XVIII. reál- és humántudományi Erdélyi Tudományos Diákköri Konferencia (ETDK)

Kolozsvár, 2015. május 21–24.

Újságterjesztést elősegítő szoftverrendszer

Szerzők:

Marton Iulia-Kinga
Babeş-Bolyai Tudományegyetem, Kolozsvár, Matematika és Informatika Kar, Informatika szak, 3. Évfolyam

Szabó Zoltán
Babeş-Bolyai Tudományegyetem, Kolozsvár, Matematika és Informatika Kar, Informatika szak, 3. Évfolyam

Témavezetők:

Dr. Simon Károly
Egyetemi adjunktus, Babeş-Bolyai Tudományegyetem, Matematika és Informatika Kar
Projektmenedzser, Codespring Kft.

Kandó Norbert
Projektmenedzser, Codespring Kft.

Kivonat

A dolgozat egy projektet mutat be, amelynek célja egy újságterjesztést segítő szoftverrendszer kifejlesztése. A rendszer egy központi szerverből, egy webes adminisztrációs felületből és egy mobil kliensalkalmazásból áll. A rendszer adminisztrátorai a webes felületen keresztül kezelhetik az adatokat (megrendelők adatbázisa, terjesztők profiljai, terjesztési ciklusok indítása stb.). A terjesztők a mobil kliensalkalmazás segítségével kapják meg a hozzájuk rendelt megrendelők listáját, útbaigazítást kapnak a kézbesítéshez, majd az egyes helyszíneken visszajelzést küldhetnek a kézbesítéssel kapcsolatban. Az adminisztrációs felület kimutatásokat szolgáltat és ellenőrzési lehetőségeket biztosít a működtetők számára.

Tartalomjegyzék

Bevezető

A dolgozat az “Igen, tessék! Newspaper Distribution” (ITNPD) újságterjesztő szoftver funkcionalitásait, felépítését és megvalósítását mutatja be. A szoftver célja, hogy megkönnyítse az újságterjesztők munkáját. Általánosan használható, de mivel fejlesztése az Igen, tessék! havilap felkérése alapján történt, elsősorban ennek az újságnak a terjesztését hivatott támogatni.

A kereskedelmi többnyelvűséget előtérbe helyező **Igen, tessék!** mozgalmat egy nonprofit civil szervezet indította. A szervezet célja, hogy hálózatba fogja a többnyelvű kiszolgálást, illetve tájékoztatást biztosító vállalkozásokat. A cél elérésének érdekében a szervezet egy újságot is működtet. Az Igen, tessék! közösségi havilap egy 20 oldalas, részben színes kiadvány, amelyet ingyenesen, 18 ezer példányban juttatnak el a kolozsvári és a Kolozsvár környéki települések magyar háztartásaiba. Lapjain a szórakoztató anyagokon kívül – az Igen, tessék! mozgalom eszmeiségének megfelelően – a tudatos anyanyelvhasználatra nevelő, a nyelvhasználati jogok terén tanáccsal szolgáló cikkek is rendszeresen helyet kapnak. Ugyanakkor, a lapban a magyar nyelvű kiszolgálást nyújtó vállalatokat is népszerűsítik.

A szervezetnél jelenleg az adatrögzítés egy Excel állományban történik, ami nagy mennyiségű adat esetén nehezen áttekinthető, illetve nehéz a rendszerezése is. Az újságkihordók nyomtatott változatban kapják meg a címeket, ahova újságot kell, eljuttassanak, amit könnyen elveszíthetnek, valamint az ismeretlen, új címek megtalálása időigényes. Egy harmadik probléma a visszajelzés hiánya a feladatok teljesítését illetően. Ezeken a problémákon igyekszik segíteni az elkészített szoftverrendszer.

A dolgozatban bemutatott projekt keretein belül kifejlesztett szoftverrendszer három fő részből áll: egy központi szerverből, egy webes adminisztrációs felületből és egy mobil kliensalkalmazásból. A rendszer adminisztrátorai a webes felület által kezelhetik az adatokat (megrendelők adatainak, illetve a terjesztők profiljainak menedzsmentje, terjesztési ciklusok kezelése, ellenőrzése stb.). A terjesztők a mobil kliensalkalmazás által kapják meg a számukra kiosztott feladatokat, útbaigazítást kapnak az egyes lakcímekkel kapcsolatban, majd a helyszíneken visszajelzéseket küldhetnek a kézbesítésről (sikeres vagy sikertelen volt a kézbesítés, a sikertelenség oka, a fogadtatás minősége). A visszajelzések alapján a rendszer kimutatásokat készít a kiválasztott terjesztési ciklusról, illetve ellenőrző listákat generál az egyes terjesztők esetében (például a rendszer véletlenszerűen kiválaszt adott számú felhasználót, akiket telefonon felkeresnek az adatok hitelességének ellenőrzése érdekében).

A dolgozat négy részre tagolódik. Az első rész az ITNPD projekt jellemzőit mutatja be, kitérve a funkcionális követelményekre, használati esetekre valamint a projekt architektúrájára. A második rész a rendszer gyakorlati használatát ismerteti, míg a harmadik a megvalósítással kapcsolatos fontosabb részletekre tér ki. Az utolsó rész a fejlesztés során alkalmazott módszereket és eszközöket mutatja be röviden.

A dolgozat témáját képező projekt a Babeș-Bolyai Tudományegyetem Matematika és Informatika Karán belül, egy Csoportos projekt tantárgy keretében indult el, a Codespring Kft. által kiadott specifikáció alapján. Kezdetben a fejlesztői csoport öt tagot számlált, név szerint: Marton Iulia-Kinga, Nagy Gergő, Szász Adorján, Zolcsák Zsolt és Szabó Zoltán. A tantárgy keretein belül egy kezdetleges prototípus készült el, egy tanulmányprojekt, amelyre a későbbi fejlesztések alapozhattak. Ezt követően a projektet a két szerző folytatta, jelenlegi formájában a dolgozat szerzői által lett kidolgozva. A fejlesztéshez szükséges infrastruktúrát és eszközöket a Codespring Kft. biztosította, emellett két szakmai irányítót kért fel a munkafolyamat felügyelésére, név szerint Kandó Norbertet és Simon Károlyt. Önzetlen segítségüket külön köszönet illeti.

1. Az Igen, Tessék Newspaper Distribution projekt

Az ITNPD projekt célja egy olyan szoftverrendszer fejlesztése, amely megkönnyíti az újságok/lapok eljuttatását az olvasókhoz, valamint lehetőséget ad az újságterjesztők munkájának nyomon követésére. A rendszer biztosít egy webes adminisztrációs felületet az újság működtetői számára, ahol nyomon követhetőek és menedzselhetőek az előfizetők és terjesztők adatai, a terjesztési ciklusok (szórások), illetve megtekinthetőek különböző jelentések és statisztikai adatok. A terjesztők egy mobil kliensalkalmazáson keresztül kapják meg a számukra kiosztott kézbesítési feladatokat, útbaigazítást kapnak ezekkel kapcsolatban és visszajelzéseket küldhetnek adott feladattal kapcsolatban.

1.1. Követelmények

A webes adminisztrációs felület az alábbi funkcionalitásokat biztosítja:

- Előfizetők adatainak importálása (Excel állományokból): mivel a fejlesztők nem rendelkeznek az előfizetőkre vonatkozó valós adatokkal, tesztadat generátor is szükséges.
- Előfizetők adatainak listázása, rendezési és szűrési lehetőségek (minden fontosabb adat szerint).
- Előfizetők adatainak módosítása.
- Terjesztők adatainak menedzsmentje.
- Terjesztők profiljának szerkesztése: pl. utcák terjesztőkhöz rendelése.
- Terjesztési ciklus (szórás) indítása: a terjesztési ciklus csak akkor indítható, ha minden adatbázisban szereplő utcához (és így minden előfizetőhöz) volt terjesztő hozzárendelve. Egy régión belül egyszerre csak egy szórás lehet folyamatban.
- Terjesztési ciklusokkal kapcsolatos információk megtekintése: a terjesztőkhöz hozzárendelt megrendelők listázása, a terjesztők visszajelzéseinek megtekintése, szűrési lehetőségek.
- Terjesztési ciklussal kapcsolatos statisztikák megtekintése.

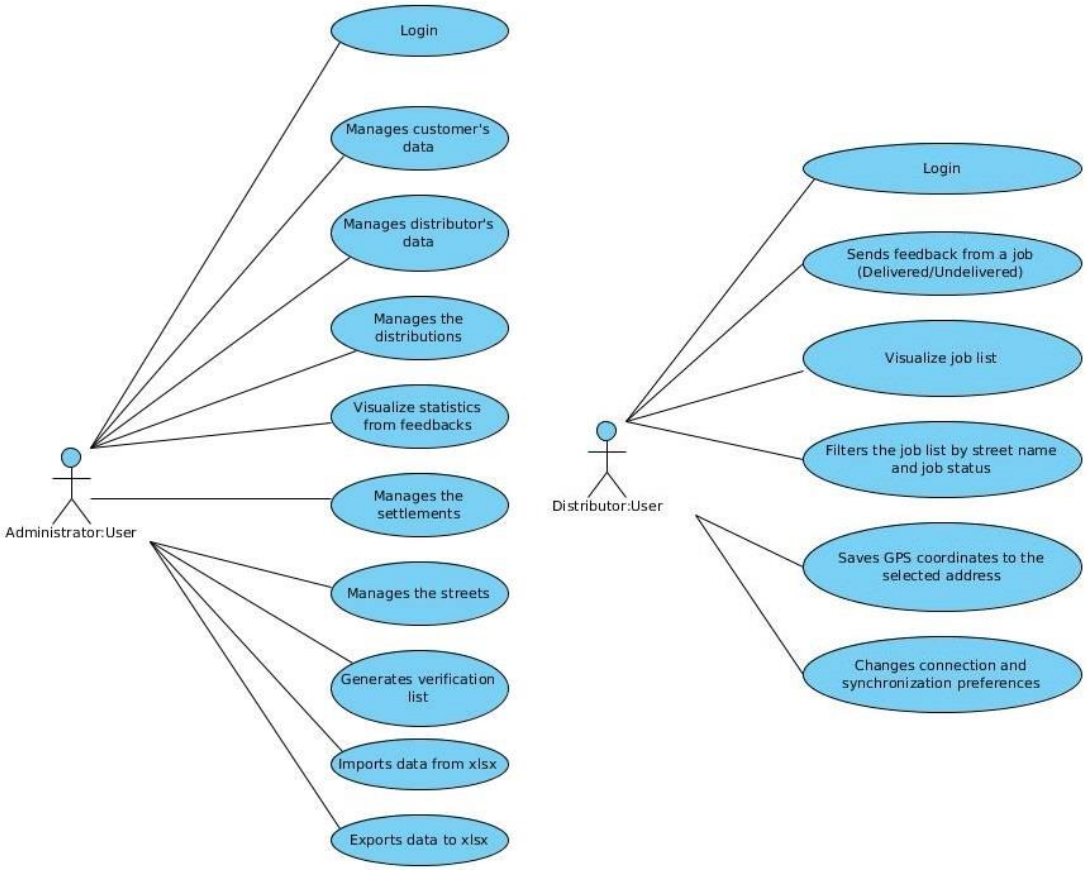
- Ellenőrző listák generálása, valamint visszajelzések rögzítése (a terjesztők pontozása és az előfizető véleménye a szolgáltatásról).

A mobil kliensalkalmazás az alábbi funkcionalitásokat biztosítja:

- A terjesztési ciklus indításakor a terjesztők megkapják a hozzájuk rendelt előfizetők listáját. A lista megtekinthető a telefonos alkalmazáson belül.
- Automatikus útvonal javaslat: a terjesztő aktuális pozíciójának függvényében a telefonos alkalmazás automatikusan javasolja a következő előfizető lakcímét.
- Visszajelzés küldése: adott lakcím esetében a terjesztő visszajelzést küldhet a teljesítés sikerességéről.
- A visszajelzés küldésekor a terjesztő választhat előre definiált standard visszajelzések közül (sikeres kézbesítés, nem nyitottak ajtót, nem voltak otthon stb.), illetve saját megjegyzést írhat.
- Adott lakcím GPS koordinátáinak mentése/felülírása a későbbi pontosabb helymeghatározás érdekében (kezdetben a térképszolgáltató által szolgáltatott koordinátákat használja a rendszer, később ezek a helyszínen „pontosíthatóak”).
- Adatok lokális tárolása és későbbi szinkronizálása a szerverrel (hálózati kapcsolat ideiglenes hiányában is használhatónak kell maradnia az alkalmazásnak).

1.2. Használati esetek

A következő leegyszerűsített use-case diagram (1. ábra) szemlélteti a rendszer fontosabb használati eseteit, az adminisztrátorok, illetve terjesztők szemszögéből.



1. ábra: Use case diagram

1.3 Architektúra

Az ITNPD szoftverrendszer két nagyobb részre osztható fel: egy szerverre és egy mobil kliensalkalmazásra.

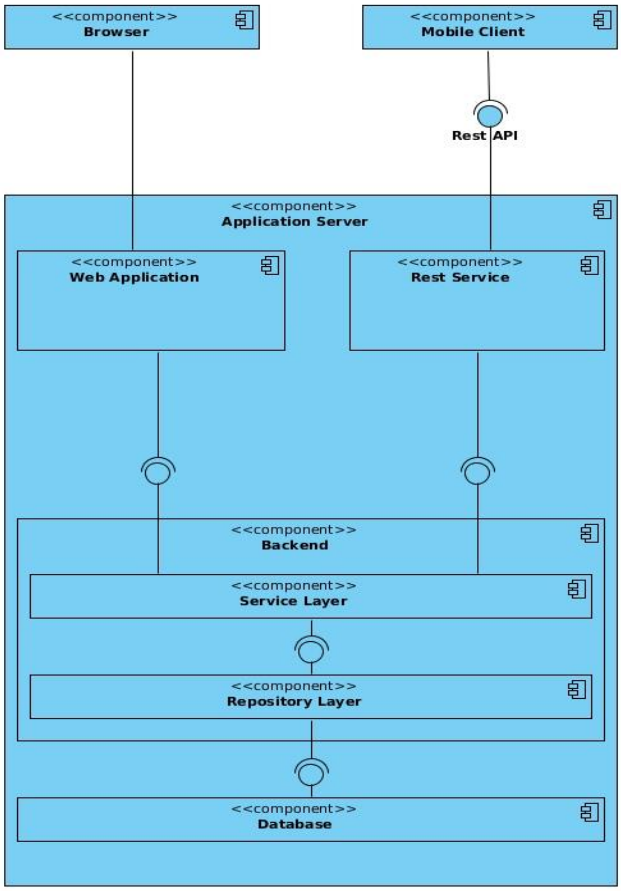
A szerver további alrendszereket is magába foglal. Biztosítja az adatmodellt és kommunikál a relációs adatbázis-kezelő rendszerrel, biztosítva az adathozzáférési réteget (Repository Layer), amelyen belül a perzisztens adatok kezelése történik. Az alkalmazáslogikával kapcsolatos műveletek a szolgáltatási rétegen (Service Layer) belül vannak implementálva és interfészeken keresztül érhetőek el a többi alrendszer számára.

A web alkalmazás (Web Application) modul a webes komponenseket tartalmazza. Ezek felelősek a webes adminisztrációs felület felépítéséért, a kérések fogadásáért, illetve a válaszok továbbításáért. Az újság működtetői egy webes böngészőben jeleníthetik meg az adminisztrációs felületet és ezen a felületen belül érhetik el a számukra biztosított funkcionalitásokat.

Bizonyos funkcionalitásokat a szerver webszolgáltatások formájában tesz elérhetővé. A REST Service modul ezeket a szolgáltatásokat implementálja.

Az Android platformra megírt mobil kliensalkalmazás a terjesztők számára biztosít funkcionalitásokat. Ez az alrendszer az előzőleg említett RESTful webszolgáltatásokon [1] keresztül kommunikál a szerverrel.

A következő diagram (2. ábra) a rendszer fontosabb alrendszereit és az azok közötti kapcsolatot szemlélteti.



2. ábra: Az ITNPD arhitekturája

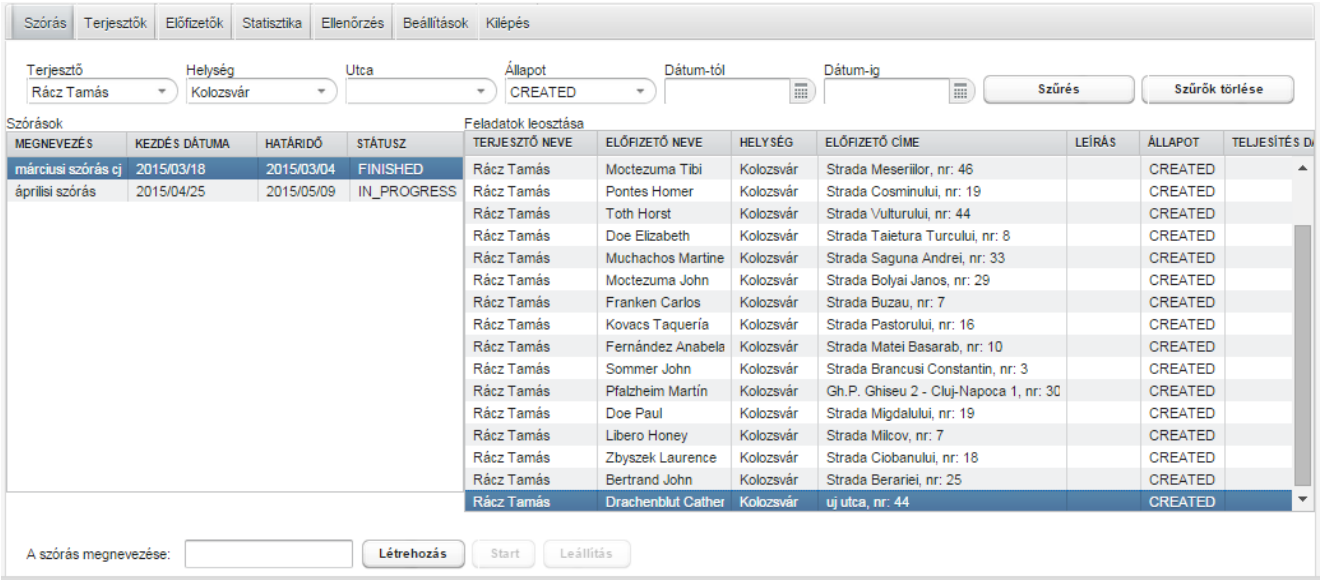
2. Az Igen, Tessék Newspaper Distribution projekt a gyakorlatban

Az ITNPD szoftverrendszer funkcionalitásaihoz az újságok működtetői a webes adminisztrációs felületen keresztül férnek hozzá, a terjesztők a mobil kliensalkalmazást használhatják erre a célra.

2.1. A webes adminisztrációs felület

A szoftverrendszer webes része a terjesztést menedzselő szervezet (az újság működtetője) számára nyújt lehetőséget az adatok kezelésére, statisztikák megtekintésére valamint ellenőrző listák készítésére. Ezeknek a funkcionalitásoknak az igénybevételéhez előzetes bejelentkezés szükséges.

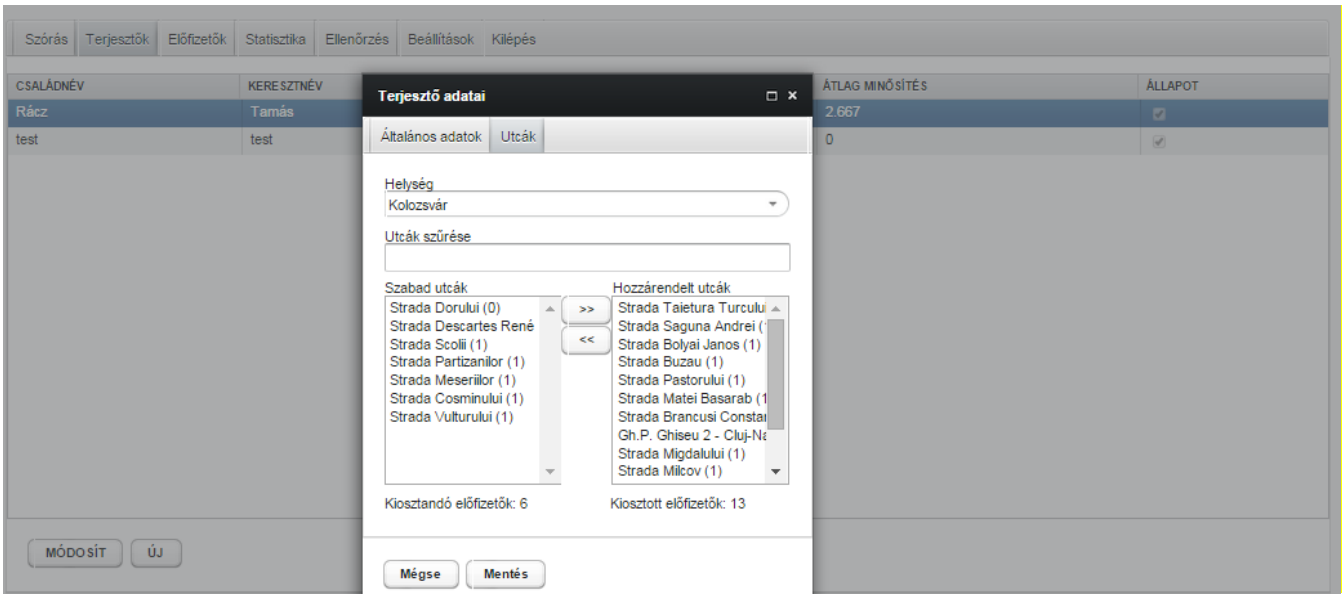
A felület hat nagyobb részből épül fel: *Szórás, Terjesztők, Előfizetők, Statisztika, Ellenőrzés*, valamint *Beállítások*.



3. ábra: Az ITNPD webes felülete

A *Szórás* fül lehetőséget biztosít a szórások (terjesztési ciklusok) kezelésére, beleértve ezek létrehozását, indítását és leállítását. Ugyanakkor lehetőséget biztosít a szórásokhoz tartozó feladatok adatainak a megtekintésére és szűrésére terjesztő, helység, utca, állapot, illetve dátum szerint, valamint ezeknek a szűrési kritériumoknak bármilyen kombinációja szerint.

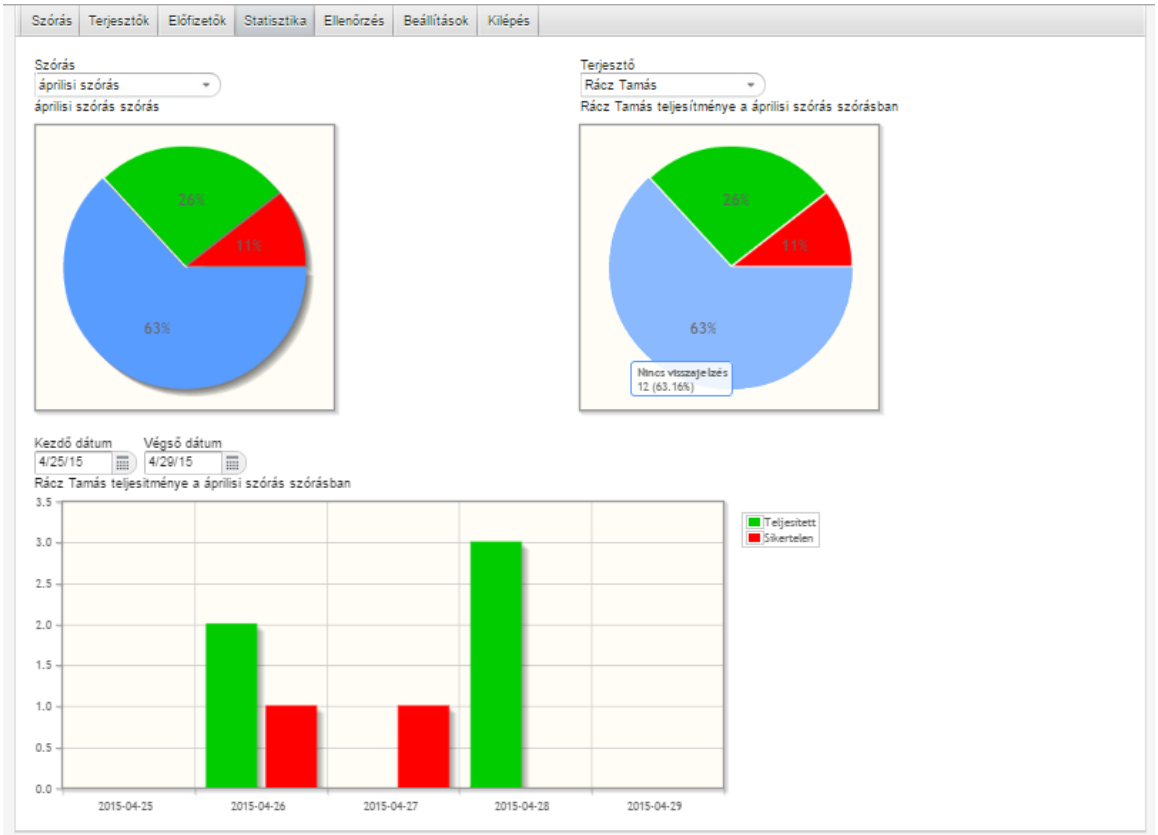
A *Terjesztők* rész alatt az újságkihordók adatait lehet szerkeszteni. Lehetőség van új terjesztők felvitelére, terjesztők inaktiválására, a régi adatok módosítására, valamint az utcák terjesztőkhöz való hozzárendelésére. Ugyanitt megtekinthető mindegyik terjesztő esetében egy átlagminősítés, az előfizetők által adott visszajelzések alapján.



4. ábra: Terjesztők adatainak szerkesztése a webes felületen

Az *Előfizetők* részen az olvasók adatait lehet módosítani, új előfizetőket lehet hozzáadni, valamint a meglévő előfizetők listáját szűrni lehet különböző szempontok szerint: név, helység, utca, állapot.

A terjesztők által küldött visszajelzések alapján a rendszer kimutatásokat készít a kiválasztott terjesztési ciklusról. A *Statisztika* fül ezeket a kimutatásokat tartalmazza. Megtekinthető, hogy egy bizonyos szóráson belül a feladatok hány százaléka lett teljesítve, mennyi a sikertelen feladat és mennyi feladat van még hátra. Ugyanezek az adatok megtekinthetőek egy adott terjesztőre is. Egy harmadik jelentés egy kiválasztott szórásra és terjesztőre kimutatja az illető terjesztő teljesítményét napi bontásban, két megadott dátum között.



5. ábra: Webes felület, statisztika oldal

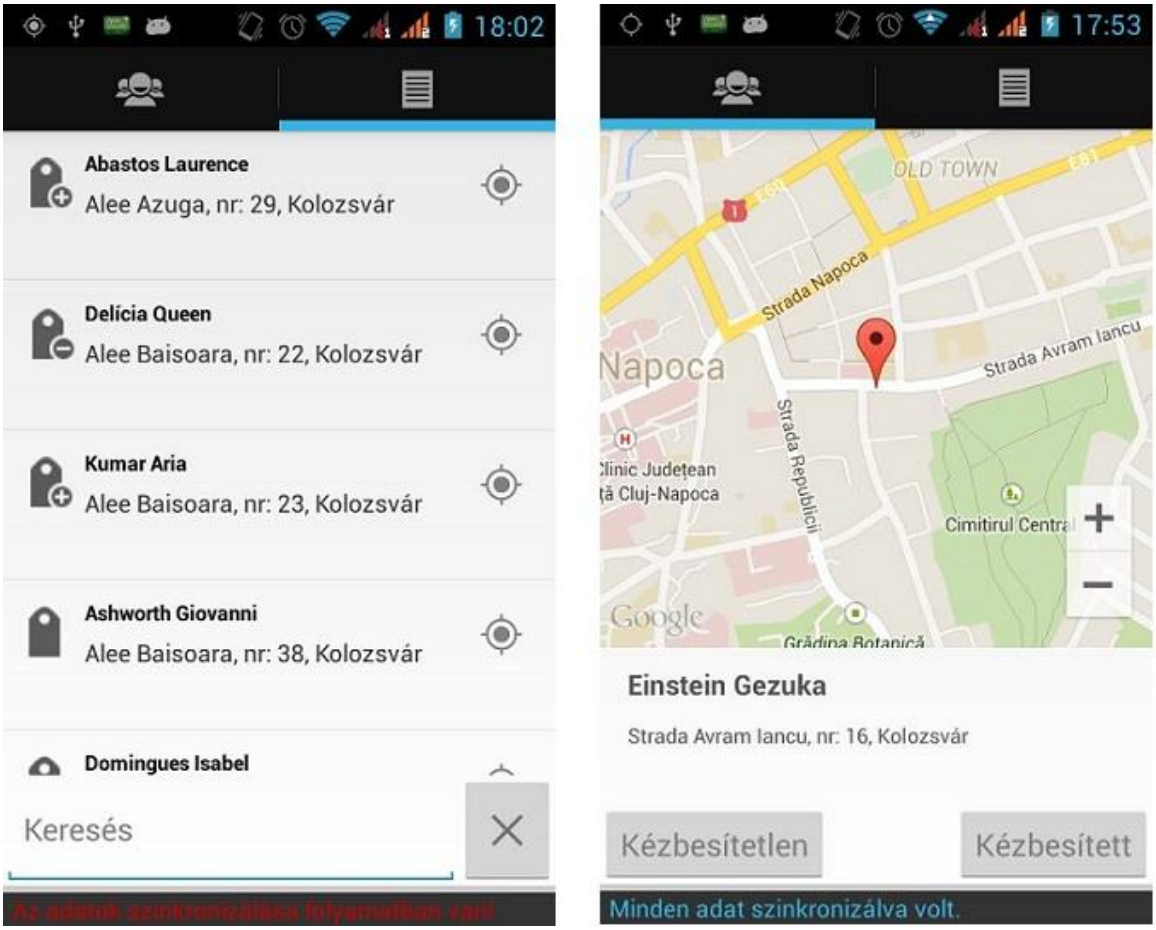
Az alkalmazás ellenőrző listákat is generálhat az egyes terjesztők esetében (például a rendszer véletlenszerűen kiválaszt adott számú felhasználót, akiket telefonon felkeresnek az adatok hitelességének ellenőrzése érdekében). Az ellenőrző listák kezelésére, valamint az előfizetőktől kapott visszajelzések rögzítésére alkalmas az *Ellenőrzés* fül.

A *Beállítások* menüpont alatt lehetőség van az előfizetők listájának az exportálására Excel állományba, valamint egy lista importálására Excel állományból. Az importálás során konfigurálható, hogy hogyan legyenek kezelve azok az adatok, amelyek az adatbázisban szerepelnek, de az állományban nem, illetve azok az adatok, amelyek megtalálhatóak az állományban, de nem szerepelnek az adatbázisban.

2.2. A mobil kliensalkalmazás

A mobil kliensalkalmazás egy Android operációs rendszerre fejlesztett alkalmazás, amely REST-API-n keresztül kommunikál a szerverrel. Fő feladata, hogy a terjesztők megkapják a számukra kiosztott feladatokat, útbaigazítást kapjanak az egyes lakcímekkel kapcsolatban, majd a helyszíneken visszajelzést tudjanak küldeni a kézbesítésről (sikeres vagy sikertelen volt a

kézbesítés, az esetleges sikertelenség oka). Az alkalmazás két főbb nézetet tartalmaz: egy térképes nézetet, amely segít a terjesztőnek eljutni az adott címhez, illetve egy lista nézetet, amely tartalmazza a terjesztő számára kiosztott feladatokat. A két nézetet az alábbi 6. ábra szemlélteti.



6. ábra: A mobil alkalmazás nézetei

A térképes nézeten a térkép alatt megjelennek a kliens adatai (például a neve, lakcíme, illetve egy megjegyzés, ami tartalmazhatja a kézbesítendő darabszámot). Ezen kívül a nézet tartalmazza a *kézbesített*, illetve *kézbesítetlen* gombokat, amelyekkel a terjesztő visszajelzést küldhet. A *kézbesítetlen* gomb megnyomásakor egy felugró ablak jelenik meg, ahol előre meghatározott üzenetek közül kiválaszthatja a sikertelen kézbesítés okát, illetve megjegyzést is tud írni.

Egy visszajelzés után az alkalmazás megkeresi a GPS koordináták alapján a következő lakcímet, amire még nem érkezett visszajelzés, majd betölti a térképes nézetbe, így a terjesztő könnyen végig tud haladni a címeken, anélkül, hogy a nézetek között váltogatnia kellene.

Esetenként előfordulhat, hogy a terjesztő végig szeretné nézni a számára kiosztott feladatokat, illetve nem az alkalmazás által meghatározott sorrendben szeretné kézbesíteni az újságokat. Ezekben az esetekben tudja használni a lista nézetet, amely tartalmazza a terjesztő számára kiosztott feladatokat egy adott terjesztési cikluson belül. A lakcímek utcanév szerint és házszám szerint vannak rendezve. Lehetőség van szűrésre utcanév, illetve állapot szerint is (például csak a kézbesítetlen címek jelenjenek meg, az Arany János utcából). A lista nézet esetében minden lakcím előtt megjelenik egy ikon, ami jelzi, hogy történt-e már visszajelzés az adott címre vagy sem. Szintén minden cím után található egy másik ikon, amire ha rákattint a terjesztő, egy felugró ablak jelenik meg, amelynek célja, hogy az adott címhez hozzá lehessen rendelni az éppen aktuális GPS koordinátákat, így a későbbiekben pontosabb útbaigazítást kaphatnak a terjesztők.

A mobil kliensalkalmazás caching mechanizmussal rendelkezik, ami biztosítja, hogy internetkapcsolat ideiglenes hiányában is elmentésre kerüljenek a visszajelzések. Az adatok először egy lokális tárolóba mentődnek. Az alkalmazás meghatározott időközönként ellenőrzi, hogy történt-e változás és amennyiben történt elküldi a változtatásokat a szervernek. Mindkét nézet alján egy állapotsor található, amely az adatok állapotát jelzi (szinkronizált, nem szinkronizált). Kijelentkezéskor minden lokálisan tárolt adat törlődik, de a felhasználó ezt megelőzően figyelmeztetést kap, amennyiben még vannak nem szinkronizált adatok.

A felhasználók azonosítása egyedi azonosítók (token-ek) segítségével történik. A terjesztő a bejelentkezés után egy egyedi azonosítót kap a szervertől, ami meghatározott ideig érvényes marad. Az egyedi azonosító a lokális tárolóba mentődik, így az alkalmazás újbóli megnyitása esetén, ha még érvényes a szervertől kapott token, akkor automatikusan belépteti a terjesztőt. A token minden REST API hívás fejlécébe is belekerül, így a szerver minden kérés esetében tudja azonosítani a felhasználót.

Lehetőség van az alkalmazás konfigurálására is. A *Beállítások* menüpont alatt megadható a szerver címe, a kommunikációhoz használt port, illetve a REST API elérési útvonala. Hasonlóképpen beállítható a szerverrel való szinkronizáció gyakorisága is.

3. Megvalósítás

A következőkben a dolgozat a szoftver megvalósításának kiemelkedőbb részleteit és lépéseit ismerteti alrendszerenként.

3.1. Az ITNPD Backend

Az ITNPD szerver oldali implementációja a Java Enterprise Edition [2] platformra épül. A Java EE specifikáció-család összetett vállalati rendszerek fejlesztését támogatja. Fontosabb részei az Enterprise JavaBean (EJB), Java Servlet API, Java Server Pages (JSP), Java Message Service API (JMS), Java Persistence API (JPA), Java Transaction API (JTA) stb.

Az ITNPD projekt üzleti logika rétege az EJB technológiára [3] épül. Az EJB, a Java nyelven megírt vállalati alkalmazások esetében alkalmazható szerver-oldali komponensmodell. Az EJB technológiára épülő alkalmazásoknak számos előnye van: skálázhatóak, tranzakcionálisak és több felhasználó esetében is biztonságosak.

Az adathozzáférési réteg megvalósítása a Java Persistence Api [4] (JPA) specifikáción alapszik. A JPA egy standard ORM (Object-Relational Mapping) leképezést határoz meg és egy objektumorientált szemléletmódnak megfelelő lekérdező nyelvet (JPQL), valamint egy lekérdezések dinamikus létrehozására megoldást biztosító Criteria Query API-t is biztosít. A projekt fejlesztése során a JPA referencia implementációja, az EclipseLink [5] keretrendszer volt alkalmazva.

A Backend modul tartalmazza a rendszer entitásait, illetve az adathozzáférési és szolgáltatási rétegeket. Öt nagyobb részből épül fel: *model*, *repository*, *service*, *interceptor* és *util*.

3.1.2 Az adatmodell

A *model* csomag tartalmazza a megvalósítás során alkalmazott központi entitásokat. Az entitások adatbázisba történő leképezése a JPA szabványnak [4] megfelelően történik. A leképezés módja annotációs mechanizmus segítségével van meghatározva. Ugyanitt a BeanValidation szabványnak [6] megfelelő annotációkkal is el vannak látva egyes mezők és ezeknek a meta-adatoknak az alapján különböző keretrendszerek ellenőrizhetik az adatok helyességét (így például már a felhasználói felület szintjén is megtörténhet a validáció).

A szoftverrendszer központi entitásait reprezentáló osztályokat az 1. melléklet szemlélteti.

A modell osztályok közös jellemzője, hogy privát adattagokkal rendelkeznek, amelyek a publikus getter és setter metódusokon keresztül érhetőek el. Az entitások a BaseEntity osztály leszármazottjai, amely a Serializable interfészt implementáló AbstractModel-ből származik. A BaseEntity osztályból származik a MultitenantEntity, amely a multi-tenancy [7] megvalósításához szükséges.

Minden entitásnak egyértelműen beazonosíthatónak kell lennie, ezért az AbstractModel tartalmaz egy univerzális azonosítót (UUID). Ezen kívül minden lementett entitás rendelkezik egy azonosítóval, amely az adatbázisrendszer által generált elsődleges kulcsnak a megfelelője. A jelenleg használt entitások esetében ez az azonosító Long típusú és a BaseEntity osztályból öröklődik. Amennyiben speciális azonosítóval rendelkező entitás bevezetésére lenne szükség (pl. összetett kulcs esetén), az származhatna közvetlenül az AbstractModel alaposztályból.

Egy újság terjesztése több régióban történhet. Minden régióban különböző terjesztők lesznek és különböző megbízott személyek adminisztrálják a terjesztéssel kapcsolatos adatokat. Az egyes régiókon belüli felhasználók csak a saját régiójukhoz tartozó adatokat láthatják. Az ITNPD ezt a követelményt a multi-tenancy mechanizmus révén biztosítja. Minden adathoz hozzárendelődik egy tenant azonosító, amely a régió függvényében van meghatározva. Ezt az azonosítót a MultitenantEntity osztályból öröklik az entitások. Amennyiben a rendszerben szükségessé válik régióktól független adatok kezelése is, az ezeknek megfelelő entitások lehetnek a BaseEntity közvetlen leszármazottjai.

A projekten belüli egyik legfontosabb entitás a Distribution, amely egy szórás adatait tartalmazza. Egy szóráshoz tartozhat több feladat (Job), valamint ellenőrzési lista

(VerificationList). A Job entitás a feladatokkal kapcsolatos információt hordozza, többek között a terjesztőt (Distributor) és a megrendelőt (Customer). Mindegyik Jobhoz tartozhat egy visszajelzés (CustomerFeedback), ami egy telefonos ellenőrzést követően kaphat értéket. A Customer-hez tartozik egy lakcím (Address), amelyhez egy utca (Street). A Street egy helységhez (Settlement) tartozik és vannak koordinátái (Coordinates). A felhasználók két típusúak lehetnek: terjesztők (Distributor) és adminisztrátorok (Admin). Mivel közös tulajdonságokkal rendelkeznek, ezért mindkét entitás egy AbstractUser-ből származik, ami ezeket a közös adatokat foglalja magába.

3.1.2 Az adathozzáférési réteg

A *repository* csomag tartalmazza az adatbázis-hozzáférési műveleteket meghatározó interfészeket, valamint az ezeket megvalósító osztályokat. A repository bean-eknek nevezett osztályok egy BaseRepositoryBean osztályt terjesztenek ki, amely az alapvető adatbázis műveleteket tartalmazza. Ezeken felül mindegyik repository osztály további specifikus metódusokat vezethet be. A repository osztályok EJB [3] komponensek, állapotnélküli (stateless) session bean-ek formájában vannak implementálva és lokális interfészeken keresztül kommunikálnak a szolgáltatási réteg komponenseivel.

A lekérdezések nagy része JPQL query-k segítségével van implementálva. Azok a lekérdezések, amelyeknek a struktúráját csak futási időben lehet pontosan meghatározni (pl. összetett szűrési feltételek esetében) a JPA Criteria Query API-jának segítségével vannak dinamikusan létrehozva. Néhány elszigetelt esetben natív query-ket is alkalmaz a rendszer (a tenant azonosító beállítása előtt szükséges adatbázis műveletek esetében).

A multi-tenancy mechanizmus megvalósítása az EclipseLink keretrendszer [5] kapcsolódó szolgáltatásain alapszik. Minden repository komponenst érintő metódushívás interceptálva van egy speciális EJB Interceptor által, amelyik a bejelentkezett felhasználó függvényében beállítja a tenant azonosítót, így a hívó fél mindig csak a saját régiójához tartozó adatokat láthatja és kezelheti.

Amennyiben valamelyik adathozzáférési művelet sikertelen, a kivétel naplózásra kerül, és egy réteg-specifikus kivétel fog a felsőbb rétegek felé továbbítódni, egészen a felhasználói felület szintjéig, ahol így a felhasználó egy számára érthető hibaüzenetet fog látni. A kivételek kezelése a rendszer további részein belül is a leírt recept szerint történik. Az ITNPD fejlesztése során a naplózás a log4j és SLF4J technológiákkal valósult meg.

Az adatok exportálása és importálása, illetve az Excel állományok kezelése és ezek feldolgozása az Apache POI csomagot felhasználva valósult meg.

3.1.2. A szolgáltatási és üzleti logika réteg

A *service* csomagon belüli komponensek publikálják a külvilág számára a repository réteg funkcionalitásait. Ezen kívül ezek az üzleti logikáért felelős komponensek további bonyolultabb műveleteket is megvalósítanak. Ilyen például egy szórás indítása, ami több adatbázis műveletet igényel.

Az üzleti logikáért felelős komponensek szintén EJB-k, állapotnélküli session bean-ek formájában vannak megvalósítva. A REST erőforrás osztályokkal, illetve a webes komponensekkel lokális interfészeken keresztül kommunikálnak. A réteg megvalósításában fontos szerepet játszik a Dependency Injection tervezési minta, mivel a komponensek függőségeit (pl. az általuk használt repository vagy más service komponenseket) a keretrendszer menedzseli. A függőségek beinjektálása a keretrendszer által történik, a megadott EJB, vagy CDI [8] specifikációnak megfelelő annotációk alapján. A függőségek hasonló módon vannak menedzselve a REST erőforrások és webes komponensek esetében is.

A rendszer adminisztrátorainak a hitelesítése a JAAS (Java Authentication and Authorization Service) [9] specifikációnak megfelelően történik. A beléptetést a Glassfish végezi el egy meghatározott JDBC security realm alapján. A Glassfish realm egy biztonsági tartomány, ahol konfigurálva vannak a hitelesítéshez szükséges adatok (adatbázis tábla és oszlop, ami a felhasználónevet, jelszót, felhasználói csoportokat tartalmazza, alkalmazott jelszó-titkosítási algoritmus stb.). A jogosultságok ellenőrzése a keretrendszer által automatikusan megvalósítható, a szolgáltatási réteg komponensein belüli metódusokra helyezett annotációk segítségével. Ezen kívül az EJB példányokhoz rendelt SessionContext objektumokból kinyerhető a hívó fél (bejelentkezett felhasználó) személyazonossága, így a programon belül ennek alapján további ellenőrzések és beállítások végezhetőek el (pl. az EJB-khez rendelt interceptorokon belül).

A *util* csomag a jelszó titkosításáért felelős osztályt, egy PropertyProvider osztályt, illetve a StreetCoordsUtility osztályt tartalmazza. A PropertyProvider a konfigurációs paramétereket szolgáltatja (egy konfigurációs állomány segítségével, megadott kulcsok alapján), míg a StreetCoordsUtility osztály az előfizetők importálását követően elindít a háttérben egy végrehajtási szálat, ami a Google GeoCoding szolgáltatásait igénybe véve lekérdezi azoknak az utcáknak a koordinátáit, amelyeknek esetében ez az információ még hiányzik.

Az *interceptor* csomag tartalmazza a multi-tenancy megvalósításához szükséges interceptorokat. Az interceptorok lekérdezik a bejelentkezett felhasználó nevét, majd ennek alapján beállítják a tenantId-t, ami az ITNPD esetében a régió azonosítója.

3.2 Az ITNPD API

Ez a modul felelős a REST alapú webszolgáltatások megvalósításáért, melyet a JAX-RS (Java API for RESTful Services) API [1] specifikációt megvalósító Jersey keretrendszer [10] segítségével valósít meg. A modul két csomagot foglal magába: egy *assembler*, illetve egy *resource* csomagot. Az *assembler* csomag az entitásoknak megfelelő assembler osztályokat tartalmazza, amely az entitások és DTO-k (Data Transfer Object) közötti átalakításért felelős. A resource csomag tartalmazza azokat az osztályokat, amelyek fogadják a REST kéréseket, illetve elküldik a válaszüzenetet.

Három erőforrás osztály került megvalósításra: a CustomerResource, a DistributorResource, valamint a JobResource. A CustomerResource szolgálja ki az előfizetővel kapcsolatos REST kéréseket, a DistributorResource a terjesztő azonosításával kapcsolatos kéréseket: bejelentkeztetés felhasználónévvel és jelszóval, bejelentkeztetés token alapján és kijelentkeztetés. A JobResource a feladatok megjelenítéséhez és kezeléséhez szolgáltatja az adatokat, mint például a bejelentkezett terjesztőhöz tartozó feladatokat, a legközelebbi feladatot stb.

3.3 ITNPD: Android

Az Android egy Linux kernelen alapuló mobil operációs rendszer. Első sorban érintőképernyős mobil eszközökre tervezték, okostelefonokra, táblagépekre, napjainkba azonban már léteznek televíziók, autók, órák Android operációs rendszerrel. Az alkalmazások elsősorban Java-ban íródnak, az Android Software Development Kit (SDK) használatával. Az SDK tartalmazza a fejlesztéshez szükséges eszközöket, mint például a debugger-t, emulátort, dokumentációt, példa kódokat stb.

Az ITNPD mobil kliensalkalmazás megvalósításának esetében fontos feladat volt a kommunikáció és az adatok lokális tárolásának a megvalósítása. A REST kérések küldéséhez és fogadásához, valamint az adatok JSON formátumba és JSON formátumból való átalakításához a

Retrofit library van felhasználva. Az adatok lokális tárolóban való rögzítését az OrmLite perzisztencia keretrendszer biztosítja. Térképszolgáltatásként a Gogle Maps-et használja az alkalmazás.

3.3.2 A kommunikáció megvalósítása

Az alkalmazás kivitelezésénél fontos szempont volt a kliens és szerver alkalmazás közötti kommunikáció helyes működése. Előnyösnek tűnt REST szolgáltatás segítségével megvalósítani az adatcserét, ahol a kérések egy-egy HTTP utasításra vannak leképezve. Az Android alkalmazás kivitelezésekor több módszer is ki volt próbálva a REST kérések küldésére és fogadására. Az első lehetőség RestService osztályok írása volt, amelyek elküldik a HTTP kéréseket, illetve fogadják a HTTP válaszokat. Eleinte ez tűnt a legegyszerűbb és legkézenfekvőbb megoldásnak, viszont a későbbiekben egyre inkább rontotta a projekt átláthatóságát, valamint lassabb is volt, mint a jelenleg alkalmazott Retrofit library. Android alkalmazások esetében nem megengedett, hogy bármilyen HTTP protokollon keresztül történő kommunikáció a fő programszálra kerüljön. Ennek következtében a RestService osztályokon belül a kérések elküldéséhez AsyncTask-ot kellett létrehozni, melynek működése sokban hasonlít a web programozásból jól ismert AJAX technológiának a működési elvéhez. A módszer működött, viszont az AsyncTask-ok miatt a RestService osztályok olvashatósága sokat romlott, emellett még nem tartalmazta a JSON kódba/kódból való átalakítást. Az Android támogatja a JSON formátum átalakítását az Android.util.JsonReader csomagnak köszönhetően, így meg is valósult a kommunikáció, viszont tovább rontotta a programkód olvashatóságát és sokban bonyolította az Android alkalmazás szerkezetét, mivel minden entitás esetében implementálni kellett az objektum és JSON formátum közötti átalakítást. Mivel a mobil kliens is több modell osztályt tartalmaz, így sok hasonló funkcionalitással rendelkező osztály jött létre. Értelmetlennek tűnt ezen az úton továbbhaladni, így más megoldást keresett a csapat.

A Retrofit library-t a fentebb említettekhez hasonló feladatok megoldására tervezték. Előnye az előbbiekben említett megközelítéshez képest, hogy használata egyszerű, nem eredményez sok új programkódot, gyorsan végzi az objektumok és JSON formátum közötti átalakítást, és nem kell minden entitás esetében implementálni az átalakítást, mivel a választ a meghatározott objektum típusának megfelelően kapjuk vissza.

A Retrofit library egy Java interfészbe fogja össze a REST hívásokat. Egy példa az ItnpdService interfész, amely egy lekérdezést tartalmaz:

```
public interface ItnpdService {

    @GET("/job/streets/{token}")

    void listStreets(@Path("token") String token, Callback <List<StreetDto>> cb);

}
```

A fenti kódrészletből is látható, hogy átláthatóbb a kód egy AsyncTask-hoz képest. A példából kiderül, hogy a lekérdezést GET kérés formájában küldi el a /job/streets/{token} útvonalra, ami egy token paramétert is tartalmaz, amely a bejelentkezett terjesztőt azonosítja. A listStreets metódus deklarálásánál kijelenti, hogy a bemenő token paramétert String-ként adja meg, míg a Callback metódus egy StreetDto-ból álló listával fog visszatérni. Ahhoz, hogy meg lehessen hívni a listStreet metódust létre kell hozni egy RestAdaptert a következőképpen:

```
RestAdapter restAdapter = new RestAdapter.Builder().setEndpoint("https://server.ro") .build();

ItnpdService service = restAdapter.create(ItnpdService.class);
```

A RestAdapter inicializálása két lépésben történik. Az első lépésben meg kell adni a csatlakozási pontot, majd a második lépésben meg kell határozni, hogy a RestAdapter-nek melyik interfésznek kell megfelelnie. A példányosítás végén létrejön egy adapter, amely ebben az esetben az ItnpdService interfész implementációja. Az adapter segítségével meg lehet hívni a listStreet metódust, ami egy callback hívást eredményez. A callback hívás a megadott típusnak megfelelő objektummal/objektum listával tér vissza, nem kell külön foglalkozni a JSON formátumból való átalakítással.

3.3.3 Az adatok tárolásának módja

A kliensalkalmazáson belül kétféleképpen történik az adatok tárolása. Az alkalmazás beállításai a SharedPreferences-be lesznek lementve, míg a terjesztő számára kiosztott feladatok, címek mentése az OrmLite keretrendszer segítségével történik.

Az OrmLite (Lightweight Object Relational Mapping Java Package) egy pehelysúlyú perzisztencia keretrendszer, amely Java objektumok adatbázisba való leképezését teszi lehetővé. Használata sokban hasonlít más perzisztencia keretrendszerek használatához. Például, a leképezés módját meg tudjuk adni a Java osztályok adattagjainak a felannotálásával. Használata előnyösnek bizonyult, mivel támogatja a tranzakció kezelést, SQL lekérdezéseket lehet írni az adatok visszatérítésének érdekében, így gyorsabban visszakaphatóak az adatok, összehasonlítva a

SharedPreferences-ből vagy állományból történő adatkinyeréssel. Másik előnye, hogy kevés erőforrás felhasználása mellett is nagyon jól működik. Az OrmLite képes kötegelt műveletek elvégzésére is, ami a fejlesztés során szintén hasznosnak bizonyult, például az előfizetők adatbázisba való mentésekor, mivel minden előfizetőhöz tartozik egy lakcím, minden lakcímhez tartozik egy GPS koordináta és így tovább. Ez esetben a különböző entitásokat sorban le kellene menteni, és tartalmazás szerint referenciát adni róla a másik rá hivatkozó objektumnak, majd azt szintén lementeni. Ilyen formában mindig új tranzakció indulna el, ami nagyban növelné az adatok mentési idejét. Gyorsabb megoldás, ha egy objektum a hozzá tartozó összes más objektummal együtt kerül lementésre, így kevesebb tranzakció kezelésre van szükség, tehát a mentési fázis gyorsabb. Ugyanez érvényes a lekérdezések esetében is.

A Retrofit library-nek és az OrmLite-nak köszönhetően a fejlesztés gyorsabb, egyszerűbb és kényelmesebb lett, a végeredmény pedig egy biztonságosabban, stabilabban működő alkalmazás.

3.3.4 Nézetek közötti kommunikáció

A fejlesztés során kihívást jelentett a nézetek közti kommunikáció, illetve a nézetek közti váltásnak a megvalósítása. A mobil kliensalkalmazás két főbb nézetből áll: egy térképes, és egy lista nézetből. Fontos szempont, hogy ez a két nézet együttműködjön, és erre a problémára több megoldás közül lehet választani.

Az első lehetőség az, hogy a két nézetet külön-külön Activity-ként hozzuk létre. A módszer előnye, hogy így mind a két nézet külön Context-el rendelkezik. Hátránya, hogy meg kell oldani a két Activity közötti kommunikációt, illetve esetünkben a két nézet redundáns információt is tartalmazott volna, mivel mind a két nézetnek rendelkeznie kellett volna a terjesztő számára kiosztott feladatlistával.

A második lehetőség, ha a két nézet külön-külön fragment-ként van kezelve. Ebben az esetben egyik nézet sem rendelkezik saját kontextussal és szükség van egy szülő Activity-re is, viszont így mind a két nézet hozzáférhet a szülő Activity adattagjaihoz. A jelenlegi megvalósítás alapját ez utóbbi ötlet képezi. Az implementáció esetében a szülő Activity tartalmaz egy JobManager példányt, amely a terjesztő számára kiosztott feladatlista kezeléséért felelős. Ehhez az osztályhoz mind a két nézet hozzáfér a szülő Activity getJobManager() metódusa révén, így nem tárolnak redundás adatokat. Ahhoz, hogy ez működőképes legyen, a JobManager osztálynak

singleton-nak kell lennie, melyet a szülő Activity indításakor hoz létre a rendszer. Kijelentkezéskor a JobManager egyetlen létező példánya megsemmisül, majd a következő bejelentkezéskor újra létrejön.

3.3.5 Az adatok szinkronizálása

Az Igen, tessék! újságterjesztést elősegítő szoftverrendszer esetében fontos követelmény volt, hogy a kliens és szerver alkalmazások között megfelelően működjön az adatok szinkronizálása. Ennek érdekében a mobil kliensalkalmazás elindításakor két háttérszolgáltatást indít el. Az egyik a szerverről való friss adatok letöltéséért felelős, a másik a kliens oldalon módosult adatok feltöltéséért. Mindkét háttérszolgáltatás az IntentService osztály leszármazottja. Az IntentService előnyei közé tartozik, hogy azonnal végrehajtó szálra (worker thread) kerül, és miután elvégzi a számára kiosztott feladatot megsemmisül.

A módosult adatok feltöltése a következőképpen történik: meghatározott időközönként az erre a feladatra indított háttérszolgáltatás bejárja a terjesztő számára kiosztott feladatlistát és amennyiben talál olyan feladatot, amely nem szinkronizáltként van megjelölve, elküldi ezt a szervernek. Hasonlóképpen történik az adatok automatikus frissítése is. A felületen lehetőség van az adatok frissítésének kézzel történő indítására, valamit a szinkronizálási időközök beállítására is.

3.4 ITNPD: Web és Widgetset

A rendszer adminisztrációs webes felülete Vaadin [11] technológiára épül, így lehetőség volt a Java nyelv alkalmazására a webes fejlesztés során. Kivételt képeznek a bejelentkező és a hibakezelő oldalak, amelyek JSP (Java Server Pages) technológia segítségével vannak megvalósítva.

A Vaadin egy nyílt forráskódú webalkalmazás-keretrendszer, amely lehetőséget biztosít web alkalmazások gyors és hatékony fejlesztésére. Szerver oldalon Java Servlet technológiát használ, a megjelenítés pedig HTML és JavaScript alapú. A felület a Java desktop alkalmazásokhoz hasonlóan Java komponensekből építhető fel (szerver oldali komponensek), a keretrendszer Google Web Toolkit-et [12] használ, amely ezek alapján JavaScript kódot generál. A kommunikáció AJAX alapú, az adatok JSON formátumban továbbítódnak. A biztonsági ellenőrzések és a munkamenet kezelése automatikusan biztosítva vannak.

Az ITNPD esetében a különböző diagramok megjelenítése dCharts Vaadin addon alkalmazása által valósult meg. Ezt a kiegészítő csomagot tartalmazza a Widgetset modul.

Akárcsak a szolgáltatási réteg esetében, itt is a Context and Dependency Injection minta volt követve a Vaadin CDI addon alkalmazása által. Ennek alapján egy Vaadin CDI által menedzselt nézetet a @CDIView("viewTitle"), a fő felhasználói felületet a *@CDIUI("uiPath")* annotációval kell ellátni.

A felülethez való hozzáférés kizárólag adminisztrátor jogokkal rendelkező felhasználók számára lehetséges. A felhasználók bejelentkeztetését és a jogosultságok ellenőrzését a Glassfish végzi el, a JAAS (Java Authentication and Authorization Service) specifikációnak megfelelően, egy JDBC realm-ben konfigurált adatok alapján.

4 Felhasznált módszerek és eszközök

Az Igen, tessek! Newspaper Distribution szoftverrendszer fejlesztése Scrum módszertan alapján történt, a folytonos integrációs (Continuous Integration, CI) elveknek megfelelően. A fejlesztés során számos modern, széles körben alkalmazott fejlesztői eszköz használatára sor került. Az alkalmazott eszközök közül kiemelt szerepet töltenek be a build eszközök, a verziókövetés során alkalmazott eszközök, valamint a kódminőség biztosítására alkalmazott kódelemző eszközök.

A projekt fejlesztése során a verziókövetés a Mercurial [13] osztott rendszer által volt biztosítva. A Mercurial, Python és Portable C nyelvekben megírt nyílt forráskódú rendszer. A fejlesztés során, a hatékonyság növelésének érdekében, a TortoiseHg [14] grafikus felülettel rendelkező asztali kliensalkalmazást használta a csapat. A központi tároló menedzsmentje RhodeCode rendszer által volt biztosítva. A projekthez tartozó dokumentációk és egyéb információk megosztása egy XWiki rendszeren belül történt.

A Java alapú projektek esetében a legelterjedtebb build és függőségmenedzsment eszközök az Apache Ant [15] és Apache Ivy, az Apache Maven [16], illetve a Gradle. Az ITNPD szoftverrendszer esetében két build eszköz volt alkalmazva: az Apache Maven, illetve az Android fejlesztők körében népszerű Gradle [17]. A fordítási folyamathoz elegendő lenne a felsorolt build eszközök egyike, viszont ebben az esetben a projekt konfigurációja körülményesebb lett volna. A fejlesztőcsoport nem akarta a projektet két külön részre bontani, ezért, a build folyamathoz maven-gradle plugin-t használtak, így a Maven fordításkor elindítja az Android alkalmazás fordítását is. A Maven a rendelkezésre álló tapasztalat, dokumentáció és a Continuous Integration rendszerekkel történő egyszerűbb integrálhatóság szempontjából tűnt megfelelőbb választásnak a Java modulok esetében. A projekt külső függőségeinek menedzsmentjében egy Artifactory repository management rendszer segített. Folytonos integrációt támogató rendszerként a Jenkins szolgált.

Jelenleg a szoftverfejlesztők számos minőségmenedzsment eszköz közül válogathatnak, melyek feladata a kód statikus, vagy dinamikus elemzése, valamint az esetleges hibák megjelenítése. Ilyen ezközök a Squale, MetrixWare, Black Duck Suite, ConQAT, SonarQube stb. Az ITNPD projekt fejlesztése során a kódminőség biztosítására a SonarQube [18] minőségmenedzsment platform volt felhasználva. A rendszer a 2.0 verziójától kezdődően a

minőségelemzés hét aspektusát fedi le, melyek a következők: duplikátumok kiszűrése, kódolási standardok betartása, kód tesztekkel való lefedettsége, kód komplexitása, potenciális hibák, megfelelő dokumentáltság, design és architektúra.

A Java modulok esetében alkalmazott NetBeans fejlesztői környezet mellett az Android platformra történő fejlesztés Andriod Studio [19] környezettel történt.

Az alkalmazásszerver szerepét a Java EE referencia implementációja, a GlassFish [20] tölti be. Az adatbázis kezelése MySql adatbázis-menedzsment rendszerrel történik.

Következtetések és továbbfejlesztési lehetőségek

A projekt keretein belül sikerült létrehozni egy szoftverrendszert, amely egyrészt az újságterjesztőknek nyújt segítséget, mivel egyszerűen menedzselhetővé teszi a feladataikat és a kézbesítéseknél segít a tájékozódásban. Másrészt a terjesztést menedzselő szervezet számára nyújt nyomon követési és ellenőrzési lehetőségeket, a terjesztés minél eredményesebb lebonyolítása érdekében. Ugyanakkor az Excel alapú adatrögzítést és adatkezelést felváltja egy adatbázis, melyhez csak a jogosultságoknak megfelelően férhetnek hozzá a különböző felhasználók. Amennyiben Excel állományban is szeretnék az adatokat rögzíteni, lehetőség van az adatok exportálására is.

A szoftverrendszer további funkcionalitásokkal is kiegészíthető. Például, a webes felület az alábbi funkciókkal bővíthető:

- az előfizetők megjelenítése térképes nézetben a lakcímek alapján klaszterezve, esetleg hőtérképen való megjelenítés;
- az előfizetők adatainak az ellenőrzése, pl. a valószínűsíthetően hibás címek kiküszöbölése;
- az „egy utca egy terjesztőhöz” megszorítás megszüntetése;
- különböző algoritmusok implementálása az ellenőrző listák generálására.

A mobil alkalmazás az alábbi funkcionalitásokkal egészíthető ki:

- egy terjesztés indításakor a terjesztők kapjanak értesítést arról, hogy új feladatcsomagot kaptak;
- útvonaljavaslat készítése;
- összesített jelentések megtekintése a mobilalkalmazáson belül: pl. előző ciklusokon belüli teljesítmény, aktuális ciklus haladása stb.

Jelenleg a szoftverrendszer adott régiókhoz tartozó adminisztrátoroknak nyújt funkcionalitásokat. Egy további fejlesztési feladat egy központi adminisztrációs felület készítése, ahol lehetőség nyílik a régiók, illetve a régióhoz tartozó adminisztrátorok adatainak kezelésére, valamint központosított statisztikák megtekintésére.

Hivatkozások

[1] JAX-RS Hivatalos Weboldal, https://jax-rs-spec.java.net/, utolsó megtekintés dátuma:

2015.04.24

[2] Aron Grupta, *Java EE 7 Essentials, First Edition,* O'Reilly Media, 2013.

[3] Andrew Lee Rubinger, Bill Burke, *Enterprise JavaBeans 3.1, 6th Edition.*, O'Reilly Media, 2010.

[4] Java Persistence API Hivatalos Weboldal,

http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html,

utolsó megtekintés dátuma: 2015.04.24

[5] EclipseLink 2.5 Hivatalos dokumentációja

http://www.eclipse.org/eclipselink/documentation/2.5/eclipselink_otlcg.pdf,

letöltés dátuma: 2015.04.07

[6] JSR 303: Bean Validation Specification, version 1.0 final, http://download.oracle.com/otn-pub/jcp/bean_validation-1.0-fr-oth-JSpec/bean_validation-1_0-final-spec.pdf?AuthParam=1429970020_eef67b7376527ab1d040000685372a93, letöltés dátuma: 2015.04.24

[7] Introduction to the Multitenant Architecture,

https://docs.oracle.com/database/121/CNCPT/cdbovrvw.htm#CNCPT89242,

utolsó megtekintés dátuma:2015.04.24

[8] JSR-299: Contexts and Dependency Injection for the Java EE platform,

http://download.oracle.com/otn-pub/jcp/web_beans-1.0-fr-eval-oth-JSpec/web_beans-1_0-fr-eval-spec.pdf?AuthParam=1429971305_e600fbf327282e78f0ec9120eebd4901, letöltés dátuma: 2015.04.24

[9] JAAS Reference Guide,

http://docs.oracle.com/javase/7/docs/technotes/guides/security/jaas/JAASRefGuide.html, utolsó megtekintés dátuma:2015.04.24

[10] Jersey Hivatalos Weboldal, https://jersey.java.net/, utolsó megtekintés dátuma: 2015.04.24

[11] Marko Grönroos**,** *The Book of Vaadin: Vaadin 7 Edition - 4th Revision,*Vaadin, Ltd., 2014.

[12] Google Web Toolkit Hivatalos Weboldal, http://www.gwtproject.org/, utolsó megtekintés dátuma: 2015.04.19

[13] Mercurial Hivatalos Weboldal, http://mercurial.selenic.com, utolsó megtekintés dátuma: 2015.04.13

[14] TortoiseHg Hivatalos Weboldal, http://tortoisehg.bitbucket.org, utolsó megtekintés dátuma: 2015.04.13

[15] Apache ANT Hivatalos Weboldal, http://ant.apache.org, utolsó megtekintés dátuma:

2015.04.18

[16] Apache Maven Hivatalos Weboldal, http://maven.apache.org, utolsó megtekintés dátuma: 2015.04.18

[17] Gradle Hivatalos Weboldal, http://gradle.org, utolsó megtekintés dátuma: 2015.04.18

[18] Sonar Qube Hivatalos Weboldal, http://www.sonarqube.org, utolsó megtekintés dátuma: 2015.04.19

[19] Android Hivatalos Fejlesztői Weboldal, https://developer.android.com, utolsó megtekintés dátuma: 2015.04.19

[20] GlassFish Hivatalos Weboldal, https://glassfish.java.net, utolsó megtekintés dátuma: 2015.04.19

Mellékletek

1.Melléklet: Az ITNPD központi entitásai