriumNavigátor projekt célja egy turisztikai alkalmazás létrehozása, amely interaktív segítséget nyújt felhasználóinak a székelyföldi legendákkal kapcsolatban. A szoftver fejlesztése együttműködésben történt a Székelyföldi Legendárium csapatával, ők segítettek a koncepció kidolgozásában és szolgáltatták a szükséges adatokat. A rendszer részét képezi egy szerver alkalmazás, egy webes felület, amely adminisztrációs felületként szolgál a legendákhoz kapcsolódó tartalmakat feltöltéséhez és szerkesztéséhez, illetve egy telefonos alkalmazás, amely megjeleníti ezeket a tartalmakat, tájékoztatja a felhasználót a közelében levő legendákról és a helyszínre navigálja őket. A legendákhoz a szöveges leírásokon kívül más média-tartalmak (hangos könyvek, képek és video anyagok) is csatolhatóak. Az alkalmazás egy térképen jeleníti meg a legendák helyszíneit és ennek alapján ad útbaigazítást a turistáknak. Alkalmas továbbá kirándulási útvonalak megtervezésére a legendák helyszíneinek útbaejtésével, valamint a felhasználó böngészheti a helyszínekhez köthető eseményekkel kapcsolatos hírfolyamot is.

A dolgozat a rendszer szerkezetét írja le, kitér a megvalósítás részleteire, érintve a felhasznált módszereket, technológiákat és eszközöket, valamint bemutatja a szoftver működését.

Tartalomjegyzék

Kivonat 2

Bevezető 5

1 A LegendáriumNavigátor projekt 6

1.1 Alapvető funkcionalitások 6

1.2 Architektúra 7

2 Felhasznált technológiák 9

2.1 Szerver oldali technologiak 9

2.1.1 Spring keretrendszerek 9

2.1.2 RESTful webszolgáltatások 10

2.1.3 Swagger 10

2.2 Android kliens oldali technológiák 11

2.2.1 Retrofit 12

2.2.2 Butterknife 12

2.2.3 OrmLite 12

2.2.4 Icepick 13

2.2.5 Térképszolgáltatások 13

2.2.6 Picasso 14

2.3 Kliens oldali webes technológiák 14

2.4 További technológiák 15

3 A rendszer megvalósításának összefoglalója 17

3.1 A szerver megvalósítása 17

3.2 Az Android kliens alkalmazás megvalósítása 18

4 Alkalmazott módszerek és felhasznált eszközök 19

5 A Legendárium Navigátor működése 21

5.1 Az Android kliens alkalmazás használata 21

5.2 A Webes adminisztrációs felület használata 25

Következtetések és továbbfejlesztési lehetőségek 27

Hivatkozások 28

Bevezető

A „Székelyföldi Legendárium” egy székelyföldi projekt, amely 2008-ban indult és fő célja, hogy helyi legendákat gyűjtsön össze (eddig 156-ot sikerült) Erdély területén. A projekt keretein belül már megjelent kifestős könyv, kirakós- és társasjáték, könyv, illetve egy 3D-s rajzfilmsorozat első része is.

A Legendárium Navigátor projekt ötletgazdája Fazakas Szabolcs, a „Székelyföldi Legendárium” vezetője, aki azzal kereste meg a Codespringet, hogy szeretnének egy telefonos alkalmazást, amely megkönnyíti a turisták számára a legendák helyszíneinek megtalálását és információt ad a legendákkal kapcsolatos fontosabb tudnivalókról, illetve a legendák helyszínein szervezett rendezvényekről.

Az alkalmazás funkcionalitásait egy példán keresztül lehetne a legjobban bemutatni: egy kiránduló család érkezik Székelyföldre, aki használja a Legendárium Navigátor mobilalkalmazást. Az applikáció segítségével elolvashatják a legendákat, megtekinthetik a csatolt képeket, hanganyagokat hallgathatnak meg (például az utazás közben hangos könyveket hallgathatnak). A térkép segítségével a felhasználók betájolhatják a legenda pontos helyét és kérhetnek navigálást a rendszertől. Az alkalmazás értesítést is küld felhasználójának, ha egy legenda helyszínének közelébe érkezik. Továbbá, lehetőséget ad a legendákkal kapcsolatos tartalmak letöltésére, így nem szükséges a folyamatos internet kapcsolat. Ez fontos tulajdonsága az applikációnak, mivel vannak olyan legenda helyszínek, ahol nincs mobil internet hozzáférés. A felhasználó böngészheti az alkalmazáson belüli hírfolyamot is, ahol friss információkhoz juthat a legendákhoz, illetve azok helyszíneihez köthető közelgő eseményekről.

A Legendárium Navigátor projektnek van egy webes felülete, ahol a legendáriumos munkatársak könnyen feltölthetnek, menedzselhetnek legendákkal kapcsolatos tartalmakat. Lehetőség van a már meglévő, Legendárium weboldalt kiszolgáló szerverről az adatok (megfelelő formátumú) importálására is. Mindezeknek a műveleteknek a hátterében a Legendárium Navigátor szerver található.

A projekt fejlesztése a Codespring Mentorprogram részeként, a szakmai gyakorlat ideje alatt kezdődött el, majd a „Csoportos projekt” tantárgy keretein belül folytatódott, ahol ideiglenesen három új taggal is kibővült a csapat. Kudor Róbert, Magdás Walter és Márton Zete-Örs egy egyetemi félév ideéig segítettek be a szerzőknek a fejlesztésbe. A projekt létrejöttéért köszönet illeti dr. Simon Károly projektmenedzsert, valamint Kelemen-Fehér Dénes-Bálint és Szécsi Zsolt mentorainkat, akik a fejlesztési folyamatot irányították. Továbbá Fazakas Szabolcsot és a Székelyföldi Legendárium munkatársait, akikkel folyamatos volt a konzultáció az applikációt illetően.

1 A LegendáriumNavigátor projekt

A következő részben a projekt fontosabb funkcionalitásai lesznek összefoglalva, illetve a projekt architektúrája lesz röviden bemutatva.

1.1 Alapvető funkcionalitások

A projekt két nagy részre osztható: a szerver és a kliens oldalra. A kliens oldali rész tartalmaz egy webes és egy mobil kliens alkalmazást.

A szerver felelős az alkalmazás üzleti logikájáért (business logic) és RESTful API-t biztosít az adatok menedzselésére. A szerver a következő funkcionalitásokat nyújtja:

- a felhasználók beléptetése és jogköröknek megfelelő funkciók biztosítása;
- az adatok perzisztenciájának megvalósítása;
- az adatokkal kapcsolatos műveletek végrehajtása;
- már meglévő külső adatbázisból exportált legendák adatainak importálása, a média tartalmakkal együtt;
- a szerver oldal szolgáltatásainak publikálása egy RESTful API-n keresztül, a kliensekkel történő kommunikációval kapcsolatos műveletek megvalósítása.

Kliens oldalon jelennek meg a legendákhoz tartozó tartalmak, valamint a webes felület esetében elérhetőek az adminisztrátor számára biztosított funkciók. Az adatokat a kliensek a szerver oldalon lévő MySQL adatbázisból kapják, a szerverrel RESTful webszolgáltatásokon keresztül kommunikálnak.

A **Legendárium Navigátor webes felhasználói felülete** lehetőséget nyújt:

- új legendáriumos munkatárs beléptetésére;
- új legendák létrehozására;
- már meglévő legendák szerkesztésére és törlésére;
- a legendákkal kapcsolatos média tartalmak feltöltésére és menedzselésére;
- a hírfolyamban megjelenő tartalom menedzselésére;

- legendák importálására külső adatbázisból;
- az importálás során keletkezett konfliktusok kezelésére (amennyiben egy adott importálandó legenda már szerepel a rendszer adatbázisában).

A **Legendárium Navigátor webes adminisztrációs felülete** lehetőséget nyújt:

- legendáriumos munkatársak regisztrálására;
- API hívások indítására Swagger segítségével;
- működési statisztikák megjelenítésére;
- naplózási szintek beállítására;
- szolgáltatások aktuális állapotának a megjelenítésére (adatbázis, e-mail);

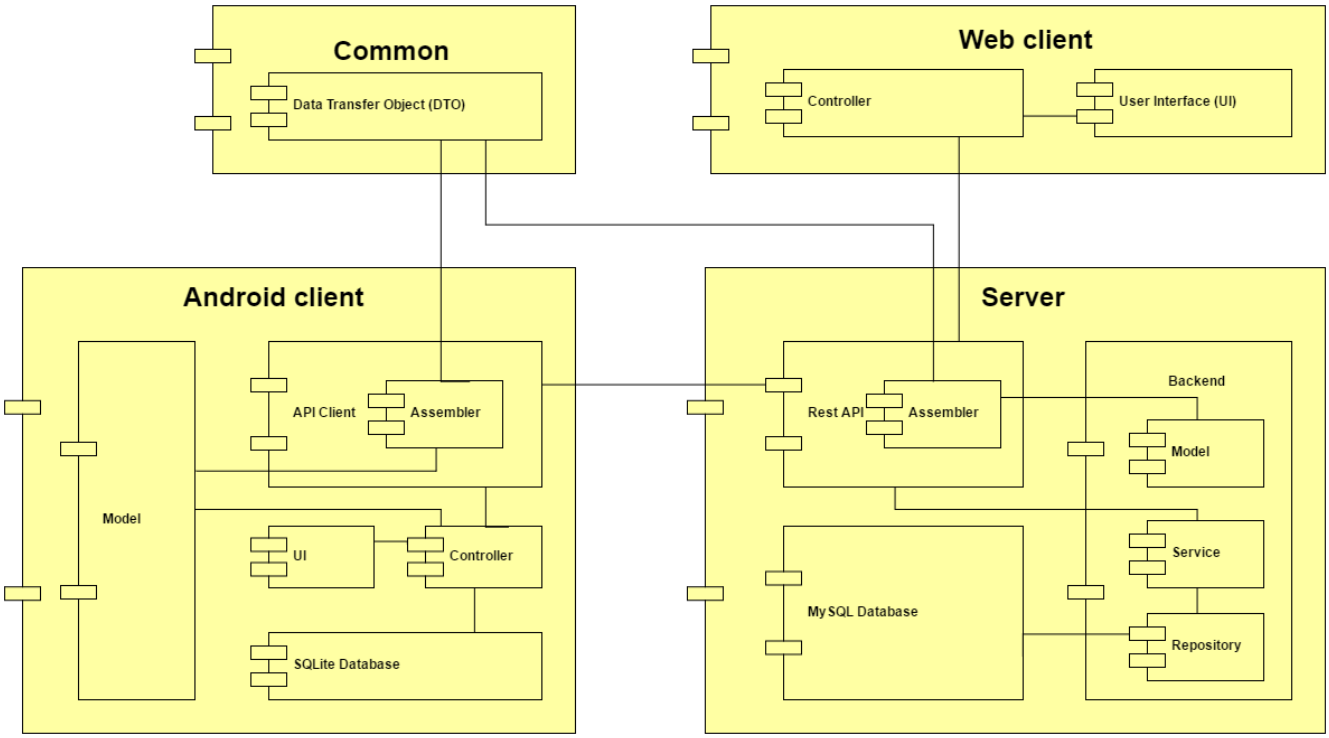
Továbbá, a webes adminisztrációs felület rendelkezik a legendáriumos munkatársak számára biztosított webes felhasználói felület minden funkcionalitásával.

A **Legendárium Navigátor mobil alkalmazás** lehetőséget nyújt:

- felhasználók regisztrációjára és bejelentkezésre;
- legendák megjelenítésére és letöltésére (beleértve a média tartalmakat);
- letöltött legendák lokálisan mentett adatainak törlésére;
- legendák térképen való megjelenítésére;
- navigáció kérésére adott legenda helyszínéhez;
- értesítések megjelenítésére a legendahelyszínek közelében;
- hanganyagok lejátszására;
- útvonaltervezésre;
- hírfolyam megjelenítésére.

1.2 Architektúra

Architektúra szemontjából a projekt két nagy komponensre osztható: szerver és Android kliens. A szerver oldal újabb három komponensre bontható: itt található a backend, az API és a webes felület.



1. *ábra:* A Legendárium Navigátor rendszer architektúrája

Szerver oldalon az alkalmazás központi entitásai a Model csomagban lévő osztályok által vannak reprezentálva. Ezeket az entitásokat közvetlenül használja a szerver másik három alrendszere: a Backend, a Web, valamint az API. Az adatok tárolására MySQL relációs adatbázist használ a rendszer.

A Backend tartalmazza a Service és a Repository komponenseket. A szolgáltatás réteg (Service) interfészeken keresztül továbbítja az adatokat a REST API felé. Az adatok eléréséhez szintén interfészeken keresztül kommunikál a repository komponensekkel. A Repository rétegben az adatok lekérdezése, beszúrása, módosítása és törlése van megvalósítva.

Az alrendszerek közötti kommunikációt a szerver oldalon elhelyezkedő API komponens biztosítja. Az API garantálja, hogy megfelelő kérésekre minden esetben megfelelő, helyes válaszok érkezzenek. Az alkalmazás az API hívások során DTO objektumokkal dolgozik, az adatátvitel JSON formátumban történik. A DTO-k oldják meg az alrendszerek adatmodelljei közötti adatreprezentációs különbségek kezelését. Mind szerver, mind kliens oldalon megtalálhatóak az Assemblerek, amelyek a DTO-kat átalakítják a megfelelő modell osztályokba és fordítva.

A mobil kliens az adatokat SQLite adatbázisban tárolja. A Controllerek felelősek a vezérlését, valamint a kliens oldali alkalmazáslogikával kapcsolatos műveletek megvalósításáért. A kliens oldali alkalmazás UI komponesei által egy könnyen használható felhasználói felületet biztosít.

2 Felhasznált technológiák

Az alábbiakban a Legendárium Navigátor megvalósítása során használt fontosabb technologiak lesznek bemutatva.

2.1 Szerver oldali technologiak

A Legendárium Navigátor szerver oldalának váza a JHipster eszköz segítségével volt létrehozva, amely biztosított a fejlesztők számára egy működő, Spring Boot-ra épülő szervert, amely egy MySQL adatbázist használ.

2.1.1 Spring keretrendszerek

A Spring [4][12] leginkább Java-alapú enterprise alkalmazások készítésére használt keretrendszer. Egy Inversion of Control (IoC) konténeren keresztül biztosítja a komponensek menedzsmentjét és a Dependency Injection (DI) minta alkalmazásának lehetőségét.

Rod B. Johnson, egy ausztrál származású informatikus fejlesztette ki, aki társalapítója volt a SpringSource vállalatnak és CEO-ként volt alkalmazva egészen 2009-ig, amikor a VMware felvásárolta a vállalatot. A Spring keretrendszer megkönnyíti a programozók munkáját és könnyen karbantarthatóvá teszi a projekteket, a modulok közötti szoros függőségek feloldása által.

A DI minta alkalmazhatósága által lehetőséget nyújt a függőségek meghatározására (XML fájlok, Java konfigurációs osztályok, illetve annotációk segítségével). A Spring Bean-ek egy standard mechanizmust biztosítanak a DI megvalósítására, konstruktorokon vagy setter metódusokon keresztül.

A Legendárium Navigátor a Spring Core-on (IoC konténer) kívül használja a Spring Boot (konfiguráció), Spring Security (biztonság), Spring Data JPA (perzisztencia) és Spring MVC REST (webszolágltatások) modulokat.

A Spring Security biztosít egy Oauth2-es szabványra épülő bejelentkeztetést. Bejelentkezéskor a rendszer, ha helyesnek nyilvánítja a megadott felhasználó és jelszó párost, akkor egy úgynevezett access token-t és refresh token-t térít vissza. Ezután ezt az access token-t fogja minden kérésnél használni a felhasználó, ha azonosítani szeretné magát a szerveren. A refresh token akkor lesz használva, amikor lejár az access token érvényességi ideje. A refresh token segítségével a program új access token-t tud igényelni, és így a felhasználót és jelszót kevesebb alkalommal kell a hálózaton küldeni.

A Spring Data JPA keretrendszer egy JPA fölötti absztrakciós szintet biztosít (háttérben a Hibernate ORM keretrendszer szolgál JPA implementációként), és lehetővé teszi az adatbázis könnyű

cserélhetőségét, anélkül, hogy a felette levő rétegeket változtatni kellene. A keretrendszernek köszönhetően a repository komponenseket nem kell feltétlenül implementálni, elegendő az interfészeket létrehozni, a megfelelő elnevezési konvenciók alkalmazásával. Bonyolultabb lekérdezések is megadhatóak az interfészeken belül, speciális annotációkkal együtt alkalmazott JPQL query-k által. Természetesen, specifikusabb esetekben a komponensek implementációjára is lehetőség van, a háttérben működő JPA EntityManager minden funkcionalitása elérhető.

A Spring MVC REST keretrendszer leegyszerűsíti a RESTful webszolágltatások elkészítését. A REST erőforrások Spring komponensek, amelyek metódusai speciális annotációkkal vannak ellátva. Ezeknek az annotációknak a segítségével lehet meghatározni, hogy milyen típusú kéréseket szolgál ki az illető metódus, beleértve a szabványos REST URI-t és egyéb paramétereket.

2.1.2 RESTful webszolgáltatások

A Legendárium Navigátor-on belül a szerverrel való kommunikáció a REST architektúrán alapszik, RESTful web-szolgáltatásokon keresztül valósul meg.

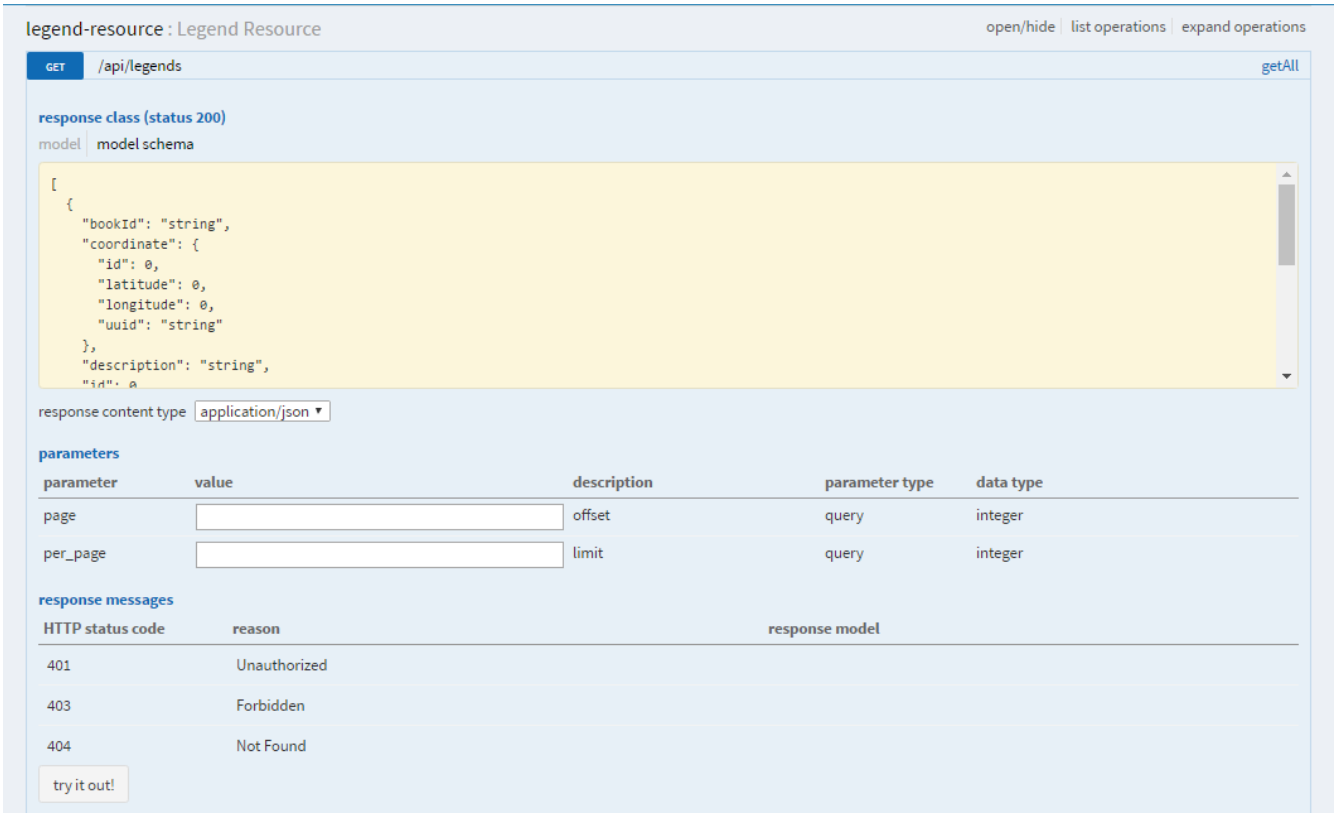
Az adatok küldése a kliens és szerver között JSON formátumú objektumok segítségével történik. A JSON szerializációban és deszerializációban a Jackson keretrendszer nyújt segítséget, amely a JAX-B (Java Architecture for XML Binding) implementációja.

A REST egy architekturális minta kliens-szerver kommunikáció megvalósítására, amely HTTP analógiára épít. A fejlesztők a szerver oldalon bizonyos erőforrásokat tesznek elérhetővé a kliens alkalmazások számára. A kliens alkalmazások adott konvenciónak megfelelő URI-k alapján, a HTTP kérések típusainak megfelelő egyszerű műveletek (GET, POST, PUT, DELETE stb.) használatával férhetnek hozzá az erőforrásokhoz.

2.1.3 Swagger

A Swagger egy leírást biztosít a REST API-hoz, amelynek formátuma adott szabályokra épül. Ez a formátum gép és ember számára is olvasható.

A SwaggerUI láthatóvá teszi a felhasználó számára a REST erőforrásokat egy webes felhasználói felületen, megkönnyítve a tesztelést és a fejlesztést. Az említett felhasználói felület automatikusan generálódik, Swagger annotációk alapján.



2. ***ábra:*** A Swagger webes felhasználói felülete

A SwaggerUI segítségével API hívásokat lehet indítani, és a felületen a szerver kérésekre adott válaszai is láthatóak. A felület új kliens alkalmazások fejlesztői számára pontos leírást ad a létező erőforrásokról, így az API használatához nem szükséges a szerver belső felépítésének, vagy más kliens alkalmazások működésének az ismerete. A Swagger tehát egy hasznos eszközt biztosít, mind a tesztelés, mind a dokumentálás szempontjából.

2.2 Android kliens oldali technológiák

Az Android egy Linux kernelre épülő, főként mobileszközökön használt operációs rendszer. A platform többrétegű architektúrával rendelkezik. A legalsó rétegen található a Linux kernel, a következő rétegen találhatóak a programkönyvtárak és szolgáltatások (libc, OpenGL, SSL stb.); ezek C-C++-ban vannak megvalósítva, és közvetlenül a Linux kernelen futnak. Ezekre a könyvtárakra épül az Android futtatókörnyezet, amelynek alapja a Dalvik virtuális gép volt, de az Android 5.0-tól be van vezetve az ART is, amely ahead-of-time kompilálást végez, így növelheti az applikáció teljesítményét. Ezen kívül az ART szigorúbb telepítési időben történő verifikációt végez, mint a Dalvik. A legfelső szinten találjuk a Java-ban megírt applikációkat.

Ahhoz, hogy az Android platformra fejlesszünk, szükségünk van az Android szoftverfejlesztői csomagra (Android SDK), amely számos fejlesztési eszközt tartalmaz (könyvtárak, dokumentumok, példák, emulátor stb.).

2.2.1 Retrofit

A Retrofit egy REST kliens csomag Androidra, amelyet a Square csapata fejleszt. Jól dokumentált, sok hasznos tulajdonsággal, segítségével egyszerűen tudunk küldeni GET, PUT, POST, DELETE kéréseket a szervernek.

A technológia központi eleme a RestAdapter osztály, amely Gson annotációkat használ az adatok szerializálásához és deszerializálásához. A Legendárium Navigátor esetében az adatok JSON formátumban érkeznek, DTO-kba vannak deszerializálva, majd a DTO-kat Assembler-ek alakítják át modell objektumokba. A kéréseknek megfelelő URL-eket speciális interfészekben adhatjuk meg.

2.2.2 Butterknife

A Butterknife adattagok és metódusok Android nézetekhez való hozzákapcsolását (binding) valósítja meg. Annotációkat használ, hogy olyan kódrészleteket generáljon, amelyeket sok helyen fel lehet használni kevés vagy semmilyen változtatással. Például:

- kiküszöbölhetőek a *findViewById* metódushívások, a *@Bind* annotáció segítségével;
- eltávolíthatóak a figyelők (listener) megvalósítására használt névtelen belső osztályok, például az *@OnClick* annotációt használva a metódusokon.

2.2.3 OrmLite

Az Orm-Lite egy pehelysúlyú keretrendszer, amely Java objektumokat képez le relációs adatbázisok sémáinak megfelelően, egyszerű annotációk alapján (pl: *@DatabaseTable*, *@Databasefield* stb.). A Legendárum Navigátor ennek a keretrendszernek a segítségével oldja meg a legendákkal kapcsolatos tartalmaknak az elmentését és menedzselését a telefonon. Ennek célja, hogy a felhasználónak ne legyen szüksége folyamatos internethozzáférésre a tartalmak böngészéséhez.

2.2.4 Icepick

Az Icepick egy olyan könyvtár, amely kiküszöböli azokat a kódrészleteket, amelyek sokszor jelennének meg ugyanolyan formában a "bundle[1]"-ök elmentésekor és visszanyerésekor. Mivel a bundle-ök alkalmazása egy alapvető mechanizmus az Activity példányok közötti adatátvitel megvalósítására, az Icepick alkalmazása sok redundáns kódrészletet kiküszöbölhet. Ezt annotációk segítségével oldja meg, például:

```
public class ExampleActivity extends Activity {
@State
String username;

@Override public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      Icepick.restoreInstanceState(this, savedInstanceState);
}

 @Override public void onSaveInstanceState(Bundle outState) {
      super.onSaveInstanceState(outState);
      Icepick.saveInstanceState(this, outState);
}
```

A Legendáium Navigátor ezzel a mechanizmussal valósítja meg például az álló és fekvő mód támogatását, az adatokat elmenti, amikor a forgatás történik és betölti, amikor a képernyő elfordult.

2.2.5 Térképszolgáltatások

Térképszolgáltatások szempontjából több API közül választhatunk, amelyek egy része ingyenes. A legelterjedtebb a Google Maps API, de számos alternatíva is létezik, mint például az OpenStreetMap API, a Route-Me, a Mapjam stb. A Legendárium Navigátor projektbe a Google Maps térképszolgáltatás van integrálva, mind az Android kliens alkalmazás, mind a webes alkalmazás esetében.

A Google Maps API a Google által fejlesztett, ingyen használható térképszolgáltatás, amelyet 2005 nyarán adtak ki. Számos lehetőséget biztosít a fejlesztők számára és nagy előnye, hogy a térképnézet teljes felülete személyre szabható. Az alapvető funkciókon kívül (pl. görgetés, közelítés stb.) lehetőség van a térkép feletti rétegre rajzolni (bizonyos információk megjelenítése, markerek). Könnyen integrálható webes alkalmazásokba és mobil alkalmazásokba (Android, iOS) egyaránt.

[1] http://developer.android.com/reference/android/os/Bundle.html

2.2.6 Picasso

A Picasso egy széles körben használt, nyílt forráskódú képletöltő és cachelő könyvtár, amelyet a Square csapata fejleszt. Több általános funkcionalitás megvalósítását teszi egyszerűbbé, mint például:

- Képek letöltése: `Picasso.with(context).load(thumbnailURL).into(ivThumbnail);`
- Képek transzformációja (átméretezés, forgatás):

  ```
  Picasso.with(context).load(thumbnailURL).resize(50,50).rotate(90).into(ivThum
  bnail);
  ```

- Helykitöltők (placeholder) használata. A kép letöltéséig egy placeholder-t (előre megadott kép) jelenít meg, majd mikor letöltődött a kép kicseréli ezt. Emellett a fejlesztőnek lehetősége van egy hibák esetében alkalmazott placeholder-t megadni, ami akkor töltődik be, mikor letöltés során valamilyen hiba lép fel (pl.: megszakad az internetkapcsolat, a kép már nem található meg a szerveren stb.). Például:

  ```
  Picasso.with(context).load(thumbnailURL).placeholder(coverImagePlaceHolder).e
  rror(generalErrorPlaceHolder).into(ivThumbnail);
  ```

- Különböző adatforrások kezelése: képek forrásaként megadhatóak fájlok (file), eszközök (assets), tartalomszolgáltatók (conent providers), erőforrások (resources).
- Párhuzamosan több letöltés megvalósítása.

2.3 Kliens oldali webes technológiák

A webes kliens AngularJS-t használ MVW (Model View Whatever) keretrendszerként. Ez egy nyílt forráskódú webes keretrendszer, amely a Model-View-Controller architektúrára épül. A keretrendszer a HTML oldalakat kiegészíti saját tag attribútumokkal. Ezeket az attribútumokat direktíváknak értelmezi, ezek felelősek a modelleknek egy adott oldal részéhez való kapcsolásával. A modellek standard JavaScript változókban vannak eltárolva. Az egyik leghasználtabb direktíva az *ng-app*, ez a gyökér elemet határozza meg, amelyben további direktívák használhatok úgynevezett binding-ok deklarálására. A keretrendszer egyik legnagyobb előnye hogy elkerüli a Document Object Model (DOM) aktív használatát. Ennek megvalósítására a *$scope* szolgáltatását használja amely eszre veszi ha a modellben történtek változások és módósítja annak megfelelően a felületet egy kontroller segítségével amely visszafele is érvényes.

A Legendárium Navigátor projekt webes felületén egy nagy burkoló angular modulról beszelunk, ez a *legendariumApp* modul amelz egy *app.js* nevezetű fájlban van. Itt vannak a külső angular dependenciák hozzáadva a felülethez. Ugyanakkor külömböző konfigurációk hajtódnak végre, mint például a három fő nézet (footer, navbar, sidebar) mappelése amelyekhez rendelve vannak kontrollerek, illetve az interceptorok hozzáadása. Minden oldalhoz rendelve van egy kontroller amit megadunk a *legendariumApp* modulban, viszont ezt már nem az *app.js* fájlban írjuk meg a projektunk esetében. Pontosan azért, hogy a kódunk átlátható legyen és lehessen látni, hogy az a fájl melyik oldalt is mappeli. A kontrollerek a Rest erőforrásokat *factory*-k segítségével érik el, ahol minden erőforrás fel van mappelve.

```
angular.module('legendariumApp')
    .factory('NewsFeed', function ($resource, DateUtils) {
        return $resource('api/newsFeeds/:id', {}, {
            'query': { method: 'GET', isArray: true},
            'get': {
                method: 'GET',
                transformResponse: function (data) {
                    data = angular.fromJson(data);
                    return data;
                }
            },
            'update': { method:'PUT' },
            $imageUpload: {
                method: 'POST',
                url: 'api/newsFeeds/image/:id'
            }
        });
    });
```

Mindezen funkciónalítások értelmüket veszítenék ha púr HTML-ben kéne megjeleníteni az adatokat. Itt lép be szerepbe a Bootstrap[13] nyílt forráskódú HTML, CSS, JavaScript keretrendszer.

A Bootstrapet (első neve Twitter Blueprint) Mark Otto és Jacob Thornton fejlesztette a Twitternél. Egy elegáns és egyben egyszerű felépítésű webes felületet lehet elkészíteni vele. Az egyik legfőbb funkciónalítása, hogy támogatja a responsive web design-t. Az oldalak megjelenítését dinamikusan valtoztatja annak függvényében hogy milyen eszközről töltöttük be az oldalt.

2.4 További technológiák

Az Legendárium Navigátor adatbázisa menedszelésére A Liquibase [14] keretrendszert használjuk, amely egy platform független, adatbázis független és nyílt forráskódú keretrendszer. Lehetővé teszi az adatbázis sémák alkalmazását, ez segít az adatbázis változásainak követésében. Az adatbázis sémákat a Legendárium Navigátor projekt esetében XML formátumban tárolja (ezek

lehetnének akár YAML, JSON vagy SQL formátumban is). A fájlok nevei egy id-val és maga az entitás nevével van felépítve, ami alapján készült az adott séma. A táblák létrehozásánál automatikusan készít egy DataBaseChangeLog táblát. Minden egyes újrafuttatáskor ellenőrzi ezek hitelességét, amennyiben változás történt a sémában akkor kiegészíti a neki megfelelő táblát.

A szoftver minőségét illetve funkcionalitások betartását a tesztelés során biztosítjuk. Erre a Junit-ot [15] hasznláljuk, amely a Java programozási nyelvnek szánt unit teszt keretrendszer. Lehetővé teszi ezen nyelven megírt szoftverek automatizált tesztelését. Egy konkrét példa az, amikor adatbázisban egy törlés esetén milyen választ várunk el a rendszertől:

```
@Test
@Transactional
public void deleteLegend() throws Exception {
    legendRepository.saveAndFlush(legend);
    int databaseSizeBeforeDelete = legendRepository.findAll().size();
    restLegendMockMvc.perform(delete("/api/legends/{id}", legend.getId())
            .accept(TestUtil.APPLICATION_JSON_UTF8))
            .andExpect(status().isOk());
    List<Legend> legends = legendRepository.findAll();
    assertThat(legends).hasSize(databaseSizeBeforeDelete - 1);
}
```

Mivel előfordúl, hogy az objektumoknak vannak külső függőségei ezért azok viselkedését kell szimulálni és itt lép be a Mockito [16]. A Mockito egy Java alapú teszt keretrendszer, amely lehetővé teszi az úgynevezett mock objektumok használatát. Ezen mock objektumok az eredeti objektumok butított változatai, amelyek helyettesítik a vizsgált objektum függőségeit és szimuláljak viselkedését.

Naplózási keretrendszernek az Simple Logging Façade for Java [17] (SLF4J) keretrendszert használtuk. Ezen keretrendszerek biztosítják az üzenetek rendszerezését az alkalmazáson belül. Az üzenetek kimeneti helyét és az üzenetek szintjét be lehet állítani.

Az SLF4J absztrakciós szintet képez a Log4J [18] naplózási keretrendszer fölött. Lehetőséget biztosít a felhasználók számára, hogy a kívánt naplózási keretrendszert telepítési időben kössék be. Ugyanakkor a Log4j, naplózási szinteket (Info, Warning, Debug, stb.) biztosít az üzenetek elhatárolásához.

3 A rendszer megvalósításának összefoglalója

A következőkben a dolgozat röviden összefoglalja a LegendáriumNavigátor projekt megvalósításának fontosabb részleteit.

3.1 A szerver megvalósítása

A szerver modul alapszerkezete JHipster segítségével volt generálva. A JHipster egy Yeoman[6] generátor, amely segítségével egy Spring Boot + AngularJS projekt hozható létre. A projekt alapszerkezete magában foglalt egy előre konfigurált szervert és egy kezdetleges webes felületet.

A repository réteg megvalósítása a Spring Data JPA keretrendszer segítségével történt. Ez leegyszerűsíti a programozó dolgát, mivel nem kell konkrét implementációt biztosítani az adathozzáférési réteg komponenseire, csak interface-eket kell létrehozni. Ezek leszármazottjai kell legyenek a JpaRepository-nak, amely már tartalmazza a CRUD metódusokat (*findOne*, *findAll*, *save*, *delete*, *deleteAll*). A metódusok konvenciónak megfelelő nevei alapján a repository interfészek funkcionalitásai könnyen bővíthetőek. A nevek alapján generálja le a keretrendszer a lekérdezéseket és végzi el a megfelelő adatbázis műveleteket.

```
public interface LegendRepository extends JpaRepository<Legend,Long> {
      Legend findByName(@Param("name") String name);
      List<Legend> findByRegionId(@Param("region_id") String regionId);
      Legend findByBookId(@Param("book_id") String bookId);
}
```

Ha komplexebb lekérdezésekre van szükség, akkor erre is van lehetőség: a metódusokon alkalmazott *@Query* annotáció segítségével JPQL lekérdezések is megadhatóak.

Biztonsági megoldásokra Spring Security keretrendszert használ a projekt. A keretrendszer segítségével van megvalósítva az URL-ek levédése különböző jogköröknek megfelelően, a @Secured annotáció segítségével. Az autentikáció Oauth2-es szabvány szerint történik. A felhasználó bejelentkezéskor két tokent kap, egy refresh és egy access token-t. A bejelentkezés pillanatától a felhasználó által küldött minden egyes kérés ellenőrzése az access token-el történik, majd ha ennek az életciklusa lejár a refresh token segítségével új refresh és access token-t lehet igényelni a szervertől.

Az e-mail-ek kiküldéséről a Spring keretrendszer mail szolgáltatása gondoskodik. Ennek bekonfigurálása egy kontextus állományban történik.

A REST erőforrások Spring Web MVC - REST annotációk segítségével vannak konfigurálva. A *@RequestMapping* annotáció segítségével van megadva a metódus elérési útvonala, és az, hogy milyen HTTP kérés hatására hívódjon meg. A kliens válaszában többek között egy állapotkód is szerepel, amely alapján el lehet dönteni, hogy történt-e hiba a rendszerben. A *@Transactional* annotáció arra szolgál, hogy miután egy erőforrásra hivatkoztak és annak megfelelő metódusának futtatása során valami hiba keletkezett, akkor az addig elvégzett műveleteket érévnytelenítse (a kérésre érkezett válasznak kötelező módon teljesnek kell lennie, másképp hibás működéshez juthatnánk). Amennyiben hiba lép fel, egy kivétel keletkezik, aminek a kezelése után a megfelelő hibaüzenet jut el a felhasználóhoz.

3.2 Az Android kliens alkalmazás megvalósítása

Az Android alkalmazás alapját Activity-k és Fragment-ek képezik, amelyek képernyő nézeteket írnak le. Ezekhez XML fájlok tartoznak, amelyekben konfigurálható a képernyő kinézete, itt lehet hozzáadni komponenseket, illetve meghatározni azok tulajdonságait. Az alkalmazás támogatja az álló és fekvő módot is, ezt az Icepick könyvtár segítségével valósítja meg. Többféle képernyőfelbontás is támogatott, ez különböző méretek megadása, illetve 9-patch típusú képek használata által lehetséges. A 9-patch típusú képek használatának lényege, hogy be lehet állítani, hogy egy képnek melyik részeit skálázhatja a rendszer.

Az alkalmazásban több helyen is szükség van képek betöltésére, ez a Picasso segítségével valósul meg. A technológia előnye, hogy egyszerűen lehet megadni, hogy honnan hová töltse be az alkalmazás a képeket, ezen kívül placeholder-eket is lehet használni.

Egy fontos része az alkalmazásnak a perzisztens adatok kezelése. A felhasználó letöltheti a legendákhoz tartozó tartalmakat a telefonjára. Azért fontos ez a funkció, mert így nem szükséges folyamatos internet hozzáférése legyen a felhasználóknak.

Az alkalmazáson belül navigációt is lehet kérni az egyes helyszínekhez. Először kirajzolódik az útvonal a jelenlegi helyszín és a legenda helyszíne között, majd a felhasználó útbaigazítást kérhet, amelyet az alkalmazás a GoogleMaps segítségével valósít meg.

4 Alkalmazott módszerek és felhasznált eszközök

A projekt fejlesztése során a csapat Scrum módszert [1] használt, ami egy agilis szoftverfejlesztési szemléletmód. A Scrumnak megfelelően a fejlesztés sprintekben valósult meg, ezek mindig egy tervezési fázissal kezdődtek (sprint planning), ahol eldőlt, hogy melyik felhasználói történetek (User Story-k) kerülnek be az elvégzendő feladatok listájába. A felhasználói történetek prioritási sorrendbe voltak rendezve, az alkalmazás folyamatosan egészült ki új funkcionalitásokkal és minden futam (sprint) végén volt egy bemutató (demo).

Verziókövető rendszerként a Mercurial [2] volt használva, amely megkönnyítette a csapat dolgát a forráskód változásaink nyomonkövetésekor, illetve támogatta a csapatmunkát. Kliensalkalmazásként a csapat a SourceTree-t használta, a központi tároló menedzsmentjét a RhodeCode rendszer biztosította.

A projekt elkezdésekor segített a JHipster [3], amely felépített a projekt számára egy kiindulási pontként használható alapszerkezetet. Build és függőségmenedzsment rendszerként a Legendárium Navigátor Gradle-t [7] használ, így a projekt konfigurálása Groovy-alapú. A webes modul esetében a build folyamatot támogatta továbbá a Yeomann és a Grunt [8].

A folytonos integráció (Continuous Integration - CI) egy olyan fejlesztési folyamat, ami hatékonyabbá teszi a csapatmunkában történő fejlesztést. A fejlesztők folyamatosan törekednek arra, hogy a kód fordítható, az alkalmazás futtatható legyen. A módszer alkalmazásában a CI rendszerek segítenek, a Legendárium Navigátor projekt esetében a Jenkins-t [9] töltötte be ezt a szerepet. A verziókövető rendszerrel összekapcsolva a Jenkins képes automatikusan felépíteni a Gradle projekteket és a build folyamat eredményeit egy webes felületen is elérhetővé teszi. Összekapcsolható az automatikus kódelemző eszközökkel, így minden fordítás után lefutnak az elemzések. Ezen kívül automatikusan kitelepíti a friss verziókat a tesztszerverekre.

A megírt kód minőségének biztosítását nagyban elősegítik a különböző kódelemző szoftverek, mint például a SonarQube [10]. A SonarQube egy Javaban fejlesztett rendszer, amely jelenleg több mint 20 programozási nyelvet támogat. Az ellenőrzés több szempont szerint zajlik: komplexitás és kódolási konvenciók, duplikátumok, dokumentálás, tesztlefedettség, lehetséges programozási hibák kiszűrése adott szabályrendszer alapján. Hibák esetében a rendszer javaslatot is tesz a fejlesztők számára a lehetséges megoldásokkal kapcsolatban.

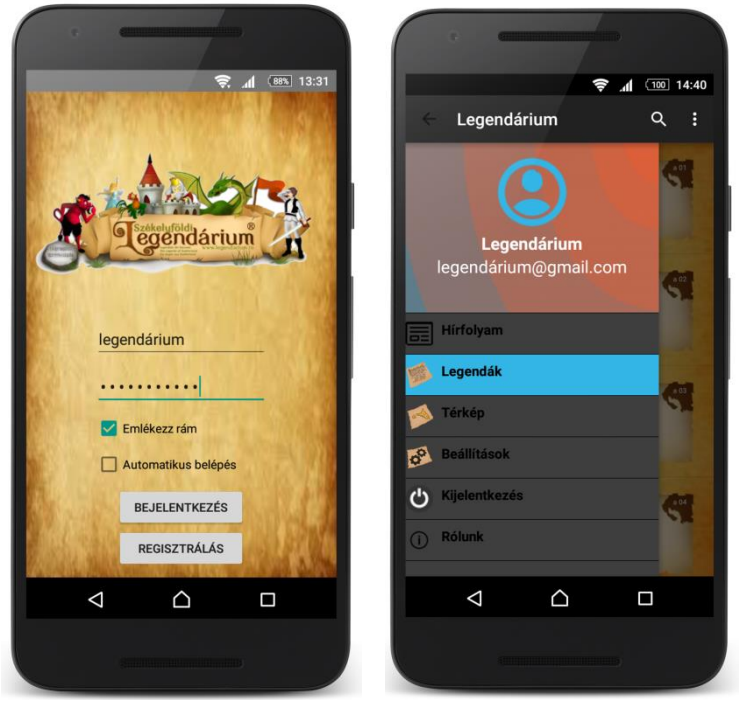
Fejlesztői környezetként Android Studio-t és IntelliJ IDEA-t használt a csapat, kiegészítve ezeket további fejlesztést támogató eszközökkel (pl. az adatbázis menedzsmentjét, tervezést, felhasználói felület vázlatainak elkészítését támogató eszközökkel).

5 A Legendárium Navigátor működése

A következőkben a Legendárium Navigátor kliens alkalmazásának és adminisztrációs felületének működése lesz röviden bemutatva.

5.1 Az Android kliens alkalmazás használata

A Legendárium Navigátor alkalmazás elindításakor egy bejelentkezési felület fogadja a felhasználókat, ahol bejelentkezhetnek a rendszerbe. Ezen kívül a felhasználónak van lehetősége a szokásos *Jegyezz meg* és *Léptess be* funkciót kiválasztani, hogy ha nem szeretné mindig beírni a felhasználónevét és jelszavát, illetve ha szeretné, hogy a rendszer automatikusan beléptesse az alkalmazás legközelebbi indításánál. Ha valaki még nem regisztrált felhasználó, akkor rákattintva a *Regisztrálás* gombra, egy új nézet jelenik meg, ahol ezt megteheti. Itt meg kell adnia egy felhasználónevet, jelszót és egy e-mail címet. Ezt követően a rendszer egy megerősítő e-mailt küld a felhasználó által megadott címre, amelyben egy link található. A felhasználó a link-re kattintva igazolja az e-mail címének a helyességét és ezután be tud jelentkezni a rendszerbe.



3. *ábra*: Bejelentkezési felület és kinyíló menü

Sikeres belépés után egy főmenü jelenik meg, amelyen három gomb látható: *Hírfolyam*, *Legendák*, *Térkép*. Ezeket a menüpontokat a felhasználó elérheti a baloldalról kinyíló menüből is, ahol

ezek mellet még további menüpontok láthatóak: *Beállítások*, *Rólunk*, *Kijelentkezés*. Továbbá, elindul a háttérben egy szolgáltatás, amely jelez a felhasználónak, ha egy legenda közelébe ér. Ezt a szolgáltatás egy alapértelmezetten beállított időköz és minimum távolság alapján végzi.

A *Hírfolyam* menüpontra kattintva megjelennek a legendáriumos munkatársak által feltöltött hírek, programleírások. Ezek egy listában jelennek meg, a hozzájuk feltöltött képpel, illetve a leírásuk első pár szavával. Ha rákattint a felhasználó egy hírre, akkor egy külön nézet jelenik meg a hírhez csatolt nagyobb méretű képpel, valamint a teljes leírással.

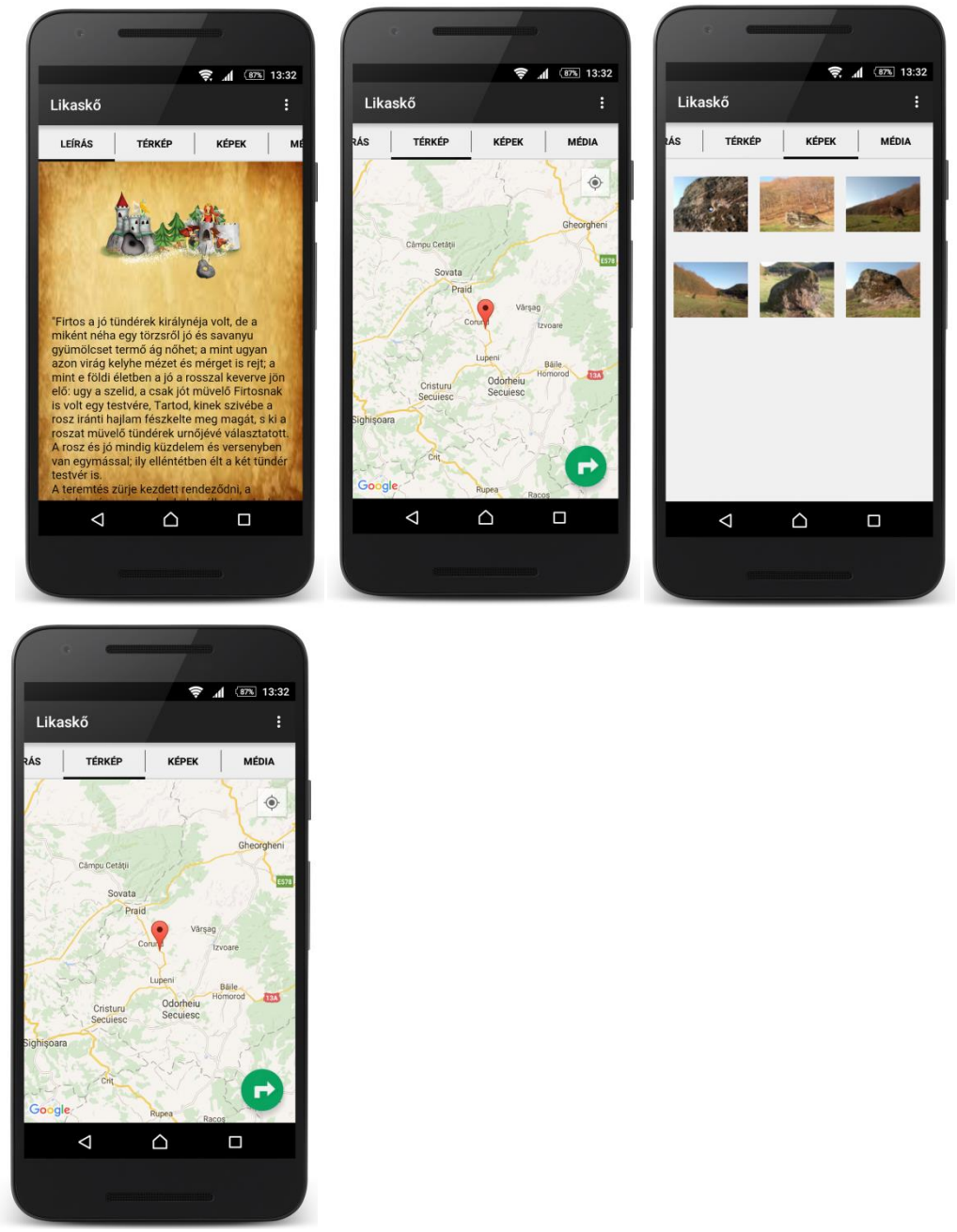
4. *ábra*: Hírfolyam

A *Legendák* menüpont kiválasztása után megjelennek a legendák. Minden legenda egy külön „kártyán” jelenik meg, amelyen fel van tüntetve a legenda neve, régiója, a Legendárium könyvön belüli azonosítója, illetve egy kisebb kép.

5. *ábra*: Legenda lista

A felhasználónak lehetősége van keresni a legendák között: a jobb felső sarokban található *kereső gomb* megnyomásával megjelenik egy szövegmező, ahová beírhatja a legenda nevét vagy régióját, és a lista automatikusan frissül a keresés eredményeinek megfelelően. Ha a felhasználó rákattint egy legenda kártyára, akkor egy új nézet jelenik meg, négy panellel (tab): *Leírás*, *Térkép*, *Képek*, *Média*. A *Leírás* rész alatt látható a legendához csatolt kép nagyobb változata, valamint a legenda teljes leírása. A térképes nézetnél Google Maps segítségével megjelenik a legenda helyszínét jelölő POI[2]. A felhasználó kérhet útvonaltervezést és az alkalmazás el is tudja navigálni a legenda helyszínéhez. A *Képek*-nél a legendához tartózó fényképek jelennek meg egy galériában. Ezeket a fényképeket is meg lehet tekinteni nagyobb méretben, ha kiválasztunk a galériából egy képet. Az utolsó nézeten belül a legendához csatolt hanganyagokat lehet meghallgatni.

[2] https://en.wikipedia.org/wiki/Point_of_interest



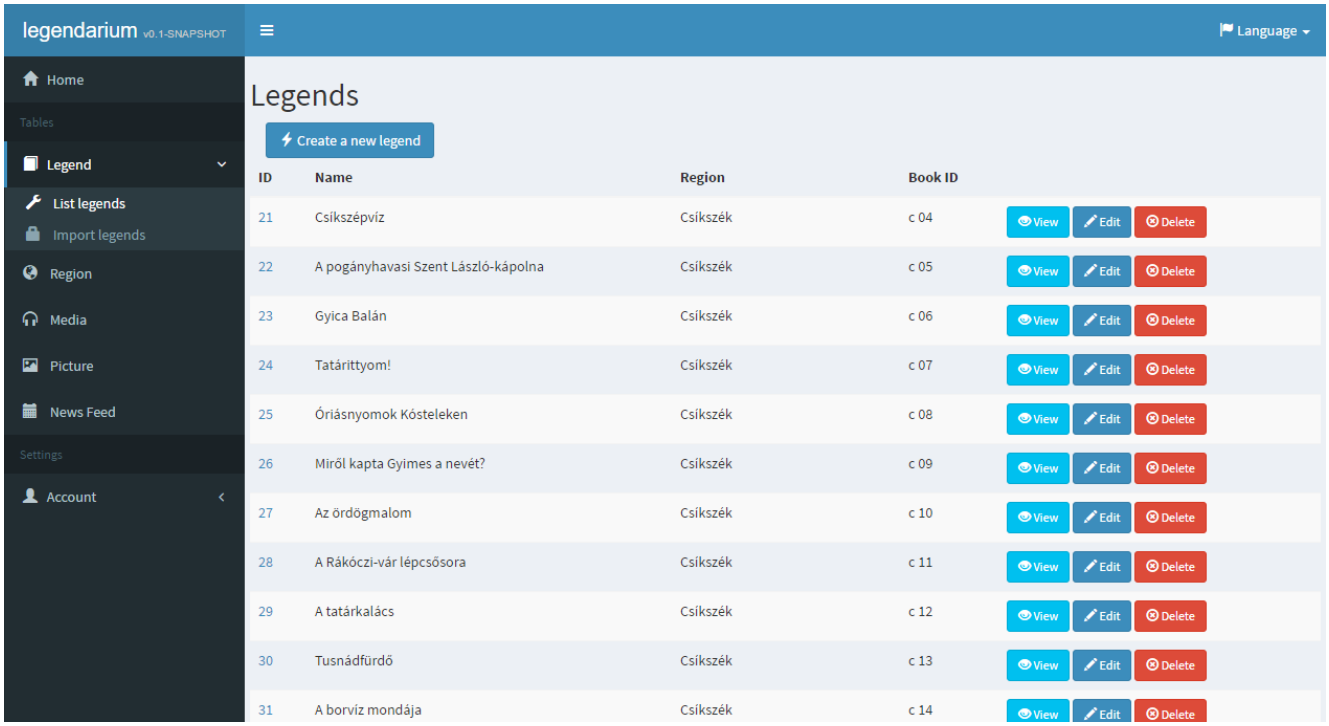
6. *ábra*: Leírás-, Térkép-, Képek-, Média nézet

A *Térkép* menüpont kiválasztása után a legendák helyszíneit POI-k jelölik. Itt is lehet a legendák között keresni, hasonlóan, mint a legenda listánál. Egy POI-ra rákattintva egy felnyíló menü jelenik meg, ahol látható a legenda neve és helyszíne. A menü segítségével megnyitható egy nézet, ahol megjelenik a legenda képe, leírása valamint három kis gomb. Utóbbiak közül az első jelöli, hogy a legenda le van-e töltve. Ha nincs, akkor a gombra kattintva letölthető. A második gombra kattintva az alkalmazás megjeleníti a legendának a részletes nézetét. A harmadik gomb frissíti a legendát: ha az előzőleg már el volt mentve, a tartalom frissül a szerver oldalon történt esetleges változtatások alapján.

A *Beállítások* menüpont alatt a felhasználó megváltoztathatja a felhasználónevét és az e-mail címét, valamint beállíthatja, hogy milyen minimum távolságra kell lennie egy legendától ahhoz, hogy jelezzen az alkalmazás, illetve ki is kapcsolhatja az értesítéseket. Ugyancsak a beállításoknál törölhetőek az eddig letöltött legendákat.

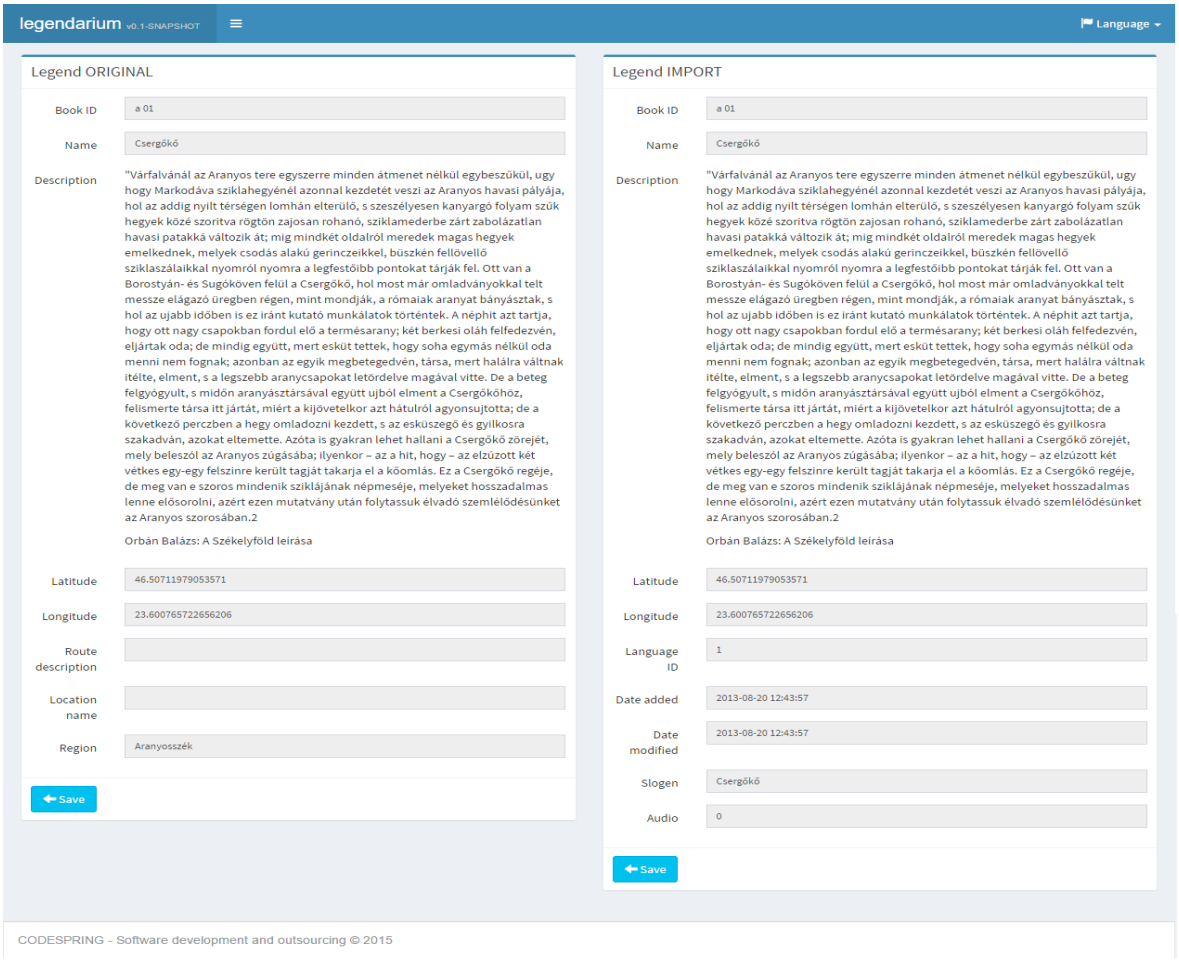
5.2 A Webes adminisztrációs felület használata

Bejelentkezés után a Legendárium munkatársainak lehetőségük van a legendák módosítására, törlésére, illetve új legendák létrehozására. A legendákhoz fel tudnak tölteni képeket és média állományokat. Mindezek megtekinthetőek a felületen.



7. *ábra:* A webes felületen kilistázott legendák

Lehetőség van legendák importálására is. A külső adatbázisból származó adatokat egy JSON állományból lehet feltölteni a felületen keresztül a szerverre, amely ezeket az adatokat feldolgozza és megjeleníti a felhasználó számára az oldalon. Ezek után lehetőség van a legendák mentésére. Konfliktus esetében (ha a legenda már szerepel a szerver adatbázisában, akkor a Legendárium könyvön belüli azonosítók megegyeznek) a felületen jelzi a rendszer, hogy nem sikerült elmentenie az adott legendát. Ezután két oszlopban megjeleníti a konfliktusba került legenda tartalmakat (a szerveren tárolt, illetve az importált változatot), és a felhasználóra bízza a helyes változat kiválasztását.



8. *ábra*: Konfliktusok kezelése

A rendszer adminisztrátora plusz jogosultságokkal rendelkezik a Legendárium munkatársakhoz képest. Regisztrálni tud új felhasználókat és különböző szerverrel való információkat tud megtekinteni (statisztikák, konfiguráció, naplók, API dokumentáció), illetve API hívásokat tud kezdeményezni a SwaggerUI segítségével.

Következtetések és továbbfejlesztési lehetőségek

A Legendárium Navigátor projekt fejlesztése során sikerült egy követelményeknek megfelelő prototípust létrehozni, amellyel a Székelyföldi Legendárium csapata is meg volt elégedve. Az alkalmazás fejlesztéséről már több beszámoló is megjelent különböző újságokban és a tévében is (az M1 hírcsatorna műsorában).

A fejlesztés során több lehetőség felmerült azzal kapcsolatban, hogy milyen funkciókkal lehetne bővíteni az alkalmazást. Ezek közül a fontosabbak:

- az alkalmazás szorosabb összekapcsolása közösségi hálózatokkal (pl. események megosztása közösségi hálók segítségével);
- a kirándulási útvonalak tervezése (a funkcionalitás fejlesztés alatt áll);
- gamification: a legendák helyszíneinek meglátogatása után a felhasználók jutalmakat kapnak, ezek esetleg beválthatóak a Székelyföldi Legendárium termékeire stb.;
- legendákkal kapcsolatos video anyagok megjelenítése;
- a legendák helyszíneihez közeli szálláshelyek és vendéglátó ipari egységek megjelenítése, foglalási lehetőség az alkalmazáson keresztül.

A közeljövőben tervezi a csapat az alkalmazás első verziójának hivatalos kiadását, így remélhetőleg hamarosan használhatóvá válik a kirándulók, Erdélybe látogató turisták számára.

Hivatkozások

[1] Scrum hivatalos weboldal, http://www.scrumguides.org/scrum-guide.html, utolsó megtekintés dátuma: 2016.03.10

[2] Mercurial hivatalos weboldal, http://mercurial.selenic.com/, utolsó megtekintés dátuma: 2016.04.03

[3] JHipster hivatalos weboldal, http://jhipster.github.io/, utolsó megtekintés dátuma: 2016.04.03

[4] Spring keretrendszer hivatalos weboldal, http://spring.io/, utolsó megtekintés dátuma: 2016.04.03

[5] AngularJS hivatalos weboldal, https://angularjs.org/, utolsó megtekintés dátuma: 2064.04.15

[6] Yeoman hivatalos weboldal, http://yeoman.io/, utolsó megtekintés dátuma: 2016.03.05

[7] Gradle hivatalos weboldal, http://gradle.org/, utolsó megtekintés dátuma: 2016.04.08

[8] Grunt hivatalos weboldal, http://gruntjs.com/, utolsó megtekintés dátuma: 2016.04.08

[9] Jenkins hivatalos weboldal, https://jenkins.io/, utolsó megtekintés dátuma: 2016.03.10

[10] SonarQube hivatalos weboldal, http://www.sonarqube.org/, utolsó megtekintés dátuma: 2016.03.10

[11] MySQL hivatalos weboldal, https://www.mysql.com/, utolsó megtekintés dátuma: 2016.03.24

[12] Chris Schaefer, Clarence Ho, Rob Harrop, Pro Spring, 4th edition, Apress 2014

[13] Bootstrap hivatalos weboldal, https://bootstrapdocs.com/v3.3.6/docs/, utolsó megtekintés dátuma: 2016.04.27

[14] Liquibase hivatalos weboldal, http://www.liquibase.org/documentation/, utolsó megtekintés dátuma: 2016.04.26

[15] Junit hivatalos weboldal, http://junit.org/junit4/javadoc/latest/, utolsó megtekintés dátuma: 2016.04.26

[16] Mockito hivatalos weboldal, http://mockito.org/, utolsó megtekintés dátuma: 2016.04.26

[17] SLF4J hivatalos Weboldal, http://www.slf4j.org/, utolsó megtekintés dátuma: 2016.04.26

[18] Log4j hivatalos Weboldal, http://logging.apache.org/log4j/2.x/, utolsó megtekintés dátuma: 2014.04.26