XVII. reál- és humántudományi Erdélyi Tudományos Diákköri Konferencia

(ETDK)

Kolozsvár, 2014. május 15- 18.

On-line kölcsönzési szolgáltatás közösségek számára

Szerzők:

Szőcs Emőke–Katalin

Babeș-Bolyai Tudományegyetem, Kolozsvár, Matematika és Informatika Kar, Információ mérnöki szak, 4. évfolyam

Merli András Bertalan

Babeș-Bolyai Tudományegyetem, Kolozsvár, Matematika és Informatika Kar, Információ mérnöki szak, 4. évfolyam

Témavezetők:

Dr. Simon Károly

egyetemi adjunktus, Babeș-Bolyai Tudományegyetem, Matematika és Informatika Kar, Magyar Matematika és Informatika Intézet, Codespring Kft.

Szilágyi Zoltán

szoftverfejlesztő, Codespring Kft.

Kivonat

A dolgozat a SoLeBo (Social Lending and Borrowing Service) szoftverrendszer alapötletét, szakmai hátterét és megvalósítását tárgyalja. Leginkább az különbözteti meg a rendszert a jelenleg elérhető más online kölcsönzési szolgáltatásoktól, hogy nem kíván globális megoldásként szolgálni, hanem kisebb, zártabb közösségeket, baráti társaságokat, szakmai csoportokat céloz meg. A felhasználók csoportokat hozhatnak létre, több csoportnak lehetnek tagjai, és a csoporton belül hirdethetik kölcsönözhető tárgyaikat, melyek különböző kategóriákba tartozhatnak (pl. könyvek, sportfelszerelések, társasjátékok stb.). Az adatok (tárgyak, felhasználók, foglalások stb.) menedzsmentje mellett a rendszer folyamatosan figyeli a kölcsönzések állapotát és automatikusan értesítéseket küld az érdekelt felhasználóknak. A felsoroltakon kívül számos további funkcionalitást biztosít a szoftver, többek között támogatja kölcsönzések helybeni létrehozását és megerősítését egy mobil alkalmazás segítségével.

Tartalomjegyzék

Bevezető

A SoLeBo projekt célja egy új típusú, online kölcsönzési szolgáltatás létrehozása. Az ötlet ismertetésének céljából nézzünk egy konkrét példát: egy személy olyan szakkönyvek birtokában van, amelyeknek a munkatársai is hasznát vennék. Kölcsönadhatja a könyveket az érdeklődőknek, viszont, hogy visszakérhesse ezeket, rendelkeznie kell egy listával a kölcsönzőkről. Ezt a listát közzé teheti weboldalak, email szolgáltatók, esetleg más információmegosztó rendszerek segítségével. A lista állapota azonban még eleget kell, hogy tegyen néhány követelménynek: a kölcsönzések és kölcsönzők adatainak (email cím, telefonszám stb.) naprakészeknek kell lenniük. Emellett előnyös lenne, ha a kölcsönzést lebonyolító mindkét fél böngészhetné ezeket az információkat és módjában állna frissíteni azokat. Célszerű lenne, ha valamilyen formában értesítést kapnának a kölcsönzés lejártának közeledtével stb.

Több olyan szoftvert találunk a piacon, amelyeket kimondottan az online kölcsönzésekre terveztek, de ezek esetében észrevehető néhány hiányosság. Tipikusan átfogó elgondolásokkal találkozhatunk, abban az értelemben, hogy a kölcsönözhető tárgyak a rendszer minden felhasználója számára láthatóak, nem alakíthatóak ki privát felhasználói csoportok (pl. egy szakmai közösség részére). A SoLeBo projekt célja ennek a helyzetnek a megoldása.

A SoLeBo rendszer keretén belül lehetőség van zártabb felhasználói csoportok létrehozására, például munkaközösségek, baráti társaságok részére. Így a kölcsönözhető tárgyak (pl. könyvek) nem válnak elérhetővé mindenki számára, a tulajdonos tudja beállítani ezek láthatóságát a csoportjaiban. Az alkalmazás segítségével a felhasználók követhetik a kölcsönzések állapotát, lefoglalhatnak bizonyos tárgyakat, értesülhetnek egy lefoglalt tárgy felszabadulásáról stb. A rendszer adminisztrátora a felsoroltakon kívül speciális jogosultságokkal is rendelkezik, ilyen például a tárgykategóriák kezelése.

A webes felület mellett a felhasználók számára SoLeBo egy mobil alkalmazással biztosítja, hogy már a találkozáskor, helyben is létre lehessen hozni, meg lehessen erősíteni, illetve le lehessen zárni egy kölcsönzést.

A dolgozat első része a felhasznált eszközök és módszerek rövid leírását tartalmazza, a második része a technológiákat, fejlesztési módszereket, mintákat szemlélteti. A harmadik részben a projekt követelményeiről, környezeti elemzéséről, a rendszer architektúrájáról, tervéről, illetve a megvalósítás néhány kiemelt részletéről talál információkat az olvasó, amelyet egy rövid esettanulmány követ, betekintést engedve a rendszer működésébe. Végül az

elért eredmények összefoglalása után néhány továbbfejlesztési lehetőség megemlítésével ér véget a dolgozat.

Az alkalmazás ötlete a kolozsvári Codespring Kft.-től származik, amely vállalta, hogy támogatja a projekt megvalósítását, többek között a szakmai irányítást és a megfelelő infrastruktúrát biztosítva.

1. Felhasznált eszközök és módszerek

A SoLeBo projekt fejlesztése során a hatékonyság növelése érdekében a fejlesztők több modern, széles körben alkalmazott fejlesztői eszközt használtak.

A verziókövetés a Mercurial [1] osztott verziókövető rendszer segítségével történt. Ennek egyszerűbb és kényelmesebb használata érdekében a fejlesztők a TortoiseHg [2] grafikus felhasználói felülettel rendelkező asztali alkalmazást használták. A központi tároló menedzsmentjét a RhodeCode biztosította.

Build folyamatok automatizálására és függőségek kezelésére a fejlesztés során az Apache Mavent [3] használták a fejlesztők. A Maven projektek a *Project Object Model* segítségével vannak definiálva, pom.xml állományok által. Minden Maven projekt rendelkezik három tulajdonsággal, amelyek egyértelműen azonosítják a projektet: groupId, artifactId, version. A groupId azt határozza meg, hogy a projekt melyik projektcsoportba tartozik, az artifactId a csoporton belüli azonosításra szolgál, a version pedig a projekt aktuális verziószámát adja meg. Egy Maven projekt életciklusa meghatározza, hogy egy adott fázisban mi történjen a projekttel. Egy adott fázis megadásának segítségével az összes előtte lévő fázison belüli műveletek lefutnak.

Egy szoftver esetében a kód helyességén kívül a kód minőségére is nagy hangsúlyt kell fektetni. Egy széles körben használt eszköz kódanalízisre a SonarSource által kifejlesztett SonarQube [4]. A SoLeBo projekt forráskódjának ellenőrzésére a fejlesztők ezt az eszközt alkalmazták. A SonarQube ellenőrzi a kód megfelelőségét egy előre rögzített szabályrendszer alapján, javaslatokat tesz javításokra, ellenőrzi a komplexitást, duplikátumokat és tesztlefedettséget, valamint kóddal kapcsolatos statisztikai adatokat biztosít.

Egy megfelelő projektmenedzsment és hibakövető szoftver használata nagyon fontos egy fejlesztés során. Ennek segítségével a fejlesztők különböző tényezők szerint csoportosíthatják a feladatokat, illetve nyomon követhetik ezeket. Ennek köszönhetően követhető a munkafolyamat, és jobban átlátható, hogy milyen fázisban van egy projekt. Projektmenedzsment és hibakövető szoftverként a SoLeBo fejlesztése során a fejlesztők a Redmine [5] rendszert használták. A Redmine egy ingyenes, nyílt forráskódú, webalapú projektmenedzsment és hibakövető eszköz, amelyet a Ruby on Rails keretrendszert használva írtak meg. Támogatja több projekt egyidejű nyomon követését, lehetőség adódik a hozzáférés szerep alapú szabályozására, verziókövető rendszerek egyszerű integrációjára, dokumentum- és fájlkezelésre, különböző diagramok és naptárak vizuális ábrázolására.

Több programozó által fejlesztett rendszerek esetében komoly feladat a különböző kódok integrálása, fejlesztési ágak összefésülése. A folytonos integráció (*Continous*

Integration, CI) módszere ennek hatékony elvégzését támogatja, azt a célt szolgálja, hogy a rendszer mindig helyes, felépíthető és futtatható állapotban legyen. Az alábbi alapelvekre épül: támogatja a módosítások gyakori megosztását (min. napi commit/push műveletek a központi tárolóba), minden változtatás után (vagy adott időközönként) a rendszer automatikus felépítését, beleértve az automatizált tesztek lefuttatását is. A rendszer aktuális állapotával kapcsolatos információkhoz a fejlesztő csapat tagjai bármikor hozzáférhetnek. Ezeknek a céloknak a megvalósításában segítenek a különböző CI eszközök, amelyek build szervereket biztosítnak. A Solebo projekt esetében a Jenkins [6] CI szerver volt használva, amely egy nyílt forráskódú, Java programozási nyelvben megírt eszköz. A rendszer összeköthető a verziókövető rendszerrel, a változtatások után automatikusan buildelni tudja a Maven projekteket. Összeköthető minőségbiztosítási rendszerekkel is, például automatikusan futtatni tudja a SonarQube elemzéseit. Ezen kívül lehetőséget ad a projekt automatikus telepítésére tesztszerverekre és folyamatosan aktuális képet biztosít a csapat számára a projekt állapotáról.

A web alkalmazás a GlassFish [9] alkalmazásszervert használja. A GlassFish egy nyílt forráskódú alkalmazásszerver, amely jelenleg az Oracle Corporation tulajdonában van. A Java Enterprise Edition referencia implementációja, így támogatja az EJB, JPA, JMS, RMI stb. technológiákat.

A SoLeBo projekt MySQL relációs adatbázis-menedzsment rendszert használ. Ennek konfigurálása a MySQL Workbench [10] eszköz segítségével történt. A relációs adatbázis változásainak követésére és kezelésére egy adatbázis-független könyvtár van használva, a Liquibase [11].

A SoLeBo fejlesztői a NetBeans [7] integrált fejlesztői környezetet használták az alkalmazás elkészítéséhez. A felhasználói felület tervezése a Balsamiq [8] mockupkészítő eszköz segítségével történt.

2. Felhasznált technológiák

A SoLeBo szerver oldala a Java Enterprise Edition platformra épül, webes felülete a Vaadin keretrendszer segítségével van felépítve és a projekt keretein belül egy Android kliensalkalmazás fejlesztése is megtörtént.

1. A Java Enterprise Edition platform

A Java Enterprise Edition (Java EE) [12] egy széles körben alkalmazott szerver oldali Java programozási platform, amely komponens alapú, osztott vállalati rendszerek fejlesztését támogatja.

A Java EE alkalmazások alkalmazásszerverekre telepíthetőek és ezeken futtathatók. Az alkalmazások hordozhatók a Java EE specifikációnak megfelelő alkalmazásszerverek között. A SoLeBo projekt fejlesztése során a fejlesztők az Oracle által fejlesztett Glassfish alkalmazás szerver nyílt forráskódú változatának 4.0 verzióját használták (létezik egy kereskedelmi változat is). A Glassfish-en kívül számos más alkalmazásszerver megfelel a Java EE specifikációnak: JBoss (nyílt forráskódú, a Red Hat által fejlesztett, az új neve WildFly), WebLogic (Oracle), WebSphere (IBM) stb.

Az alkalmazásszerver feladata a fejlesztési folyamat egyszerűsítése különböző szolgáltatások által (pl. tranzakció-kezelés, központosított konfiguráció, biztonsági megoldások stb.), hogy a fejlesztők az alkalmazás logikájára koncentrálhassanak.

A Java EE komponensek, olyan szerver oldali szoftverkomponensek, amelyek önálló funkcionalitással biztosítanak és kommunikálnak egymással. Ezek a komponensek lehetnek web komponensek (Servlet, JSP), amelyek a Java EE web-konténerében futnak, illetve lehetnek Enterprise JavaBean-ek (EJB) amelyek az üzleti logikáért felelősek és az alkalmazásszerver EJB konténerében menedzseltek. A Java EE komponensek kezeléséért a használt alkalmazásszerver felelős a különböző konténer beállítások alapján.

A Java EE számos programozási interfészt és keretrendszert biztosít, amelyek segítségével a platform könnyen kezelhetővé válik és ezáltal egyszerűen használható komplex problémák megoldására.

1.1. Java Persistence API (JPA)

A JPA [13] egy specifikáció, amely standard eljárást definiál objektumok adatbázisba történő leképzésére. Központi egységei POJO-k (Plain Old Java Object), amelyeket JPA entitásoknak nevezünk, és JPA metaadatok (annotációs mechanizmus vagy XML leíró állomány) segítségével lesznek leképezve az adatbázisba.

A JPA egy Entity Manager API-t biztosít, amelyen keresztül megvalósíthatóak a perzisztenciával kapcsolatos műveletek. Az adatok betöltése, módosítása, törlése így megtörténhet anélkül, hogy a fejlesztőknek ehhez kapcsolódó forráskódot kellene írniuk.

A JPA egy lekérdező nyelvet is meghatároz, amelynek funkcionalitásai azonosak az SQL lekérdezőnyelvek által biztosított funkcionalitásokkal, de – ellentétben az SQL nyelvekkel – Java objektumokkal dolgozik az objektumorientált szemléletmódnak megfelelően. Továbbá egy Criteria Query API-t biztosít, amelynek segítségével dinamikusan, futási időben is felépíthetőek lekérdezések. Természetesen natív query-k használatára is lehetőséget ad.

A JPA szabványnak léteznek nyílt forráskódú implementációi, a SoLeBo a referencia implementációt, az EclipseLink-et használja.

1.2. Enterprise JavaBeans (EJB)

Az Enterprise JavaBean-ek [14][15] osztott vállalati Java EE alkalmazásokban használt szerveroldali komponensek. Az EJB-k az EJB konténerben menedzseltek, melyet a használt, EJB specifikációnak megfelelő alkalmazásszerver biztosít.

Kétféle EJB-típust különböztetünk meg: Session Bean-eket (SB) és Message Driven Bean-eket (MDB). A korábbi EJB specifikációkban egy harmadik kategóriát képeztek az Entity Bean-ek, amelyeket váltottak a JPA entitások.

A Session Bean-ek az alkalmazás üzleti logikáját megvalósító komponensek. Két altípust különböztetünk meg: állapottal rendelkező (stateful) és állapot nélküli (stateless) SB-k. Az állapottal rendelkezők egy adott kliens munkamenetéhez vannak hozzárendelve és képesek megőrizni állapotinformációkat két metódushívás között, míg az állapot nélküli SB-k nem függnek a hívó féltől és nem is őriznek meg állapotinformációkat. A komponensek különböző interfészeken keresztül érhetik el egy SB szolgáltatásait, ezeket az interfészeket annotációk segítségével határozhatjuk meg:

- `@Local`: ilyen interfészeken keresztül kommunikálnak egymással az egy konténeren belül menedzselt komponensek.
- `@Remote`: az ilyen interfészeken belüli metódusok a külvilág számára is láthatóak, távoli komponensekből is elérhetőek (JNDI alapú dependency lookup mechanizmust és Java RMI alapú kommunikációt alkalmazva).
- `@WebService`: Service EndPoint Interface (SEI), klasszikus, SOAP protokollon alapuló webszolgáltatások esetében alkalmazzuk.

Egy SB több interfészt is implementálhat, illetve arra is lehetőség van, hogy interfész nélküli SB-t hozzunk létre a `@LocalBean` annotáció segítségével. A `@LocalBean` annotáció

használatával az SB publikus metódusai minden lokális komponens számára elérhetővé válnak.

A SoLeBo állapot nélküli SB-eket használ az adathozzáférési és szolgáltatási (üzleti logika) rétegén belül, ezek egymással, illetve a webes komponensekkel lokális interfészeken keresztül kommunikálnak.

Az MDB-k olyan EJB-k, amelyek lehetővé teszik a Java EE alkalmazások számára az aszinkron üzeneteken alapuló kommunikációt. Ezek a komponensek szintén az alkalmazás üzleti logikájáért felelősek, de műveleteiket az üzenetek hatására végzik el (tipikusan Java Message Service –JMS üzeneteket fogadnak). Az EJB konténereknek támogatniuk kell a JMS alapú, javax.jms.MessageListener interfészt implementáló MDB-ket.

A SoLeBo az értesítési funkcionalitás megvalósításánál (pl. közeledő határidőkkel kapcsolatos üzenetek), az emaileket kiküldő modul esetében alkalmaz MDB-ket.

1.3. Context and Dependency Injection (CDI)

A SoLeBo szerver oldali komponensei között a függőségek a Dependency Injection (DI) tervezési minta (az Inverson of Control – IoC módszer egy formája) alapján vannak megoldva. A komponenseket az EJB konténer menedzseli, ez felelős a példányosításért és az EJB annotációk alapján meghatározott függőségek kezeléséért. A rendszer az EJB-specifikáció saját DI módszerén kívül alkalmazza a JSR-299 – Context and Dependency Injection (CDI) [16] szabványt is.

1.4. Interceptorok

Az interceptorok metódusokhoz és komponensekhez rendelhetőek hozzá. Metódushívásokat interceptálnak, illetve a célkomponensek életciklusában bekövetkező eseményekre reagálnak. Az aspektusorientált nyelvekhez hasonlóan az átmetsző követelmények megvalósítására és modularizálására szolgálnak, lehetővé teszik pl. naplózási és különböző elő-, utófeldolgozási műveletek végrehajtását.

A SoLeBo alkalmazáson belül a szolgáltatási rétegen belüli validációt biztosítják. Például regisztrációkor a felhasználó emailcímének egyediségét ellenőrzik, új csoport létrehozásakor a csoportnév egyediségét vizsgálják.

1.5. Java Message Service

A Java Message Service (JMS) API az üzenetalapú Java fejlesztéseket támogatja, különböző MOM (Message Oriented Middleware) rendszerek fölötti absztrakciós szintet képez. A SoLeBo esetében szoftverkomponensek közötti üzenetküldés feladatát látja el. A rendszer a felhasználói értesítések kiküldésekor egy laza kommunikációs csatornát használ:

egy kiadó/feliratkozó modellt, melyben a kiadói szerepet az üzenet objektumokat létrehozó és várakozási listára helyező Session Bean-ek töltik be, a feliratkozó szerepet az ezeket kiküldő Message Driven Bean-ek látják el.

1.6. Időzített szolgáltatások (Timer Service)

A Java EE három különböző időzített szolgáltatást biztosít: időzíthetünk úgy, hogy egy konkrét időpillanatban aktiválódjon az időzítő, úgy, hogy bizonyos időközönként, vagy úgy, hogy a szolgáltatás műveletei egy adott periódus eltelte után legyenek végrehajtva. A SoLeBo ez utóbbit alkalmazza, abban a pillanatban, amikor a kölcsönzés lejártakor figyelmeztető értesítést ki kell küldeni.

1.7. JAX-RS, RESTful webszolgáltatások

A JAX-RS (Java API for RESTful Web Services) [17] olyan Java API, amely lehetővé teszi REST (Representational State Transfer) architektúrájú alkalmazások fejlesztését. Hivatalos része a Java EE 7 szabványnak, és a JSR-311 specifikálja. Használatához nem szükséges konfiguráció, a publikálandó metódusok felannotálásával a JAX-RS elvégzi a megfelelő lépéseket. Ezek az annotációk a következők lehetnek:

- `@Path`: erőforrást elérési útvonala;
- `@GET, @PUT, @POST, @DELETE`: jelzik, hogy milyen HTTP kérésen keresztül szeretnénk elérni az erőforrást;
- `@Produces`: megadja a válasz formátumát, MIME típusát;
- `@Consumes`: az erőforrás által elfogadott adatttípus meghatározása;
- `@PathParam, @QueryParam, @HeaderParam, @CookieParam, @MatrixParam, @FormParam`: a paraméter forrását specifikálja.

1.8. Security

Egy Java EE alkalmazás komponenseinek biztonságát az őket menedzselő konténerek biztosítják. Ennek megvalósítására az alap mechanizmus a JAAS (Java Authentication and Authorization Service) alapú biztonság, a SoLeBo is ezt alkalmazza. A JAAS olyan API-k összessége, amelyek ellenőrzik egy felhasználó hitelességét, illetve hozzáférési jogát bizonyos szolgáltatásokhoz.

A JSR-196 Login Bridge Profile (LBP) specifikáció lehetővé teszi egy alkalmazás szerver számára a hitelesítést (Server Authenication Module – SAM), hogy delegálhassák a JAAS bejelentkezési modulját (LoginModule). A SoLeBo projekt esetében a fejlesztők a Glassfish alkalmazásszerver által támogatott JDBCRealm-et hoztak létre a felhasználók

bejelentkeztetésére és jogainak menedzselésére, tehát a felhasználókkal és a szerepkörökkel kapcsolatos adatokat a rendszer belső adatbázisából kapja az alkalmazásszerver.

2. A Vaadin keretrendszer

A SoLeBo alkalmazás webes felhasználói felülete a Vaadin [18][19] keretrendszert felhasználva van felépítve. A *Vaadin* egy nyílt forráskódú keretrendszer, amely asztali alkalmazásokéhoz hasonló felülettel rendelkező, Rich Internet Application szoftverek fejlesztését segíti elő.

Előnye, hogy egy komponenskészletet bocsájt rendelkezésre, melyekből Java nyelvben, hatékonyan összerakható a felhasználói felület. A komponensek összeköthetőek Java gyűjteményekkel, objektumokkal és ezek adattagjaival, egyszerűen megoldható a validáció és a felület, valamint a modell frissítése.

A Vaadin komponensek tárháza bővíthető, kiterjeszthető a *Google Web Toolkit* (GWT) eszköztár, HTML és JavaScript használatával. A stílusozás a *Cascading Style Sheets* stílusleíró nyelv segítségével történik.

3. Az Android platform

Az Android [20][21] egy Google által fejlesztett, Linux kernelre épülő, mobil operációs rendszer, leginkább telefonokon és táblagépeken fut, de használata előfordul már más típusú eszközökön is (pl. navigációs rendszerek, fedélzeti számítógépek stb.). Az operációs rendszer elterjedése miatt lett az Android az első SoLeBo által támogatott platform.

Android alkalmazást a programozók Java kódban írhatnak, a Google által fejlesztett programkönyvtárat, az Android SDK-t használva. A Java állományok fordítás után Dalvik állományokként egy Dalvik virtuális gépen futnak. A SoLeBo telefonos alkalmazása a NetBeans fejlesztőkörnyezetben készült, az NBAndroid plugint és az Android Maven Plugint használva.

3. A SoLeBo projekt

3.1 Fontosabb követelmények, funkcionalitások

A SoLeBo alkalmazás négy alrendszerből tevődik össze.

3.1.1 SoLeBo Server

A Server megvalósítja az adathozzáférési réteget és az alkalmazáslogikát, illetve ennek elérhetővé tételét egy szolgáltatási rétegen keresztül, így kiszolgálja a Web UI-t és az JAX-RS API-t. Főbb funkcionalitásai:

- Adathozzáférési réteget biztosít a rendszerben tárolt adatok menedzsmentjéhez.
- Üzleti logika réteget kínál a csoportokon belüli információ-megosztáshoz, kölcsönzések követésére, menedzselésére, keresésekhez stb.
- Szolgáltatási rétegen keresztül kommunikál a kliensalkalmazásokkal.
- Automatikus értesítéseket küld egy lefoglalt tárgy felszabadulásakor és a határidők lejárta előtt.

3.1.2 SoLeBo Web UI

A Web UI feladata a webes felhasználói felületek biztosítása. Főbb funkcionalitásai:

- Lehetőséget nyújt regisztrálásra, bejelentkezésre, a felhasználói adatok módosítására.
- Lehetővé teszi a csoportok menedzselését: egy csoport létrehozását, tagok meghívását, eltávolítását, jelentkezések elfogadását, elutasítását stb.
- Biztosítja a tárgyak bevezetését a rendszerbe, majd ezek menedzselését, igénylését, láthatóvá tételét, a kölcsönzések, foglalások követését stb.
- Biztosítja a felhasználóknak a jogosultságaik szerinti funkcionalitások elérhetőségét.
- Az adminisztrátori jogosultsággal rendelkező felhasználóknak lehetőséget ad a tárgykategóriák menedzselésére.

3.1.2 SoLeBo Android

Feladata a telefonos alkalmazáson keresztül elérhető felületek és funkcionalitások biztosítása, melyek közül a fontosabbak:

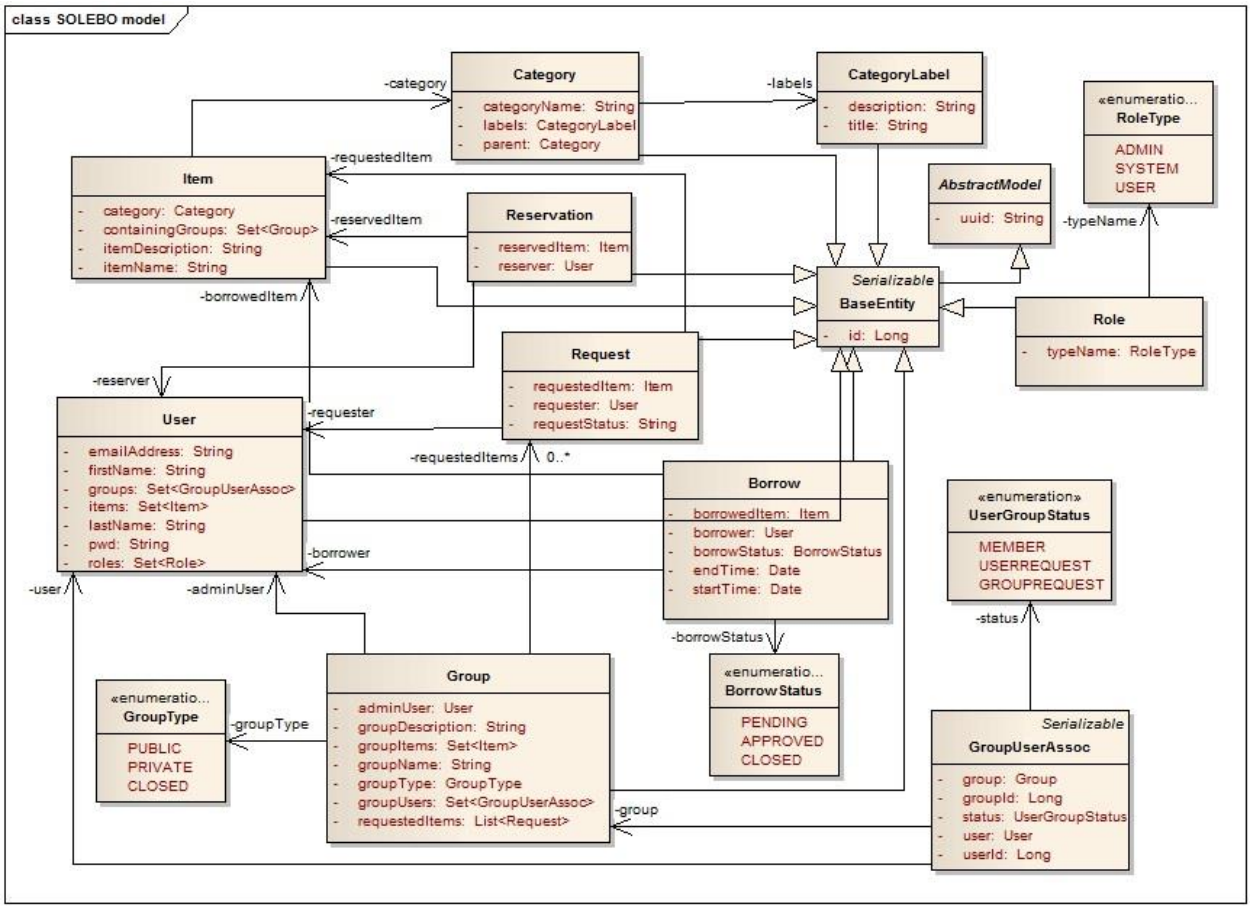
- Lehetőséget biztosít a már regisztrált felhasználók bejelentkezésére.
- Lehetőséget ad a felhasználó saját, illetve kölcsönvett tárgyainak, valamint ezek kölcsönzési állapotainak megtekintésére.
    - Lehetővé teszi kölcsönzések létrehozását, megerősítését, valamint lezárását.

3.1.3 SoLeBo API

Feladata a kommunikáció biztosítása a Server és az Android modulok között, a szerver bizonyos szolgáltatásainak elérhetővé tétele a telefonos kliens számára.

3.2 Környezeti elemzés

Az *edu.codespring.solebo.backend.model* csomag a SoLeBo szoftver központi entitásait reprezentáló JavaBean-eket tartalmazza, ezt szemlélteti az 1. ábra.



1. ábra: *A modell osztályok vázlatos diagramja (hiányoznak a konstruktorok, a getter, illetve setter metódusok, valamint az Object ősosztály újradefiniált metódusai)*

A modell osztályok kiterjesztik a `BaseEntity` osztályt, amely biztosítja az elsődleges kulcsnak megfelelő azonosítót, implementálja a java.io.Serializable interfészt, és a maga során kiterjeszti az `AbstractModel` absztrakt osztályt, amely az UUID (Universally Unique Identifier) generálásáért felelős minden objektum számára.

Az `Borrow` osztály a kikölcsönzött tárgyakra vonatkozó információkat tárolja. A *`BorrowStatus`* enum határozza meg, hogy egy kölcsönzés éppen milyen állapotban van.

Az `Category` osztály a tárgyakat osztályozó kategóriákat reprezentálja.

Az `CategoryLabel` osztály a különböző kategóriákhoz tartozó címkéket és leírásokat tartalmazza (pl. a sportszer kategória címkéi a típusa és a tulajdonságai; a könyv kategória címkéi a cím és író, valamint a rövid leírás).

Az `Item` osztály példányai a tárgyakat reprezentálják.

Az `Reservation` osztály a már kikölcsönzött tárgyakra vonatkozó foglalási kéréseket rögzíti.

Az `Request` osztály a rendszerben még nem szereplő tárgyra vonatkozó kéréshez tartozó adatok tárolására alkalmas. Tartalmazza a tárgyat, amelyet kértek, a felhasználót, aki bejelentette a kérést, valamint a kérés státuszát (hogy teljesítették-e).

A `User` osztály a rendszer felhasználóinak adatait tárolja.

A `Group` osztály a rendszerben létrehozott csoportokat, illetve az ezekre vonatkozó információkat tárolja.

A `GroupUserAssoc` osztály a rendszer felhasználói és csoportjai közt lévő kapcsolatot reprezentálja. Ezt a `UserGroupStatus` enum segítségével valósítja meg, melynek értelmében, ha van kapcsolat egy felhasználó és egy csoport között, akkor a felhasználó vagy tagja a csoportnak (*MEMBER*), vagy kérte felvételét a csoportba (*USERREQUEST*), vagy a csoport vezetője küldött neki meghívót a csoportba (*GROUPREQUEST*).

Az `Role` osztály a `RoleType` enum felhasználásával határozza meg a rendszeren belül érvényes szerepköröket.

3.3 Architektúra

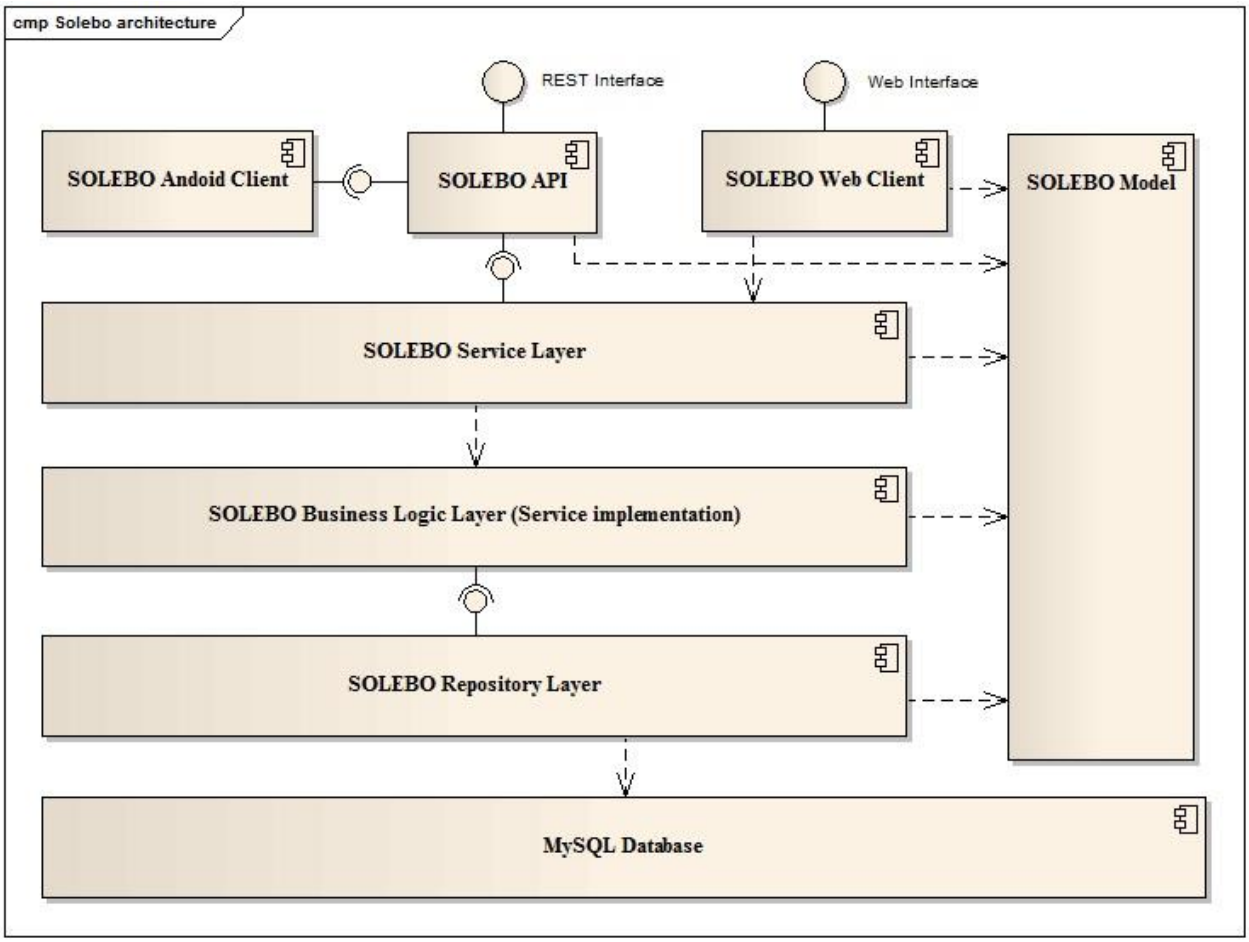
Az 22. ábrán a SoLeBo alkalmazás rétegei, valamint az ezek közötti kapcsolatok láthatóak.

A SoLeBo alkalmazás az adatait egy MySQL relációs adatbázisban tárolja. A *Model* csomag tartalmazza a SoLeBo fő entitásainak megfelelő osztályokat, a *Repository* réteg feladata az ezekkel végzett adatbázis műveletek megvalósítása.

A *Business Logic* réteg a Repository rétegtől kapott adatokkal végzett műveleteket implementál, melyek a *Service* rétegben meghatározott interfészeken keresztül válnak elérhetővé a többi modul számára.

A SoLeBo *Web Client* hozza létre a webes felhasználói felületet, illetve az ezen beérkező kérésekre reagál a Service réteg funkcionalitásait felhasználva.

A SoLeBo *API* biztosítja, hogy a program egyes funkcionalitásai elérhetőek legyenek más alkalmazások számára is, ezt használja az *Android* kliensalkalmazás is.



2. ábra: *A SoLeBo projekt architektúrája*

3.4 Rövid összefoglaló a megvalósításról

3.4.1 SoLeBo Backend

Ebben a modulban kapnak helyet a modell osztályok, a repository, az üzleti logika, illetve a szolgáltatási réteg. A modell osztályok JPA entitások, amelyek leképzését az adatbázisba a `persistence.xml` konfigurációs állományban megadott adatforrás beállításai alapján az EclipseLink valósít meg. A modellek szintjén történő ellenőrzés a Bean Validation (JSR 303) specifikáció alapján, annotációkat alkalmazva biztosított. A repository és üzleti logika rétegen belül többnyire állapot nélküli session bean-ek felelősek a műveletek megvalósításáért, amelyek egymás között és a webes komponensekkel lokális interfészeken keresztül kommunikálnak. A repository rétegen belüli komponensek hierarchiába szervezettek, az alapvető műveletek egy közös absztrakt alaposztályba vannak kiemelve. A lekérdezések JPQL-t és Criteria Query API-t alkalmazva vannak megvalósítva. Az üzenetküldésért felelős komponensek Message Driven Bean-ek, amelyek JMS üzenetekre reagálva végzik el feladataikat.

3.4.2 SoLeBo Web User Interface

A webes felhasználói felület megvalósítása a Vaadin keretrendszeren alapszik, amely a beépített Java komponensekből felépített felület alapján generálja a megfelelő JavaScript, HTML és CSS állományokat. A Vaadin esetében a szerver oldali réteg Java Servlet technológiát használ, a szerver és a kliens oldal kommunikációja Ajax modellen alapszik.

A modul a SoLeBo Backend szolgáltatásait a CDI mechanizmust kihasználva éri el. A nézetek közötti kommunikáció a Java EE eseményeket figyelő kontrollereinek segítségével történik.

3.4.3 SoLeBo API

Az API modul komponenseinek fejlesztése a JAX-RS szabványra épülő Jersey keretrendszer segítségével történt. A kommunikáció a kliens és a szerver között a DTO (Data Transfer Object) tervezési mintán alapszik. Mind szerver, mind kliens oldalon Assembler komponensek felelősek a DTO-modell és modell-DTO megfeleltetésekért. Az adatátvitel JSON formátumban történik. A JSON szerializálást és deszerializálást a Jersey a JAX-B szabványra épülő Jackson keretrendszeren keresztül biztosította.

3.4.4 SoLeBo Android Client

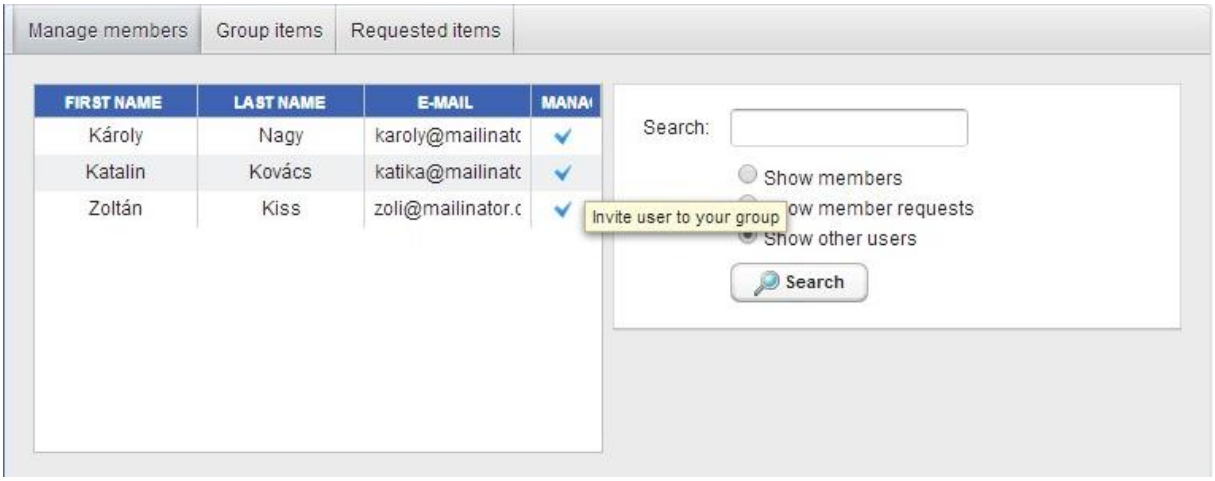
Az Android mobilalkalmazás felületének kialakításakor az átláthatóság, a komponensek logikus elrendezése és könnyen kezelhetősége, valamint a platform esetében általánosan kedvelt nézetek kialakítása volt a cél. A felületen keresztül beérkező felhasználói kérések által kiváltott események a kontrollereknek továbbítódnak. A kontrollerek implementálják a kliensalkalmazás üzleti logikáját, kommunikálva az adathozzáférési réteggel, az API-n keresztül elérve a szerver szolgáltatásait.

Az adathozzáférési réteg menedzseli az adatbázis tartalmát, az adatok lokális tárolása SQLite relációs adatbázis-kezelő rendszeren belül történik.

A szerver szolgáltatásait a mobil alkalmazás HTTP kéréseken keresztül éri el. Az elküldött üzenetek JSON objektumok, melyeket a Jackson feldolgozó hoz létre a DTO-nak megfelelő Java objektumokból.

3.5 Használati esettanulmány

A következő példa szemlélteti a SoLeBo rendszer működését. A példában három fiktív felhasználó fog szerepelni: Bandi, Katika és Karcsi. Bandi a rendszer felhasználójává azáltal válik, hogy regisztrál a rendszerbe. A funkcionalitás a bejelentkezési felületről érhető el. Hasonlóan jár el Katika is. Bandi egy Kódforrás nevű új csoportot hoz létre. Azáltal, hogy ő a csoport létrehozója, jogosultságot szerez arra, hogy felhasználókat hívjon a csoportba. Ilyen módon meg tudja keresni Katikát az elérhető felhasználók listájában, és meghívót tud küldeni neki (3. ábra).

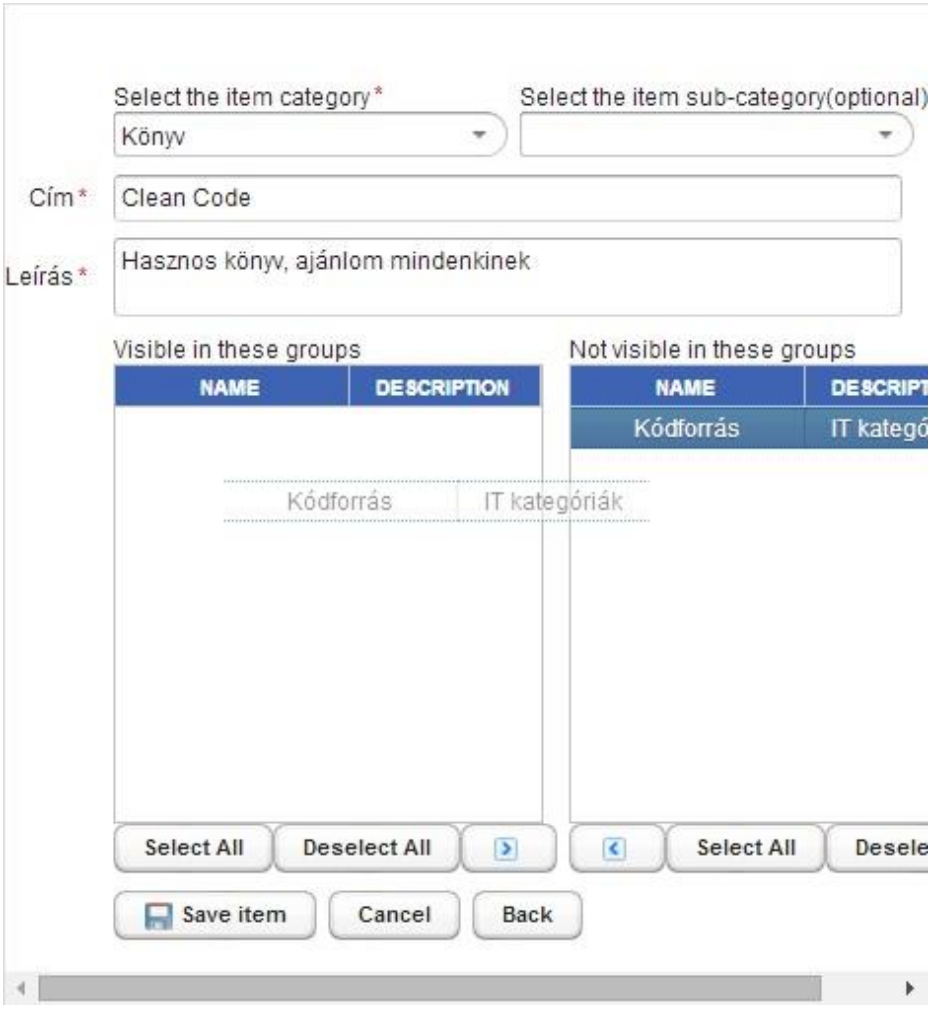


3. ábra: *Felhasználók listája, szűrési lehetőségek, valamint meghívás opció*

Katika a maga során elfogadja a meghívást a Kódforrás nevű csoportba (4. ábra).

4. ábra: *Meghívások listája és elfogadási, illetve visszautasítási lehetőség*

A továbbiakban Bandi kölcsönözhetővé teszi a Kódforrás csoportban a *Tiszta kód* című könyvet (5. ábra).

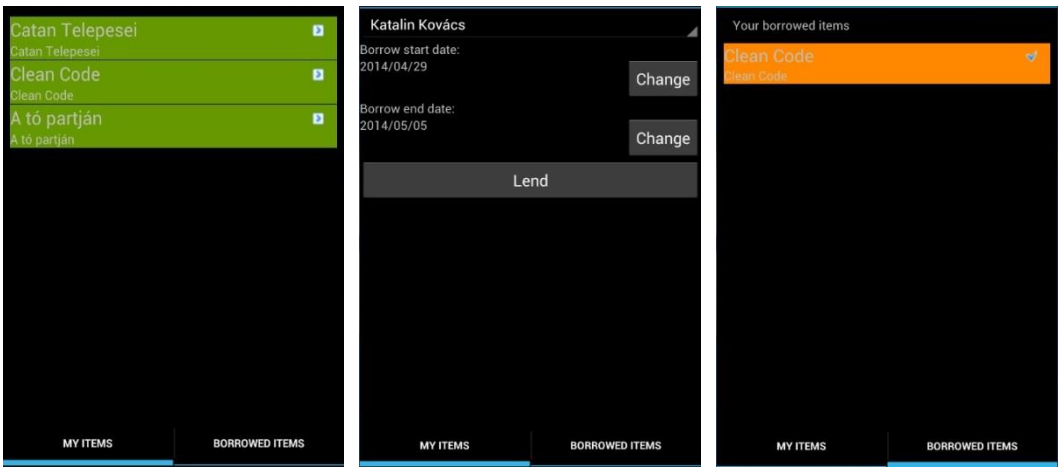


***5.* ábra**: *Tárgy közzététele csoportokon belül*

Katika tagja a Kódforrás csoportnak, így jelezheti kölcsönzési szándékát a *Tiszta kód* című könyvre. Mivel a tárgy jelenleg nincs kikölcsönözve, Katika kikölcsönözheti azt. Bandival e-mailen egyeztetnek helyszínt és időpontot. Bandi találkozik Katikával, hogy átadja neki a könyvet. Miután megegyeznek a kölcsönzés időtartamában Bandi a SoLeBo mobil alkalmazás segítségével a helyszínen bevezeti a kölcsönzésre vonatkozó adatokat a rendszerbe. A saját tárgyak listájának különböző színű sorai a tárgy kölcsönzési státuszát hivatottak jelezni: a zöld a lekérdezés pillanatában szabad tárgyaknak felel meg, a narancssárga a kölcsönzés megerősítésére váró tárgyakat jelöli, a piros színű tárgyak esetében a kölcsönzés nem lehetséges, mivel foglalt tárgyakról van szó (6. ábra).

Katika a kölcsönzés megerősítésével tudja jelezni, hogy ténylegesen megtörtént a kölcsönzés (6. ábra).

Az eddig leírt lépések után egy harmadik felhasználó, Karcsi is regisztrál a rendszerbe, aki jelzi csatlakozási szándékát a Kódforrás nevű csoporthoz.



***6.* ábra:** *Mobilalkalmazás nézetei: saját tárgyak listája, tárgy kölcsönadása, kölcsönzött tárgyak listája, illetve kölcsönzés megerősítésének lehetősége*

Bandi a felületen látja Karcsi csatlakozási szándékát, ezt elfogadhatja, vagy elutasíthatja. A kérés elfogadása után Karcsi számára láthatóvá válnak a csoportban publikussá tett tárgyak, ennek megfelelően a *Tiszta kód* című könyv is, a csoport kikölcsönzött tárgyainak listájában. Amennyiben az aktuális kölcsönzés lejárta után Karcsinak szüksége lenne az említett könyvre, jelezheti kölcsönzési szándékát, lefoglalhatja azt (8. ábra).

***7.* ábra:** *Kikölcsönzött tárgyak listája és foglalás bejelentésének lehetősége*

A kölcsönzési határidő lejártának közeledtével Katika email formájában kap értesítést arról, hogy hamarosan vissza kell adnia a könyvet Bandinak. Ha Katika visszaadja Bandinak a kikölcsönzött könyvet, Bandi lezárja a kölcsönzést. Erről Karcsi e-mailben értesül (mivel foglalása volt a könyvre), ezután jelezheti kölcsönzési szándékát Bandi felé.

4. Következtetések és továbbfejlesztési lehetőségek

A SoLeBo projekt fejlesztésének közel egy éve alatt sikerült egy olyan kölcsönzési szolgáltatást létrehozni, amely a kisebb, zárt csoportok (pl. szakmai közösségek) tevékenységét támogatja. Sikerült egy olyan rendszert kifejleszteni, amely támogatja ezeknek a csoportoknak a kezelését, valamint a bevezetett tárgyak állapotának követését. A tárgyak menedzselésén, kategorizálásán és böngészésén kívül a felhasználók segítségére vannak az időzített automatikus értesítések, valamint egy mobil alkalmazás, amellyel lehetővé válik a kölcsönzések helyfüggetlen menedzselése.

A SoLeBo jelenlegi állapotában egy kezdeti verzió, amely még számos bővítési lehetőséget rejt magában. Az rendszer tervezése és fejlesztése során felmerült fontosabb ötletek:

- Az alkalmazás együttműködésének támogatása népszerű közösségi hálózatokkal. Pl. egy másik közösségi hálózat felhasználói fiókjával történő bejelentkezés, az itt létrejött csoportok integrálására, közzétett tárgyak publikálása az üzenőfalon stb.
- A tárgyak véleményezése és értékelése lehetőséget adna arra, hogy a tulajdonos és a kölcsönzők értesüljenek, pl. a tárgyak aktuális fizikai állapotáról.
- A kisebb, specifikus csoportok tagjai megbíznak egymásban, de egy belső értékelés és jutalomrendszer tovább növelné a bizalmat a kölcsönadó és a kölcsönző között.
- A felhasználók közötti kommunikáció integrálása a rendszerbe (pl. csevegő szolgáltatáson és fórumon keresztül) kiküszöbölné az emailszolgáltatások használatát, biztosítva a kölcsönzés részleteinek biztonságos, rendszeren belüli megbeszélését.
- A webes adminisztrációs felület jelenleg a kategóriák kezelésére ad lehetőséget. Ez kiegészíthető a csoportok, felhasználók, tárgyak menedzsmentjével, egyfajta szabályozási rendszer bevezetésével, valamint összetett keresési lehetőségekkel, különböző statisztikák generálásával stb.
- A mobil alkalmazás kibővítése, valamint elkészítése más platformokra, operációs rendszerekre.

5. Hivatkozások

[1] Mercurial Hivatalos Weboldal, utolsó megtekintés dátuma: 2014.04, http://mercurial.selenic.com/

[2] Tortoise HG Hivatalos Weboldal, utolsó megtekintés dátuma: 2014.04, http://tortoisehg.bitbucket.org/

[3] Apache Maven Hivatalos Weboldal, utolsó megtekintés dátuma: 2014.04, http://maven.apache.org/

[4] SonarSource Sonar Hivatalos Weboldal, utolsó megtekintés dátuma: 2014.04, http://www.sonarsource.org/features/,

[5] Redminde Hivatalos Weboldal, utolsó megtekintés dátuma: 2014.04, http://www.redmine.org/

[6] Jenkins CI Hivatalos Weboldal, utolsó megtekintés dátuma: 2014.04, http://jenkins-ci.org/

[7] NetBeans Hivatalos Weboldal, utolsó megtekintés dátuma: 2014.04, https://netbeans.org/

[8] Balsamiq Hivatalos Weboldal, utolsó megtekintés dátuma: 2014.04, http://balsamiq.com/products/mockups/

[9] Glassfish Hivatalos Weboldal, utolsó megtekintés dátuma: 2014.04, https://glassfish.java.net/

[10] MySQL Workbench Hivatalos Weboldal, utolsó megtekintés dátuma: 2014.04, http://www.mysql.com/products/workbench/

[11] Liquibase Hivatalos Weboldal, utolsó megtekintés dátuma: 2014.04, http://www.liquibase.org/

[12] Java EE Hivatalos Weboldal, utolsó megtekintés dátuma: 2014.04, http://docs.oracle.com/javaee/

[13] JPA Hivatalos Dokumentáció, utolsó megtekintés dátuma: 2014.04, http://docs.oracle.com/javaee/7/tutorial/doc/persistence-intro.htm

[14] EJB Hivatalos Dokumentáció, utolsó megtekintés dátuma: 2014.04, http://docs.oracle.com/javaee/7/tutorial/doc/ejb-intro.htm#GIJSZ

[15] Richard Monson-Haefel, Bill Burke, EJB 3.1, O'Reilly Media, 2006

[16] CDI Hivatalos Dokumentáció, utolsó megtekintés dátuma: 2014.04, http://docs.oracle.com/javaee/7/tutorial/doc/partcdi.htm

[17] RESTful WS Hivatalos Weboldal, utolsó megtekintés dátuma: 2014.04, https://www.ibm.com/developerworks/webservices/library/ws-restful/

[18] Vaadin Hivatalos Dokumentáció, utolsó megtekintés dátuma: 2014.04, https://vaadin.com/book

[19] Jaroslav Holaň, Ondřej Kvasnovský, Vaadin 7 Cookbook, O'Reilly Media, 2013

[20] ***, Programming Android - Java Programming for the New Generation of Mobile Devices, O'Reilly Media, 2011

[21] Android, utolsó megtekintés dátuma: 2014.04, https://developer.android.com/guide/index.html