XVII. reál- és humántudományi Erdélyi Tudományos Diákköri Konferencia (ETDK)
Kolozsvár, 2014. május 15–18.

Szolgáltatás alapú szoftverrendszer raktárkészletek optimalizálására

Szerzők:
Győri Réka
Babeș-Bolyai Tudományegyetem, Kolozsvár, Matematika és Informatika Kar,
Informatika szak, III. év
Imecs Tamás
Babeș-Bolyai Tudományegyetem, Kolozsvár, Matematika és Informatika Kar,
Informatika szak, III. év
Török Enikő
Babeș-Bolyai Tudományegyetem, Kolozsvár, Matematika és Informatika Kar,
Informatika szak, III. év

Témavezető:
dr. Simon Károly egyetemi adjunktus,
Babeș-Bolyai Tudományegyetem, Kolozsvár,
Matematika és Informatika Kar,
Magyar Matematika és Informatika Intézet

Kivonat

A dolgozat az OptInv szoftverrendszert mutatja be, amely vállalatok számára nyújt segítséget raktárkészletük nyomon követésében és optimalizálásában.

A kis- és középvállalkozások általában nem engedhetik meg maguknak készlet-menedzsment modulokat is biztosító drága ERP rendszerek használatát. A vállalkozók többsége csak korábbi tapasztalatok alapján menedzseli a készletet, és előfordulhat, hogy adott pillanatban nagy pénzösszegek állnak haszontalanul a raktáron tartott termékekben, vagy éppen hiány miatt nem teljesíthető egy rendelés. Megfelelő módszerekkel és eszközökkel ezek a problémák kiküszöbölhetőek.

Az OptInv projekt célja egy SaaS megoldást nyújtani erre a feladatra, amelynek segítségével különböző könyvelőprogramokból beimportált adatok alapján megtekinthetőek termékekre, beszállítókra, eladásokra vonatkozó statisztikák, kategorizálhatóak a termékek, felmérhető a raktáron lévő készlet értéke, előre jelezhető a fogyasztás és ilyen módon optimalizálható a készlet.

Tartalomjegyzék

Bevezető

A szakdolgozat az OptInv projektet mutatja be, amely kereskedelemmel foglalkozó vállalatok számára nyújt segítséget raktárkészletük optimalizálásában. Nyilvántartja a raktáron lévő készletet, különböző statisztikai jelentéseket készít és az információkat egy webes felületen jeleníti meg.

A legtöbb nagyobb vállalkozás nemcsak könyvelő programok segítségével tartja nyilván az eladásokkal, termékekkel, beszállítókkal kapcsolatos adatokat, hanem összetett ERP szoftverrendszereket használ, amelyek segítenek az árusítás gördülékeny szervezésében, valamint a raktárak és készletek kezelésében. Hatékony készletoptimalizálási modulokat azonban nem minden rendszer biztosít. Kis- és középvállalkozások esetében nincs keret ilyen drága rendszerekre és gyakran a komolyabb vállalkozói tudás is hiányzik. A vállalkozók többsége csak a korábbi tapasztalatok alapján menedzseli a raktárkészleteket és rendeléseket. Sok esetben nem veszik észre, hogy, bár vállalatuk adott pillanatban nem tűnik veszteségesnek, nagyon nagy pénzösszeg áll haszontalanul a raktáron lévő készletben. Ugyanígy kényelmetlen helyzetekhez vezethet az is, ha egy adott rendelés nem teljesíthető, mert valamelyik termékből éppen kifogyott a készlet.

Megfelelő módszerekkel és eszközökkel meg lehetne oldani, hogy mindig a megfelelő időben és szükséges mennyiségben legyen rendelés egy adott termékből. Ilyen módon spórolni lehetne a beszerzés és raktározás költségein, pénzt és helyet lehetne megtakarítani és hasznosabban felhasználni.

Az OptInv szoftverrendszer segítségével a különböző könyvelőprogramokból (pl. HamorSoft termékcsalád) átvett adatok alapján a felhasználó megtekinthet a készletre vonatkozó statisztikai adatokat.

A felhasználó egy böngészőt elindítva, a megfelelő weboldalon keresztül történő bejelentkezés után érheti el a program által nyújtott szolgáltatásokat. Lehetőség van az adatok (termékek, eladások, beszállítók stb.) különböző könyvelési programok adatbázisából történő importálására. A beimportált adatok alapján a rendszer kimutatásokat készít. Például, az eladásokkal kapcsolatos információkat figyelembe véve a termékek adatai mellett megjelenik az is, hogy az adott termékből mekkora volt az átlageladás egy tetszőleges időintervallumon belül. Ismerve az eladási statisztikákat és tudva azt, hogy egy adott termékből mekkora mennyiség van adott pillanatban raktáron, a felhasználó azt is megtekintheti, hogy várhatóan hány napig elegendő még az adott termék. Ezáltal a felhasználó könnyebben el tudja dönteni,

hogy melyik az a termék, amelyből nagyon sok van raktáron, tehát nem szükséges rendelni, sőt esetleg árleszállítást érdemes alkalmazni a raktár felszabadításának és a költségek csökkentésének érdekében. Az is látható, hogy milyen termékekből szükséges rendelni, hogy nagy kereslet és gyors fogyás esetén is teljesíteni lehessen az ügyfelek kéréseit.

A program segítségével elkülöníthetőek a rekurrens[1], sporadikus[2] és szezonális[3] termékek, ami fontos, mivel ezeket különböző módon kell kezelni, kategóriánként különböző előrejelzési módszereket kell használni. Például, téli gumiból nyáron valószínűleg nincs, vagy nagyon kevés van raktáron. Ha a napi fogyást az elmúlt évre számoljuk, akkor ez nem lesz nulla, mivel ősszel és télen volt eladás ebből a termékből. Így, az évi eladások alapján a program azt javasolhatja, hogy rendelni kell belőle. Másrészt, könnyen belátható, hogy nem valószínű nyáron a téli gumi vásárlása. Az OptInv az eladási trendeket elemezve képes helyes javaslatot szolgáltatni az ilyen helyzetekben is.

A projekt fejlesztése szakmai gyakorlatként indult. Tulajdonképpen egy tanulmányprojekt a Codespring Kft., a Babeș-Bolyai Tudományegyetem, valamint a Kolozsvári Műszaki Egyetem közös pályázatához. Az alapötlet Pálhegyi Zoltántól, az AVE Hargita ügyvezető igazgatójától származik, aki egy tanácsadási szolgáltatást szeretne ráépíteni. A fejlesztésben a Codespring részéről segítséget nyújtott Dr. Simon Károly, a projekt szakmai irányítója, valamint Tompa Lóránd projektmenedzser, aki terméktulajdonosi szerepben vett részt a fejlesztés első fázisában. Az alkalmazás fejlesztéséhez és teszteléséhez használt valós adatbázist a székelyudvarhelyi Galfi Servco autóalkatrész kereskedés biztosította, amelynek megbízottjaként Molnár Róbert marketing igazgató folyamatos segítséget nyújtott a tesztelésben. A prototípus bemutatását követően, a HamorSoft vezetőségével való egyeztetés után körvonalazódott egy együttműködési program, amelynek keretein belül a HamorSoft csapata OptInv modellnek megfelelő adatkimentési (export) modullal egészítheti ki termékeit, az adatimportálás hatékonyabbá tételének érdekében.

[1] Azokat a termékeket nevezzük **rekurrensnek**, amelyek esetében folyamatos és nagyjából egyenletes a fogyasztás.

[2] Azokat a termékeket, amelyekre nem jellemző a rekurrens tulajdonság, például előfordulnak periódusok, amelyekben nincs eladás, **sporadikus**aknak nevezzük.

[3] Ha egy termék esetében az év bizonyos szakaszaiban rekurrens fogyasztás figyelhető meg, viszont a többi periódusban egyáltalán nincs eladás, ezt a terméket a **szezonális** kategóriába sorolhatjuk.

1. Felhasznált eszközök és fejlesztési módszerek

Az OptInv fejlesztése *Scrum* módszertan [1] alapján történt. A *Scrum* a szoftverfejlesztés egy inkrementális, iteratív módszere, mely az *Agile* metodológia egyik legismertebb változata. Rövid iterációkra (sprint) épül, melyeket egy-egy tervezési fázis előz meg (sprint planning). Mivel a módszer nagyon fontos részét képezi a kommunikáció, rendszeresek a projekttel kapcsolatos rövid megbeszélések (daily meeting). Minden iteráció végén van egy bemutató, amikor a fejlesztőcsapat működő rendszeren mutatja be a sprint alatt fejlesztett funkcionalitásokat. Ezeket a terméktulajdonos (product owner) előre rögzített elfogadási kritériumoknak megfelelően elfogadja, vagy elutasítja. A sprint végén általában van egy visszatekintő elemzés (retrospective meeting) is, amikor a fejlesztőcsapat megpróbálja levonni az elmúlt iteráció tanulságait.

Egy nagyobb projekt esetében, amelynek fejlesztésében egyszerre több programozó is részt vesz, verziókövető rendszerek segítségével lehet a változásokat nyomon követni, a különböző verziókat, fejlesztési ágakat kezelni. A *Mercurial* [2] egy osztott verziókövető rendszer, ahol a fejlesztők a központi tárolón (repository) kívül a saját gépükön egy lokális tárolóval is rendelkeznek. A lokális tárolóba történő gyakoribb *commit* műveletek lehetősége, az összefésülési (*merge*) műveletek hatékonyabb támogatása, a több lokális tárolónak köszönhető biztonság, a folytonos internetkapcsolat szükségtelensége azok az előnyei a Mercurial-nak a hagyományosabb központosított verziókövetőkkel szemben, amelyek miatt az OptInv csapata ezt az eszközt választotta a fejlesztéshez. A csapatnak a fejlesztésben asztali kliensalkalmazásként a *TortoiseHg* nyújtott segítséget, a központi tároló menedzsmentjére a *RhodeCode* rendszer szolgált.

A *Redmine* projektmenedzsment és hibakövető (bug tracking) rendszer főként a feladatok bejegyzésében, nyomon követésében nyújtott segítséget. A fejlesztők közötti dokumentum-megosztásra, valamint a különböző specifikációk tárhelyéül az *XWiki* szolgált. Ide kerültek fel a különböző segédanyagok is, amelyek előresegítették a projekt fejlődését.

Egy összetett, több modulból álló projekt esetében nagyon fontos szerepet játszanak a build és függőségmenedzsment eszközök. Az OptInv projekt a *Maven*-t [3] használja, amely megoldja a függőségként bejelentett külső csomagok letöltését, a különböző modulok fordítását, automatizált tesztek futtatását, az alkalmazás kitelepítését alkalmazásszerverre stb.

Saját belső repository-t és kapcsolódó menedzsment rendszert is használt a csapat, a JFrog által fejlesztett nyílt forráskódú *Artifactory*-t.

Egy projekt minőségének biztosítását nagyban elősegítik a különböző kódelemző szoftverek, mint például az OptInv esetében használt *SonarQube*. A szoftver egy webes felületet biztosít a fejlesztőknek, amelyen különböző jelentéseket tekinthetnek meg a kódról. Ellenőrzi a kód megfelelőségét egy meghatározott kritériumrendszer szempontjából, javítási javaslatokat tesz, kimutatást készít a tesztlefedettségről, duplikátumokról stb.

A folytonos integráció [4] módszere olyan alapelveken alapszik, amelyek a több fejlesztő által fejlesztett kódok összefésülésének hatékonyságát kívánják növelni. Fontos szempont, hogy a fejlesztés során a rendszer lehetőleg mindig helyes, felépíthető és futtatható állapotban maradjon. Jó recept lehet, ha a fejlesztők többször (min. naponta) továbbítják változtatásaikat a központi tárolóba, minden változtatás után (vagy legalább bizonyos időközönként) végbemegy a build folyamat (beleértve az automatikus tesztek lefuttatását is) és mindenki hozzáfér a rendszer aktuális állapotát tükröző jelentésekhez. A folytonos integráció módszerét támogató szoftverek build szervereket biztosítanak, amelyek képesek ezeket a műveleteket automatizálni. Ilyen rendszer az OptInv esetében használt *Jenkins* is. A verziókövető rendszerrel összekapcsolva képes automatikusan felépíteni Maven projekteket és a build folyamat eredményeit egy webes felületen elérhetővé teszi. Összekapcsolható az automatikus kódelemző eszközökkel, így minden build után lefutnak az elemzések. Ezen kívül automatikusan kitelepíti a friss verziókat tesztszerverekre.

Az OptInv esetében nagyon hasznos eszköznek bizonyult továbbá a ZeroTurnaround által fejlesztett *JRebel*, amely a módosított Java osztályokat azonnal frissíti a JVM-en belül, így például nem szükségesek folytonos újra-kitelepítések a fejlesztés során. Ezzel, és az ehhez hasonló hot deploy eszközökkel nagyon sok időt lehet megtakarítani, mivel egy *deploy* művelet egy nagyobb rendszer esetében sok időt vehet igénybe.

Alkalmazásszerverként a Java Enterprise Edition specifikáció referencia implementációjának számító *Glassfish*-t használta a csapat. A külső adatbázisokból beimportált adatok, a belső modellnek megfelelően, egy MySQL adatbázisban vannak tárolva. Fejlesztői környezetként a NetBeans szolgált, különböző kiegészítő eszközökkel együtt.

2. Felhasznált technológiák

Az OptInv Java Enterprise Edition (Java EE) platformra épül, kihasználva a platform szolgáltatásait (biztonság, tranzakció kezelés, interceptorok stb.). Az üzleti logikát Enterprise Java Bean-ek (EJB) valósítják meg, a perisztencia réteg a Java Persistence API (JPA) EclipseLink implementációját használja, beleértve az EclipseLink multi-tenancy támogatását is. A felsoroltakon kívül további, a szoftveriparban aktuálisnak számító keretrendszerek és technológiák biztosítják a szoftver minőségét és hatékony működését.

2.1. Java Enterprise Edition

A Java Enterprise Edition (JEE) [5][6] többrétegű, osztott, több felhasználós, skálázható és biztonságos vállalati alkalmazások fejlesztését támogatja. Egy szabványcsalád, amelynek implementációi a különböző alkalmazásszerverek, amelyek a szabványoknak megfelelő keretrendszereket biztosítanak. Az OptInv a JEE referencia implementációjának számító Glassfish szervert használja.

A JEE keretrendszerek által nyújtott szolgáltatások (erőforrások menedzsmentje, dependency injection, perzisztencia, tranzakció-menedzsment, biztonság, üzenetközvetítésen alapuló kommunikáció, időzített szolgáltatások stb.) megkönnyítik a fejlesztés folyamatát. A programozók az üzleti logikai rétegre, az alkalmazás konkrét funkcionalitásainak megvalósítására koncentrálhatnak.

A szerver oldali Java EE komponensek olyan egymással kommunikáló szoftverkomponensek, melyek önálló funkcionalitással rendelkeznek. Két nagyobb kategóriába sorolhatóak: web komponensek, amelyek a szerver web konténerében menedzseltek és Enterprise JavaBeanek, melyek az alkalmazásszerver EJB konténerében menedzseltek.

2.1.1. Enterprise JavaBeanek

Az Enterprise JavaBean (EJB) [7] használatával építhető fel az alkalmazás üzleti logika rétege. Az OptInv rendszer szerver oldali komponensei is ilyen EJB komponensek.

Két EJB kategória létezik: Session Bean-ek és Message-Driven Bean-ek. Az előző EJB specifikációk esetében még beszélhetünk egy harmadik kategóriáról, az Entity Bean-

ekről, ezeket váltották a JPA entitások, de ezek esetében bizonyos dokumentációkban még mindig találkozhatunk az entity bean megnevezéssel.

A JPA entitások egyszerű POJO-k (Plain Old Java Object), amelyek JPA meta adatokat tartalmaznak. Tipikusan a rendszerek központi entitásait reprezentálják, a rendszer modelljét alkotó osztályok. Az OptInv rendszer adatmodelljét is ilyen JPA entitások alkotják. Az általában annotációk segítségével megadott JPA meta adatok alapján valósítják meg a JPA-t implementáló ORM (Object Relational Mapping) keretrendszerek a perzisztenciával kapcsolatos műveleteket.

A session bean-ek esetében két további kategóriát különíthetünk el: állapot nélküli (stateless) és állapottal rendelkező (stateful) session bean-ek. Az állapot nélküli session bean-ek két hívás között nem őriznek meg állapotinformációkat. Az állapottal rendelkező session bean-ek egy-egy klienshez vannak hozzárendelve, a kliens munkamenetén belül léteznek és több egymás utáni hívás között megőrizhetnek állapotinformációkat. A kliens soha nem közvetlenül a beanek-kel kommunikál, hanem interfészeken keresztül éri el ezeket. A session beanek három típusú interfészt valósíthatnak meg: Local, Remote és Endpoint interfészeket. A Local interfészeken belüli metódusok csak a konténeren belüli komponensek számára elérhetőek. Remote interfészeken keresztül távoli kliensalkalmazások is igénybe vehetik a komponensek szolgáltatásait. Az Endpoint interfészeket klasszikus webszolgáltatások esetében alkalmazzák, ahol SOAP protokollon keresztül lehet elérni a metódusok szolgáltatásait. Az OptInv projekt esetében a Repository, Service és Import rétegek megvalósítása állapot nélküli session beanek alkalmazásával történt, amelyek Local interfészeken keresztül kommunikálnak egymással, illetve a konténeren belüli webes komponensekkel.

A message-driven bean-ek aszinkron üzenetek feldolgozására alkalmasak. Általában JMS (Java Message Service) üzenetek figyelőjeként működnek, ezekre az üzenetekre reagálva végzik el az üzleti logikával kapcsolatos műveleteket.

2.1.2. Java Persistence API és EclipseLink

A Java Persistence API (JPA) [6] egy JDBC feletti absztrakciós szint, Java objektumok állapotának relációs adatbázisba való leképezését meghatározó szabvány. Biztosít az objektum-relációs leképezéshez szükséges meta adatokat, egy objektumorientált szemléletmódnak megfelelő lekérdező nyelvet (Java Persistence Query Language - JPQL) és

egy dinamikusan felépíthető lekérdezéseket támogató Criteria Query API-t. Az alkalmazásszerverek által biztosított, JPA szabványt implementáló ORM keretrendszereknek (EclipseLink, Hibernate stb.) köszönhetően egyszerűen és hatékonyan felépíthető a perzisztencia réteg, könnyen implementálhatóak az entitásokkal kapcsolatos adathozzáférési műveletek. A Java EE 7-es platform specifikációjának a JPA 2.1-es szabvány képezi részét. Ennek a verziónak az EclipseLink referencia implementációját használja az OptInv projekt is az adathozzáférési réteg megvalósítására.

Az objektum-relációs leképezés megvalósításához szükséges meta adatok megadása kétféleképpen lehetséges: az entitásokon belül alkalmazott annotációkkal, vagy xml állományok segítségével. Az OptInv az annotációs mechanizmust használja. A modell reprezentációjáért felelős osztályok, tehát a JPA entitások @javax.persistence.Entity annotációval vannak ellátva, az @javax.persistence.Id jelöli az elsődleges kulcsnak megfelelő azonosítót. Az entitás és a neki megfelelő adatbázis tábla közti megfeleltetésekre további JPA annotációk szolgálnak, ezek a getter metódusokon, tehát a tulajdonságok szintjén vannak alkalmazva. Az annotációk segítségével vannak meghatározva az adatbázis szintű validációval kapcsolatos megkötések is. Az annotációk nagy része opcionális, amennyiben követjük a Java elnevezési konvenciókat és az adatbázis sémán, valamint az objektumorientált modellen belül használt elnevezések megegyeznek, a keretrendszer automatikusan elvégzi a megfeleltetéseket. Ennek ellenére a későbbi esetleges módosítások szempontjából jó recept lehet explicit módon meghatározni a megfeleltetéseket.

Az entitások közötti kapcsolatok többfélék lehetnek: egyirányúak vagy kétirányúak, illetve számosságuk szerint egy az egyhez, egy a többhöz, több az egyhez, vagy több a többhöz kapcsolatok. Ezek a kapcsolatok szintén annotációk segítségével vannak meghatározva.

A perzisztenciával kapcsolatos műveleteket (például entitások mentése, módosítása, törlése) egy központi szolgáltatáson keresztül lehet végrehajtani, egy EntityManager interfészen keresztül. A perzisztencia műveletek végrehajtásakor az entitások hozzá lesznek kapcsolva egy ilyen EntityManager-hez, menedzselt állapotba kerülnek, állapotuk szinkronizálva lesz az adatbázisba mentett adatokkal, bármilyen változtatás automatikusan az adatbázisban is észlelhető lesz. Az entitásokat le is lehet kapcsolni az EntityManagerről, ezután az már nem végez szinkronizálást. Az adott adatforráson keresztül elérhető, EntityManager-hez hozzákapcsolható entitások perzisztencia egységeket (persistence unit)

képeznek. Az adatforrások konfigurálása az alkalmazásszerveren történik, az persistence unit-ok meghatározása a megfelelő telepítés leírón belül (persistence.xml).

Az aktuálisan menedzselt entitások együttesen egy perzisztencia kontextust alkotnak. A perzisztencia kontextus lehet tranzakció hatókörű vagy kiterjesztett kontextus. A @PersistenceContext annotáció segítségével beinjektált EntityManager-ek, amilyeneket az OptInv is használ, tranzakció hatókörűek, csak a tranzakció időtartama alatt lesznek menedzselt állapotban az entitások.

Az EntityManager használatával egyszerűvé válik a perzisztencia műveletek megvalósítása. Például, egy entitás lementése egyszerűen az EntityManager persist metódusával történik, törlése a remove metódus használatával. Ilyen módon az adathozzáférési réteg komponenseinek esetében hierarchia bevezetésére is lehetőség van, generikus típusokat alkalmazva az általános műveletek kiemelhetők egy közös repository interfészbe, illetve absztrakt alaposztályba. Lekérdezések létrehozására az EntityManager createQuery metódusát lehet használni, paraméterként átadva egy JPQL lekérdezést. Ugyanakkor natív query-k is alkalmazhatóak.

2.1.3. Tranzakció-kezelés

Bizonyos funkcionalitások megvalósítása több lépésből álló műveletsorok végrehajtásán keresztül lehetséges és ilyen esetekben mindig fontos a tranzakciók helyes kezelése. A Java EE esetében ez a Java Transaction API (JTA) specifikáción alapszik, és programozói szempontból kétféle képen lehetséges: deklaratív módon, vagy a program utasításai által. A második esetben explicit módon meg kell határozni, hogy hol kezdődik a tranzakció (begin transaction), valamint hogy hol és milyen módon fejeződik be (commit transaction, vagy rollback transaction).

A deklaratív módon történő tranzakció-menedzsment esetében a konténer által menedzseltek a tranzakciók, adott attribútumok alapján az EJB konténer határozza meg a tranzakció kezdő és végpontját. Tranzakció attribútumok használatával meghatározható, hogy egy komponens mely metódusai számára fontos a tranzakció hatókörön belüli végrehajtás. A tranzakció attribútumokat annotációk segítségével lehet megadni az osztályok vagy metódusok szintjén. Ha nincs meghatározva explicit módon a tranzakció attribútum, az alapértelmezett érték érvényesül, amely szerint a metódusok végrehajtásának tranzakció hatókörön belül kell megtörténnie.

Az OptInv projekt mind a konténer, mind a komponensek általi tranzakció-menedzsmentet alkalmazza. A programból történő tranzakció-kezelésre az adatimportot megvalósító modul esetében volt szükség.

2.1.4. Java EE biztonsági megoldások és JAAS

Minden összetettebb, több különböző szerepkörhöz tartozó felhasználó által használt alkalmazás esetében fontos a biztonság, a felhasználók beazonosítása és jogosultságaik kiosztása, valamint a biztonságos kommunikáció.

A Java EE biztonsági mechanizmusának alapjául a Java Authentication and Autorization Service (JAAS) szolgál. A JAAS modelljén belül a felhasználók különböző „személyazonosságokkal" rendelkezhetnek (principal), amelyek különböző szerepkörökhöz (role) tartozhatnak. A jogosultságok kiosztása a szerepköröknek megfelelően történik. A felhasználók, szerepkörök és biztonsági szolgáltatók, valamint vezérelvek biztonsági tartományokat (security realm) alkotnak, amelyek az alkalmazásszervereken konfigurálhatóak. A tartományhoz tartozó adatok (felhasználók, csoportok stb.) beolvashatóak állományokból, saját adatbázisból (ezt a megoldást alkalmazza az OptInv), vagy külső rendszerek is szolgáltathatják ezeket (pl. LDAP protokollon keresztül).

Programozói szempontból, a tranzakció-kezeléshez hasonlóan, két lehetőségünk van a biztonság biztosítására: történhet deklaratív módon vagy a program által szabályozva.

A deklaratív megoldás esetében a biztonsági követelményeket telepítés leíróban, illetve annotációk használatával lehet megadni. A telepítés leíró egy XML állomány amelyben le van írva, hogy mi szükséges a felhasználó bejelentkezéshez, hogyan van ez megoldva (például, a browser beépített beléptető felületét használva, vagy saját belépési felület segítségével), milyen szerepkörök felhasználói jelentkezhetnek be a rendszerbe, valamint megadhatóak biztonsági megszorítások (bizonyos erőforrásokhoz milyen szerepkörök felhasználói férhetnek hozzá).

A komponensek szintjén annotációk segítségével meghatározható, hogy adott metódust milyen szerepkörökbe tartozó felhasználók hívhatnak meg. Szerepkörök deklarálása a @DeclareRoles annotációval lehetséges, a hozzáférés engedélyezéséhez adott metódushoz (vagy a teljes komponens szintjén) a @RolesAllowed annotációt kell használni. Ha nincs meghatározva más biztonsági megszorítás, alapértelmezetten nem kerül leellenőrzésre a felhasználó kiléte és jogosultsága, tehát bárki által hívható az illető metódus. Fontos lehet

továbbá a RunAs annotáció, amelynek segítségével egy komponens esetében lehet meghatározni azt, hogy milyen szerepkörbe tartozzon, milyen jogosultságok alapján hívhatja egy másik komponens metódusait.

A program utasításain keresztül megvalósított biztonsági mechanizmus olyan esetekben lehet hasznos, amikor a deklaratív megközelítés nem elegendő a feltételek pontos meghatározásához, a jogosultságok ellenőrzéséhez. Futási időben lekérhető az EJB komponensek környezetétől (SessionContext) a hívó fél „személyazonossága" (principal), ezen keresztül a felhasználó neve, illetve az, hogy a hívó fél milyen szerepkörbe tartozik. Ezek alapján, illetve további feltételeket figyelembe véve lehet elvégezni különböző műveleteket adott erőforrásokon.

Az OptInv projekt keretén belül egyrészt különböző szerepkörökről és ennek megfelelően különböző jogosultságú felhasználókról beszélhetünk, másrészt nagyon fontos, hogy adott céghez tartozó felhasználók csak a saját cégük adataihoz férjenek hozzá. A rendszer mindkét biztonsági megoldást alkalmazza: deklaratív módon vannak meghatározva a komponensekhez való hozzáféréshez szükséges jogosultságok, a programból van szabályozva, hogy milyen felhasználók milyen adatokat kérhetnek a rendszertől. Ez utóbbi funkcionalitás az EclipseLink multi-tenancy mechanizmusán keresztül megvalósított. A kapcsolódó (adatlekérést végző) szolgáltatások hívásait Interceptorok fogják el, amelyeken belül ellenőrzésre kerül a felhasználó személyazonossága, és ennek megfelelően lesznek elvégezve beállítások, amelyek alapján a rendszer kiszolgáltatja az adatokat.

2.1.5. Interceptorok

A Java EE esetében használható interceptorok komponensek életciklusával kapcsolatos eseményekre reagálhatnak, vagy metódushívásokat elfogva hajthatják végre utasításaikat. Az aspektusorientált nyelvekhez hasonlóan az átmetsző követelményekkel kapcsolatos műveletek megvalósítását és modularizálását célozzák.

Az interceptorok deklarálása történhet annotációk vagy telepítés leírók alkalmazásával és kétféleképpen lehetséges: metódusként a célosztályban, vagy külön interceptor osztályban. Az interceptor utasításai automatikusan végre lehetnek hajtva például adott komponens példányosításakor, konténerből való eltávolításakor, adott időtúllépés esetén, vagy metódus meghívásakor. Annak érdekében, hogy egy metódus interceptor metódussá váljon, az @AroundInvoke annotációval kell ellátni. Ha az interceptor metódus

deklarálása nem a célosztályban történik, hanem egy külön osztályban, akkor az interceptor osztály szintjén az @Interceptor annotációt kell használni. Egy osztályhoz, vagy metódushoz több interceptor is hozzárendelhető (@Interceptors annotáció), ezek deklarációjuk sorrendjében lépnek működésbe és prioritási szintek is meghatározhatóak. A célosztály minden egyes példányosítása maga után vonja a hozzákapcsolt interceptor osztályok példányosítását is.

Az OptInv keretein belül minden adatlekéréssel kapcsolatos szolgáltatás interceptálva van, így a metódusok végrehajtása előtt lehetőség van az egyes cégekhez tartozó információk beállítására, a bejelentkezett felhasználók személyazonossága szerint. Ezen a megoldáson keresztül biztosított a biztonsági mechanizmussal összekötött multi-tenancy. A mindenütt szükséges ellenőrzések és beállítások egyetlen helyen, az interceptoron belül vannak implementálva. Továbbá, az interceptorok elegáns megoldást biztosítanak a szolgáltatás rétegen belüli validációra is.

2.2. EclipseLink alapú multitenancy

A multitenancy [8][9] egy alapelv, amely arra összpontosít, hogy egy alkalmazás architektúrája úgy legyen felépítve, hogy egyetlen szoftver képes legyen több kliens (tenant) kiszolgálására. Az OptInv alkalmazás multitenancy alkalmazása révén vált alkalmassá arra, hogy bár egy adatbázissal és egy sémával rendelkezik, képes egyidejűleg több cég kiszolgálására.

Multitenant alkalmazások megvalósításában különböző keretrendszerek nyújtanak segítséget. Az egyik ilyen keretrendszer az EclipseLink. Az adatbázisok tartalmazhatnak tenant-függő és nem tenant-függő adatokat is. A multitenancy megvalósítása hasonlít az adatok JPA alapú manipulációjához olyan szempontból, hogy az egyes meta adatok megadhatóak annotációk segítségével vagy XML állományban.

Az egyik legalapvetőbb annotáció, melynek az annotációk segítségével felépített multitenant alkalmazásokban mindenképpen szerepelnie kell, a @Multitenant annotáció. Annak függvényében, hogy milyen típusú a multitenancy, az annotáció után zárójelben megadható a következő értékek egyike: SINGLE_TABLE, VPD, TABLE_PER_TENANT. Ha az annotáció után nincs megadva semmi, a rendszer alapértelmezettként a SINGLE-TABLE értéket használja. Ebben az esetben a különböző tenant-ok sémája egységes, az adatbázis osztott, az adatbázisban a különböző tenantok-hoz tartozó sorok bármilyen

sorrendben követhetik egymást. Annak érdekében, hogy helyesen működjön az alkalmazás, szükséges még egy @TenantDiscriminatorColumn (name="TENANT_ID", contextProperty = "tenant-id") annotáció is, amely meghatározza, hogy melyik az az egyedi tulajdonság, amely minden tenant/cég esetében más és más, de egy tenanthoz tartozó adatok esetében azonos (lehet például a cég neve). A TENANT_ID helyett annak az oszlopnak a nevét kell beírni, amely ezt az információt rögzíti. A contextProperty tulajdonság megadása opcionális, alapértelmezetten „eclipselink.tenant-id". Ez arra vonatkozik, hogy a későbbiekben milyen név segítségével lehet beállítani a tenant tulajdonságát. A multi-tenancy SINGLE-TABLE változata nem adatbázisfüggő, a tenant-függő adatokkal történő adatbázis műveletek esetében az EclipseLink egyszerűen hozzáfűz a művelethez egy feltételt, amely a tenant-függő oszlop nevét és a beállított értéket tartalmazza („where column-name = value").

A VPD beállítás multitenancy esetén hasonlóképpen működnek, a különbség csak az a két típus között, hogy ez utóbbit csak az Oracle Virtual Private Database típusú adatbázis esetén lehet használni. VPD használatakor nem az EclipseLink felelős a megfelelő feltétel hozzáfűzéséért az SQL lekérdezéshez, hanem az adatbázis-menedzsment rendszerre hárul ez a feladat.

A TABLE_PER_TENANT opció esetében, az előbbiekhez hasonlóan egy adatbázis van, azonban minden egyes tenant-hoz tartozik egy külön tábla.

Annak érdekében, hogy a multitenancy működjön, nagyon fontos a tenant-okat azonosító tenat-id értékének beállítása. Erre két lehetőség van: az érték beállítható statikusan vagy dinamikusan. Statikusan a persistence unit konfigurálásakor adható meg. A beállítást elvégezve unit-tal kapcsolatos összes adatbázis hozzáférési művelet adott tenant-ra fog vonatkozni. Bármilyen tenant-függő adattal kapcsolatos adatbázis lekérdezés esetén automatikusan azokat a sorokat fogja visszatéríteni a rendszer az adatbázisból, ahol a tenant-okat megkülönböztető oszlopon belüli érték megegyezik a unit beállításánál megadott értékkel. Hasonlóan, adatbeszúrás esetén a tenant-id értékét nem kell megadni, az EclipseLink automatikusan tölti ki ezt a unit konfigurálásánál megadott értékkel.

Ahhoz, hogy a tenant-id értékét futás közben lehessen változtatni, ahogyan erre az OptInv esetében is szükség volt, a hozzá tartozó értéket nem a persistence unit tulajdonságaként kell megadni. Lehetőség van arra, hogy a tulajdonság értéke az EntityManager létrehozásakor legyen megadva. Az eredmény ugyanaz lesz, mint az előbbi statikus esetben, azonban ez a rész a forráskódba kerül és nem egy külső beállításokat

tartalmazó állományba, így például a bejelentkezett felhasználó függvényében dinamikusan változtatni lehet a tenat-id-t. Az OptInv esetében ezt a feladatot végzik el az adatlekérési szolgáltatásokat interceptáló metódusok.

2.3. A Vaadin keretrendszer

A *Vaadin* [10] a Vaadin LTD által fejlesztett AJAX alapú web alkalmazás keretrendszer, amely lehetőséget nyújt a fejlesztőknek összetett felhasználói felületek készítésére Javaban. Két részből áll, a szerver oldali eszközkészletből és a kliens oldali motorból.

A szerver oldali rész a Java asztali alkalmazások fejlesztésénél használt grafikus eszközkészletekhez (AWT, SWING, SWT) hasonló megoldásokat kínál a fejlesztőknek webes felületek elkészítéséhez. A felhasználói felület felépítését segítő komponenseken kívül ez a rész felel az Ajax kommunikációért a böngésző és szerver között.

A kliens oldali motor magába foglalja a GWT-t (Google Web Toolkit), amely a Java kódot JavaScript kódra fordítja.

A Vaadin egy gazdag komponenskészletet bocsájt a fejlesztők rendelkezésére, amellyel egy teljes web alkalmazás felhasználói felületét fel lehet építeni. Amennyiben ez a készlet mégsem elegendő, a fejlesztők a Vaadin Directory-ban rengeteg kiegészítő komponens (add-on/widget) közül választhatnak. Ha specifikus komponensre van szükség, azok könnyen megírhatóak, az eszközkészlet kibővíthető HTML5, JavaScript és GWT segítségével. Megjelenítés szempontjából a keretrendszer beépített témákat biztosít, és ezek mellett CSS segítségével teljesen testre szabható a komponensek kinézete.

2.4. További technológiák

Az OptInv projekt a Java EE részét képező Bean Validation specifikációnak megfelelően oldja meg a modellek szintjén történő adat-validációt. Az alkalmazás modelljében annotációkkal vannak megadva a különböző megszorítások. Ezt a módszert a Vaadin komponensei is támogatják, így a Bean Validation meta adatok alapján a keretrendszer biztosítani tudja a felület szintjén a validációt és ennek alapján a megfelelő visszajelzést a felhasználónak.

A JDBC (Java DataBase Connectivity) standard API-t biztosít relációs adatbázisokat felhasználó alkalmazások számára. Ez a technológia áll a JPA alapú adathozzáférési réteg

hátterében, de fontos szerepet játszik az adatok importálását végző modul esetében is. A külső adatbázisokkal (könyvelési programok adatbázisai) az OptInv JDBC API-n keresztül kommunikál. A Hamor esetében például JDBC-ODBC bridge drivert alkalmaz a .dbf formátumban tárolt adatok beolvasására.

A naplózás megvalósításához az slf4j tűnt a legmegfelelőbbnek, mivel több naplózási keretrendszer felett képez absztrakciós szintet, a konkrét keretrendszer (az OptInv esetében jelenleg log4j) ezután egyszerűen beköthető a rendszerbe, a kód módosítása vagy újrafordítása nélkül.

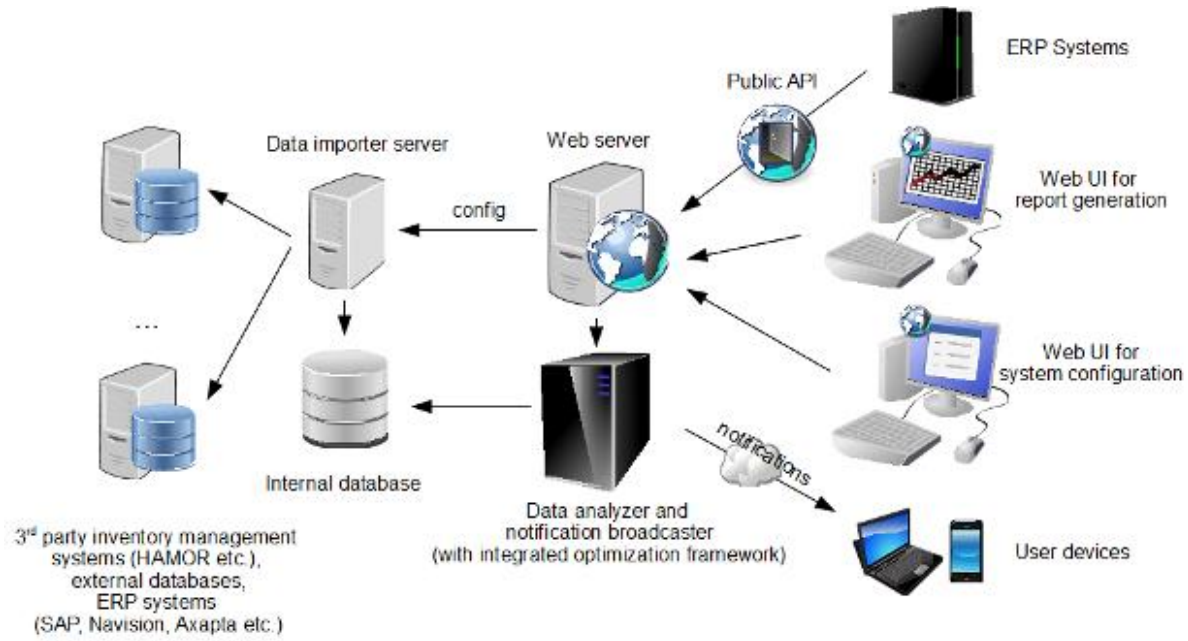
Az OptInv projekt tartalmaz egy FTP kliensalkalmazást is, amely a központi szerver és a kliens gépe között bonyolítja le a könyvelőprogramok adatbázisait tartalmazó állományok átvitelét (bár, a jelenlegi verzióban ezek már feltölthetőek a webes felületről is). Az asztali kliensalkalmazás megvalósításában az Apache Commons projekt hálózati kommunikációt támogató csomagjai nyújtottak segítséget.

3. Az OptInv projekt

A dolgozat következő része röviden összefoglalja az OptInv rendszer alapvető funkcionalitásait, bemutatja a rendszer modelljét és architektúráját, ismerteti megvalósításának fontosabb részleteit és működését.

3.1. Alapvető funkcionalitások

A projekt alapelképzelése, hogy szoftver, mint szolgáltatásként működjön (Software as a Service - SaaS). Középpontjában egy alkalmazásszerver áll, ehhez csatlakoznak a különböző kliensek. A kliensek adatbázisának adatai át lesznek másolva az alkalmazásszerverre, és be lesznek importálva az OptInv belső adatbázisába a rendszer belső modelljének megfelelően. A rendszer konfigurálását (pl. cégprofilok: felhasználók, adatbázis típus, utó-feldolgozási műveletek stb.) a megfelelő jogosultságú felhasználók egy webes felületen keresztül tehetik meg. A beimportált adatok alapján készített kimutatások szintén egy webes felületen jeleníthetőek meg. A későbbi verziók esetében lehetőség lenne a rendszer összekapcsolására ERP rendszerekkel egy publikus API-n keresztül, valamit mobil kliensalkalmazások segítségével értesítések kiküldésére a készletekkel kapcsolatos fontosabb változások esetében.



1. ábra: OptInv - Szoftver, mint szolgáltatás

A projekt jelenlegi változata egy prototípus, amely az OptInv Backend, az OptInv Web, az ezeket összefogó OptInv szerver alrendszerekből áll.

Az OptInv Backend biztosítja az adathozzáférési, az üzleti logika, valamint a szolgáltatási rétegeket. Tartalmazza a Data Importer modult, amely adatokat importál a felhasználók szerverre átmásolt állományaiból (könyvelési programok - pl. Hamor - adatbázisai). Az adathozzáférési rétegen keresztül elmenti ezeket az adatokat az alkalmazás központi adatbázisába, a projekt belső adatmodelljének megfelelően. Komponensmodelljének köszönhetően könnyen kiegészíthető új formátumokat támogató adatimportáló komponensekkel. A Backend tartalmazza továbbá a kimutatásokhoz szükséges számításokat elvégző komponenseket biztosító adatelemző modult.

Az OptInv Web alrendszer egy web alkalmazás, amelynek feladata, hogy kommunikáljon a webes kliensekkel, fogadja azok kéréseit, a Data Analyzer rendszer segítségével jelentéseket, statisztikákat generáljon a kliensek számára. Kommunikál a Data Importer modullal is, hogy továbbítani tudja a webes felületen keresztül megadott, import folyamatra vonatkozó konfigurációs paramétereket, illetve lehetőséget ad a kliensek adatait tartalmazó állományok feltöltésére (erre egy különálló asztali FTP kliensalkalmazás is rendelkezésükre áll). Egy webes felület segítségével lehetővé teszi az adatbázisban való böngészést, kimutatások megtekintését, különböző keresési, szűrési és rendezési lehetőségeket biztosítva a bejelentkezett felhasználó számára. Egy külön felületen lehetőséget

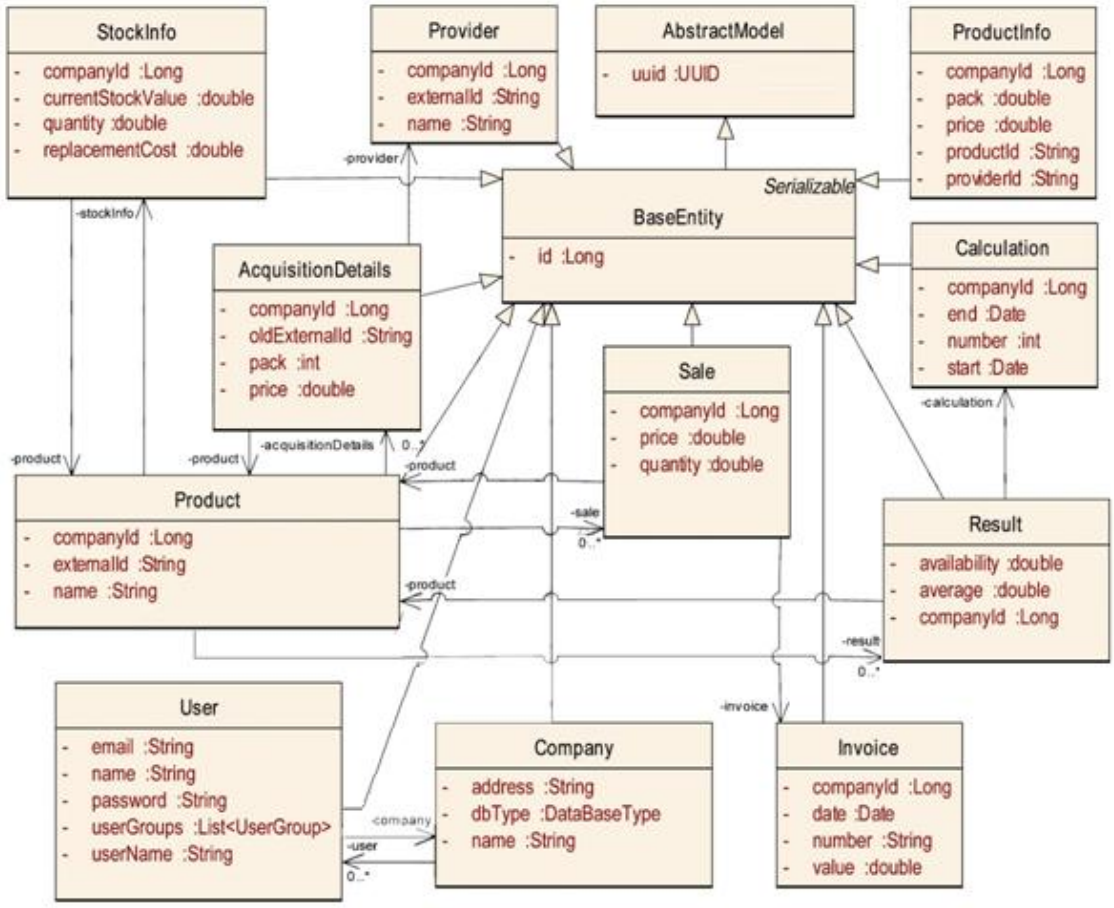
ad a megfelelő jogosultságú felhasználóknak a rendszer konfigurálására (cégprofilok beállítása, beleértve a használt adatbázis típusát, a felhasználók menedzsmentje stb.).

A webes felületen a felhasználónak lehetősége van megtekinteni a cégéhez tartozó adatokra vonatkozó kimutatásokat. Információkat kaphat az egyes termékekről, beleértve a beszállítókat, árakat, beszállítói és eladási csomagolási formákat. Megnézheti, hogy milyen termékek vannak jelenleg raktáron és milyen mennyiségben, mennyi volt adott periódusra az átlagos fogyasztás, ennek megfelelően mennyi ideig elegendő még a készlet, illetve mennyi annak az értéke. Megtekintheti az eladásokat, valamint havi bontásban kimutatást kaphat a termékek fogyásáról. Ezek az adatok táblázatokon belül vannak megjelenítve, a felhasználónak lehetősége van szűrésre (pl. termékek szűrése beszállítók szerint stb.) és minden oszlop szerint rendezhetőek az adatok. A rendezésnek köszönhetően könnyen megtekinthető, hogy melyik termékből nem volt rendelés és állnak fölöslegesen nagy mennyiségek a készleten, melyik termékekből van sok eladás, melyik termékekből kell pótolni a készletet stb. Adott terméket kiválasztva az eladások havi bontásáról grafikonos megjelenítést is kérhet, így vizuálisan könnyen elkülöníthetőek a rekurrens, sporadikus és szezonális termékek.

3.2. Környezeti elemzés

A központi entitásokat reprezentáló osztályok közös absztrakt alaposztálya az AbstractModel, amely biztosít egy rendszeren belül globálisan egyedi azonosítót (Universal Unique Identifier). Ebből az osztályból származik a BaseEntity, amely az elsődleges kulcsnak megfelelő azonosítót biztosítja, illetve az entitások szerializálhatóságát, implementálva a Serializable interfészt. Az OptInv projekt leglényegesebb entitásai közé tartozik a Product, illetve a Sale és az ehhez kapcsolódó ProductInfo, StockInfo, AcquisitionDetails, Provider illetve Invoice. Ezek írnak le egy-egy terméket, illetve egy-egy eladást, innen lehet megtudni, hogy az adott terméket milyen beszállítótól és milyen árban lehet rendelni, mekkora mennyiség van még raktáron, milyen kiszerelésben kapható a beszállítónál, mikor volt belőle eladás, mekkora mennyiséget adtak el. Ezek az információk céghez (Company) kötöttek és minden céghez tartoznak felhasználók (User), akik a rendszert kezelik. A cégek esetében fontos adat az általuk használt könyvelő program/ERP rendszer adatbázisának típusa. A különböző kimutatásokhoz tartozó számításokat Calculation objektumok azonosítják, a

gyorsabb működés érdekében a rendszer ezeknek eredményeit (Result objektumok) is eltárolja.

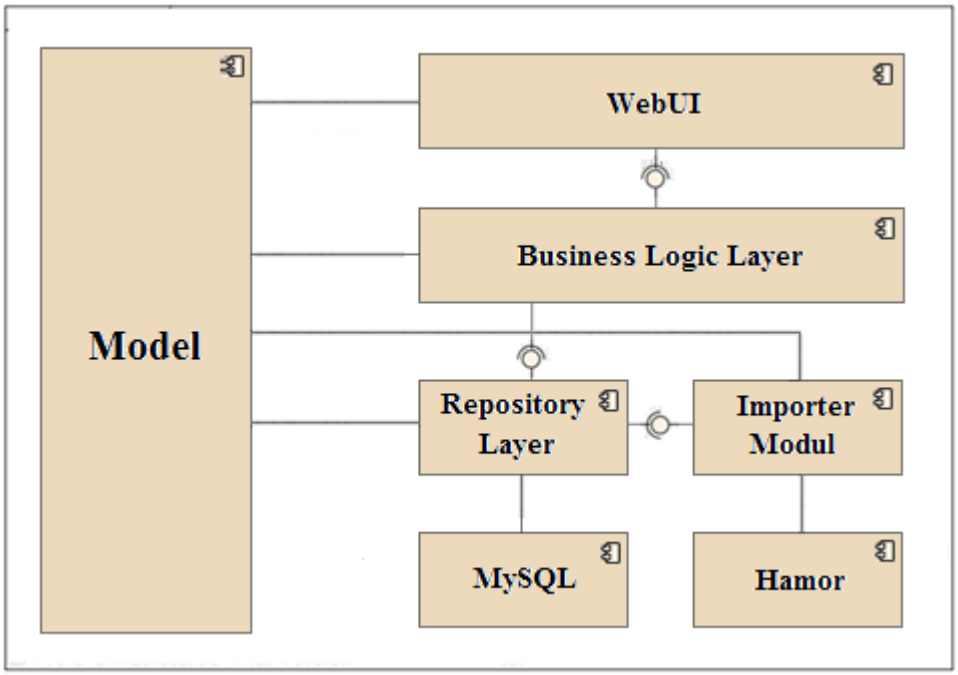


2. ábra: Az OptInv projekt részleges osztálydiagramja
(a metódusok kimaradtak és csak a fontosabb attribútumok vannak feltűntetve)

3.3. Architektúra

Az OptInv architektúrája többrétegű, a modell (entitások) közös, a különböző rétegek megosztva használják. Az adatbázis menedzsment rendszerrel (MySQL) az adathozzáférési (repository) réteg kommunikál. Az adatimportálásért felelős modul (jelen állapotában Hamor specifikus importer komponenst tartalmaz) a repository rétegen keresztül továbbítja a Hamor adatait az alkalmazás saját adatbázisába. Az adathozzáférési réteg fölött helyezkedik el az üzleti logikáért felelős komponenseket tartalmazó Business Logic Layer, amely egy szolgáltatás rétegen (szolgáltatás interfészek) keresztül kommunikálhat a kliensalkalmazásokkal. A web réteg a webes komponenseket tartalmazza, szerver oldali

grafikus komponenseket felhasználva felépíti a felületet, a szolgáltatás rétegen keresztül kommunikál az alkalmazás backend részével. Mivel a különböző modulok jelenleg közösen képezik az OptInv szerver részét, egy ear (Enterprise Archive) állományba tömörítve vannak kitelepítve az alkalmazásszerverre. A komponensek egy konténeren belül menedzseltek, egymással lokális interfészeken keresztül kommunikálnak.



3. ábra: Az OptInv projekt rendszerarchitektúrájának diagramja

3.4. A megvalósítás fontosabb részletei

Az Optinv alkalmazás a JEE platformra épül, az adatimportálás, repository és szolgáltatás rétegeket EJB-k, pontosabban állapot nélküli session beanek alkotják, amelyek egymás között, valamint a webes komponensekkel lokális interfészeken keresztül kommunikálnak. A komponensek közötti függőségek dependecy injection (DI) tervezési minta alapján vannak megoldva, az EJB, illetve a CDI (Context and Dependency Injection) DI mechanizmusát alkalmazva.

A modell adatait JPA entitások reprezentálják, az adathozzáférési réteg EclipseLink JPA implementációt használ. Az ORM megfeleltetéshez szükséges JPA meta adatok annotációk segítségével vannak megadva. A repository réteg komponensei hierarchiába rendezettek, az alapműveletek egy közös absztrakt ősosztályba vannak kiemelve és ezen belül történik meg az Entity Manager injektálása is. Az entitások szintjén történő validációval kapcsolatos megkötések Bean Validation annotációk segítségével vannak meghatározva.

A tranzakció-menedzsment deklaratív módon van megoldva, kivéve az adatok importálását, amelynek során a komponensek menedzselik a tranzakciókat.

Az alkalmazásba való bejelentkezéshez az alkalmazásszerver beléptető mechanizmusa van felhasználva. A biztonsági mechanizmus, a jogosultságok kezelése JAAS alapú, egy JDBC security realm segítségével van megoldva, az alkalmazás adatbázisából vannak betöltve a felhasználókra és szerepkörökre vonatkozó adatok.

Az EclipseLink használata révén támogatott a multitenancy. A SINGLE_TABLE változat van alkalmazva, mivel az egyes cégek esetében ugyanazokra az adatokra van szükség, az adatbázis séma azonos minden tenant esetében. Tenant id-ként a cég azonosítója van használva. Nem minden entitás tenant-függő, csak a termékekhez kapcsolódó entitások. A multitenancy-hoz szükséges meta adatok a termék és a hozzá kapcsolódó entitások felannotálása révén vannak megadva. A tenant id beállítása dinamikusan történik, interceptorok fogják el az adatlekéréssel kapcsolatos szolgáltatások hívásait és a SessionContext-től lekérve a bejelentkezett felhasználó nevét, azonosítják a felhasználóhoz tartozó céget, ennek megfelelően beállítva a tenant id-t.

3.4.1. Adatimportálás Hamor rendszerekből

Annak érdekében, hogy az egyes cégekhez tartozó adatokról kimutatásokat készítsen az OptInv rendszer, be kell importálnia az adatokat az illető cég könyvelőprogramjából a saját adatbázisába. Ez a lépés azért nélkülözhetetlen, mert a szükséges adatok a könyvelőprogram adatbázisában más formában, szétszórtan találhatóak meg, mivel egy könyvelőprogram teljesen más célt szolgál, mint egy készlet-optimalizáló alkalmazás.

Az OptInv prototípus fejlesztésében partnerként résztvevő Galfi Servco székelyudvarhelyi autóalkatrész kereskedéssel foglalkozó cég Hamor könyvelési programot használ, így az első lépésben a Hamor adatainak importálása történt meg. A Hamor esetében az adatok dbf állományokban vannak tárolva, az importer modul JDBC API-n keresztül éri el ezeket az adatok JDBC-ODBC bridge alkalmazásával.

Minden olyan adat, amelyre az importálás során többször is szükség lesz, ideiglenesen hashmap-ben lesz eltárolva a hatékonyság biztosításának érdekében. A hashmap kulcs értéke mindig a dbf állományból kiolvasott információ, az értéke pedig a saját adatbázisba lementett objektum lesz. A termékekre vonatkozó fontosabb információk a Nommarfa.dbf-ben találhatóak, a Grupa oszlopban található a beszállítók rövidítése, melyet felhasználva Provider

objektumok kerülnek létrehozásra. A Cod, a Denum és a Descriere oszlop értékeit felhasználva lesz felépítve a Product objektum, a cod a cég által termékhez rendelt kulcsot tartalmazza, a Denum és a Descriere oszlopok a termék megnevezést és leírást tartalmazzák, ezek összevonva kerülnek be a Product objektumba. A beszállító és termék közti kapcsolat az AcquisitionDetails objektumban lesz tárolva, továbbá itt lesz eltárolva az ebben a fázisban még ismeretlen beszállítási ár és beszállítói csomagolási forma. A StockInfo objektum, mely az egyes termékek raktáron lévő mennyiségét, legkisebb beszerzési árát és ennek a kettőnek a szorzatát tartalmazza, itt lesz létrehozva, azonban ebben a lépésben még csak a Product adattagja lesz inicializálva. A Product objektum tartalmaz egy AcquisitionDetails és StockInfo adattagot, melyek ebben a lépésben lesznek beállítva. Az eladásokhoz kapcsolódó számlák adatai a Facturi.dbf-ben találhatóak meg, pontosabban ennek a Nrfac, Tip, Datafac és Valvin oszlopaiban. A számlaszám a Nrfac és Tip összeolvasztásával válik egyedivé. A Datafac a számla kiállításának dátumát, a Valvin a számla összegét tartalmazza. Ennek a három adatnak a segítségével lesz felépítve és a belső adatbázisba mentve egy Invoice objektum. A Facrind.dbf Codmat, Nrfac, Tip, Cant, Puliv, Pucont oszlopaiban találhatóak az egyes eladásokkal kapcsolatos információk. A Cant oszlop mutatja az eladott mennyiséget, a Puliv és Pucont különbség az eladás értékét jelzik. Ezeknek az adatoknak a segítségével fel lehet építeni és a saját adatbázisba menteni egy Sale objektumot. Mivel a Product objektum tartalmaz egy eladás listát, így az aktuális eladást is hozzá kell adni ehhez a listához. A következő lépés a raktáron lévő mennyiség importálása, ehhez azonban több dbf bejárására is szükség van. A Tran.dbf rögzíti az egyes üzleti tranzakciókat. Innen két oszlop lesz figyelembe véve, a Nrtran és a Stare oszlopok. Egy tranzakció csak akkor lesz érvényesnek tekintve és figyelembe véve, ha a Stare mezőjének értéke V. Ahhoz, hogy ténylegesen megkapjuk a raktáron lévő mennyiséget, a Lotm.dbf Nrtran, Codmat, Stoci, Iesiri oszlopait kell végigjárni. Ha az adott Nrtran-nak megfelelő Stare mező értéke V, valamint ha a Stoci és Iesiri oszlopok különbsége pozitív, akkor az eredményt hozzá kell adni az adott Product objektumhoz tartozó StockInfo objektum quantity adattagjához, majd a módosult Product objektumokat frissíteni kell az adatbázisban és a memóriában is. A termékhez kapcsolódó beszállítási információkat a Furnart.dbf ST2A1, ST2A3, ST2B3 és ST2B4 oszlopai tartalmazzák. Ezek egy ProductInfo objektumba tárolva mentődnek le az adatbázisba. A szükséges adatok: a beszállító kódja (az ST2A1-es oszlopban), a termék kódja (az ST2A3 oszlopban), a beszállítói ár (az ST2B3 oszlopban), valamint a beszállítói csomagolási forma

(az ST2B4 oszlopban). A kapott dbf-ek hiányosságai miatt ezeket az adatokat a prototípus esetében csak külön, a többi adattól függetlenül lehetett tárolni.

A Galfi Servco esetében is szükséges volt utófeldolgozást végezni az adatokon. Mivel a tőlük kapott adatbázisban a termékkódok mellett szerepelt a beszállító kezdőbetűje is, így ugyanaz a termék két külön termékként jelent meg, ha két beszállító is szállította. Az egységes és helyes készlet-optimalizáció érdekében az ilyen termékek összevonhatóak. Az utófeldolgozás során az adatbázisból törölve lesz az összes olyan eladás, amelyhez olyan számla tartozik, ahol a számlaszám AIMTI-vel végződik, majd ezt követően, mivel ezekre a számlákra már nincs hivatkozás ezeket is ki lehet törölni az adatbázisból. A Galfi Servco adatbázisában minden termék kódja hat számjegyet tartalmaz, majd ezt követően tartalmazhat rövidítéseket a beszállítóra/típusra nézve. Ennek alapján egyesítve lesz minden olyan termék és a hozzájuk tartozó információ, amely rendelkezik az említett tulajdonsággal, valamint törölve lesznek azok, amelynek kódja rövidebb hat karakternél, vagy nem hat számjegyből áll.

3.4.2. Számítások és statisztikai adatok

A rendszer kimutatásokat készít a felhasználóknak, ehhez számításokat végez, amelyeknek eredményeit táblázatok vagy grafikonok formájában jeleníti meg.

A felületen egy táblázatban láthatóak a termékek, amelyekkel kapcsolatban adott időintervallumra vonatkozó statisztikai információkat is kapunk. A táblázat oszlopai: a termék azonosítója, a termék neve, az időintervallumra kiszámolt átlagos fogyás (average consumption), beszállítói ár (replacement cost), a készleten levő termékek értéke (current stock value), a készleten levő termék darabszáma (on stock). Az is megjelenik, hogy mennyi időre elegendő még a termék az aktuálisan kiszámolt átlagos fogyást figyelembe véve (availability). Ezek az oszlopok mind rendezhetőek. Például, az átlagos fogyás oszlopát rendezve csökkenő sorrendbe, megtudhatjuk, hogy melyik az a termék, amelyet a leggyakrabban vásárolnak. Ugyanakkor megláthatjuk, hogy melyek a kevésbé vásárolt termékek, majd a raktáron lévő készlet értéke, vagy a raktáron lévő mennyiség szerint rendezve ezeket, láthatjuk, hogy melyekből tart a cég fölösleges készleteket.

Egy külön nézeten különböző jelentéseket kérhetünk és tekinthetünk meg. Jelenleg kétféle jelentést lehet generálni: havi lebontásban kaphatunk kimutatást a termékek eladásáról táblázatos formában, vagy adott terméket kiválasztva grafikonos formában.

A kimutatásokhoz szükséges számításokat az importálás után, illetve a kliens explicit kérésére az üzleti logika réteg komponensei végzik, a nagyobb hatékonyság érdekében az eredményeket a rendszer adatbázisban is tárolja. A kimutatásokat megjelenítő felületek Vaadin komponensek segítségével vannak felépítve, a grafikonos megjelenítés a dCharts add-on által biztosított.

3.4.3. Felhasználói felület

Az OptInv projekt esetében a teljes felhasználói felület a Vaadin keretrendszer segítségével van megvalósítva. Alapját egy UI (UserInterface)-ből származtatott osztály képezi, amelyen belül egy Navigator segítségével lehet navigálni az oldalak (az OptInv esetében tab-ok) között. A felület fülekre (tab) osztott megjelenítést alkalmaz, amely a Vaadin TabSheet komponensével van megoldva. A felhasználó szerepkörének megfelelően, más-más felület jelenik meg, különböző tab-ok láthatóak.

Adminisztrátorként belépve egyetlen tab-ot láthat a felhasználó, amely a cégek és a nekik megfelelő felhasználók menedzsmentjére szolgál. Itt lehet hozzáadni, törölni, módosítani cégprofilokat, illetve a hozzá tartozó felhasználókat. Az adatmódosítási felületek megvalósítása BeanFieldGroup komponensek segítségével történt. A FieldGroup-okhoz hozzákapcsolhatóak a modellobjektumok és a Bean Validation alapú validáció is biztosított a keretrendszer által.

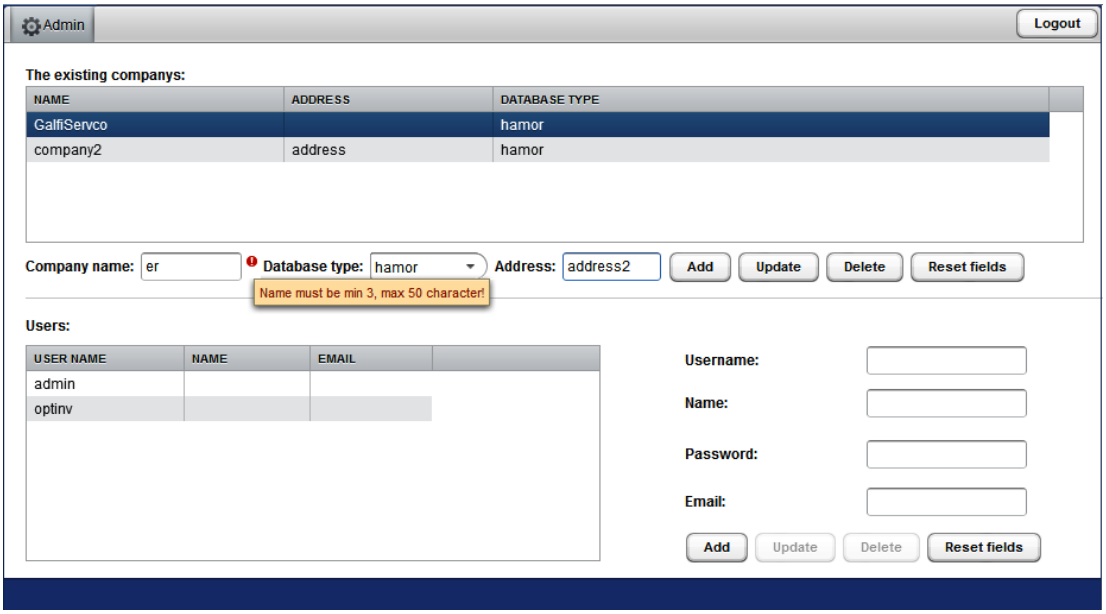
Egyszerű felhasználóként belépve több fül jelenik meg, amelyek mind a beimportált adatokkal kapcsolatos nézeteket biztosítanak. Az első nézeten a termékekkel kapcsolatos információk jelennek meg egy táblázatban. A második nézet szintén egy táblázatban mutatja az eladásokkal kapcsolatos információkat. A harmadik nézeten a jelentések tekinthetőek meg: az eladásokat havi bontásban mutató táblázat, illetve az egyes termékekre vonatkozó grafikon alapú jelentés. A grafikonos megjelenítő a dCharts widget-et használja, amely ingyenes komponens, és lehetőséget ad különböző típusú grafikonok megjelenítésére. A negyedik nézet az importáláshoz kapcsolódik, megjeleníti a beimportált adatbázis dátumát (az utolsó eladás dátuma), és itt lehet újra beimportálni az adatokat egy újabb adatbázisból. Az utolsó fül a könyvelő program adatbázis állományainak feltöltésére szolgál, az állományok drag-and-drop módszerrel is felvihetőek a felületre. Amennyiben már van a szerveren az adott cégtől származó adatbázis, és a felhasználó újakat tölt fel, a régiek átkerülnek egy *archive* mappába, dátummal ellátva.

Több adat esetén minden nézeten biztosított a lapozási lehetőség, a hatékonyság növelésének és a memóriaigény csökkentésének az érdekében az adatok lekérése dinamikusan történik. Minden kimutatás esetében biztosítottak a szűrési és rendezési lehetőségek. A táblázatok a Vaadin Table komponensére épülnek, de az adatok betöltéséhez használt lazy loading mechanizmus a projekten belül van implementálva. A Vaadin Table-je ugyan rendelkezik saját rendező funkcionalitással, de az csak akkor ad helyes eredményt, ha az összes szükséges adat a táblázat konténerében van. Mivel egy cég adatbázisában rengeteg adat lehet, ezek megjelenítésénél nem optimális az egészet betölteni, ezért a rendező algoritmusok is saját implementációk. Érdemes megemlíteni, hogy a rendezések során több variáció lehetséges, így létre lett hozva egy egységes rendezést biztosító komponens, amely a backend-en belül a felületről érkező kéréseknek megfelelően felépíti az összevont lekérdezéseket.

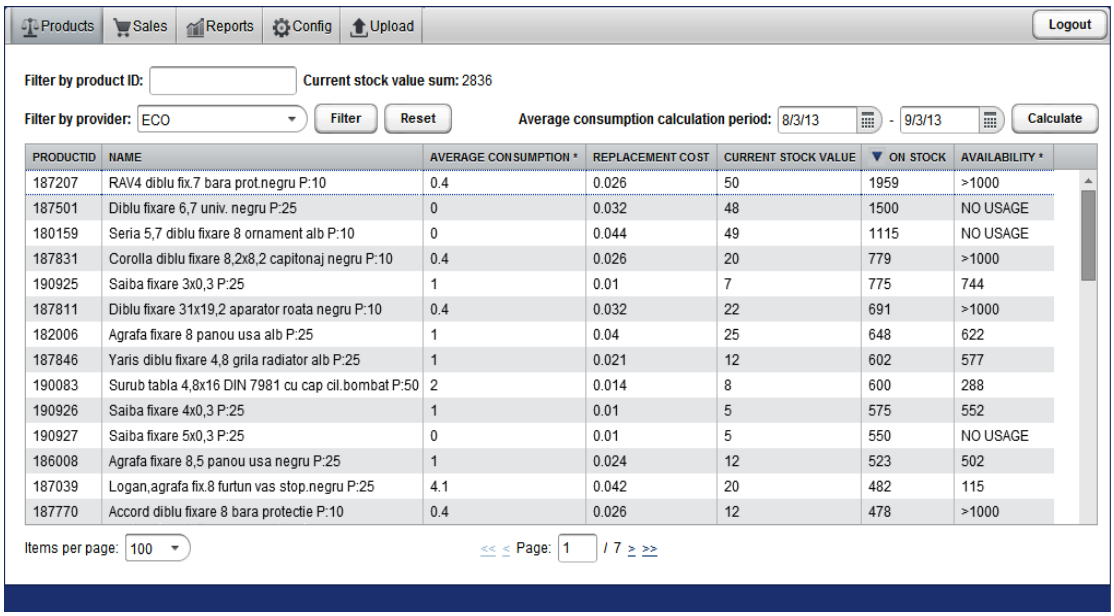
4. Az OptInv prototípus működése

A webes felületen kis fülek képezik az alkalmazás menürendszerét. A nézetek a felhasználó szerepkörének megfelelően jelennek meg.

Az adminisztrátor bejelentkezés után a szerepkörének megfelelő felületet láthatja, ahol a cégek profiljait és a cégekhez tartozó felhasználókat menedzselheti (4. ábra). A Bean Validation mechanizmusnak köszönhetően az egyes mezők esetében a megfelelő hibaüzenettel jelezve van, ha egy begépelt adat nem helyes.

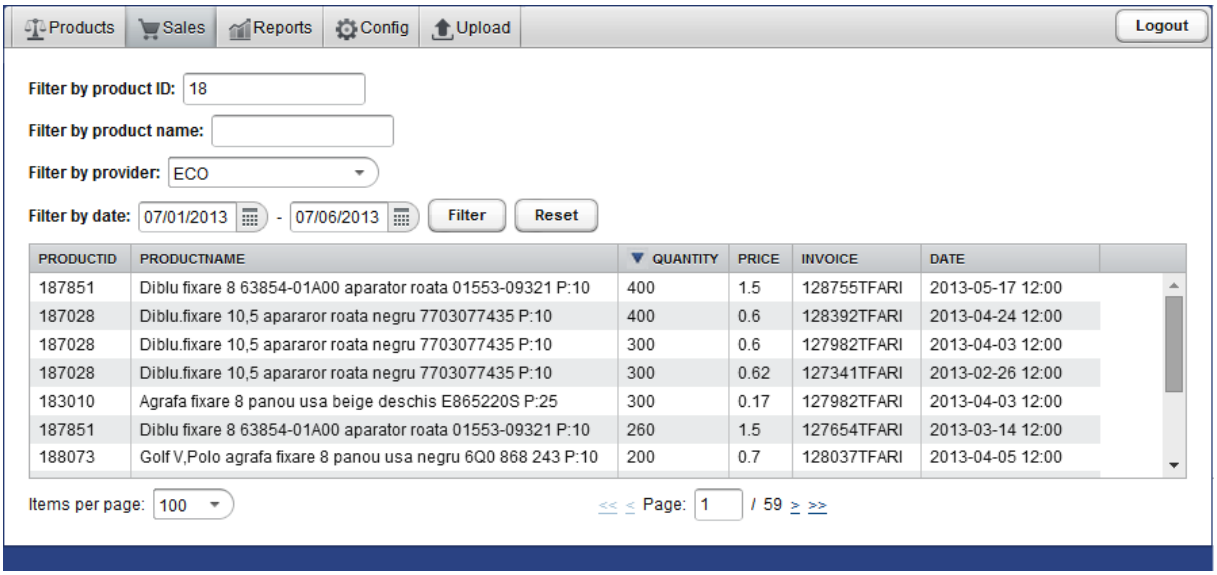


4. ábra: Az OptInv rendszer adminisztrációs felülete



5. ábra: A termékekkel kapcsolatos információk

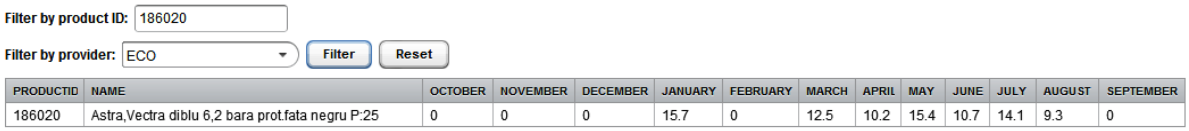
Ha egy céghez tartozó felhasználó jelentkezik be a rendszerbe, először a termékek adatait megjelenítő felületet láthatja (5. ábra). Amennyiben a felhasználó duplán kattint egy termékre, egy külön ablakban megjelennek a beszállítókkal kapcsolatos információk. A felhasználó szűrheti a termékeket kód, illetve beszállító alapján, valamint két naptár segítségével kiválaszthatja azt a periódust, amelyre az alkalmazás kiszámolja a táblázatban található számadatokat.



6. ábra: Eladásokat megjelenítő nézet szűrési és keresési lehetőségekkel

A 5. ábrán a termékek ECO beszállítóra vannak szűrve, az ettől a beszállítótól származó aktuálisan raktáron levő termékek összértéke 2836 RON. A termékekhez kapcsolódó számadatok a 2013.08.03.- 2013.09.03. közti periódusban történt eladások alapján vannak számolva, a táblázat pedig a raktáron levő mennyiség szerint van rendezve, csökkenő sorrendben.

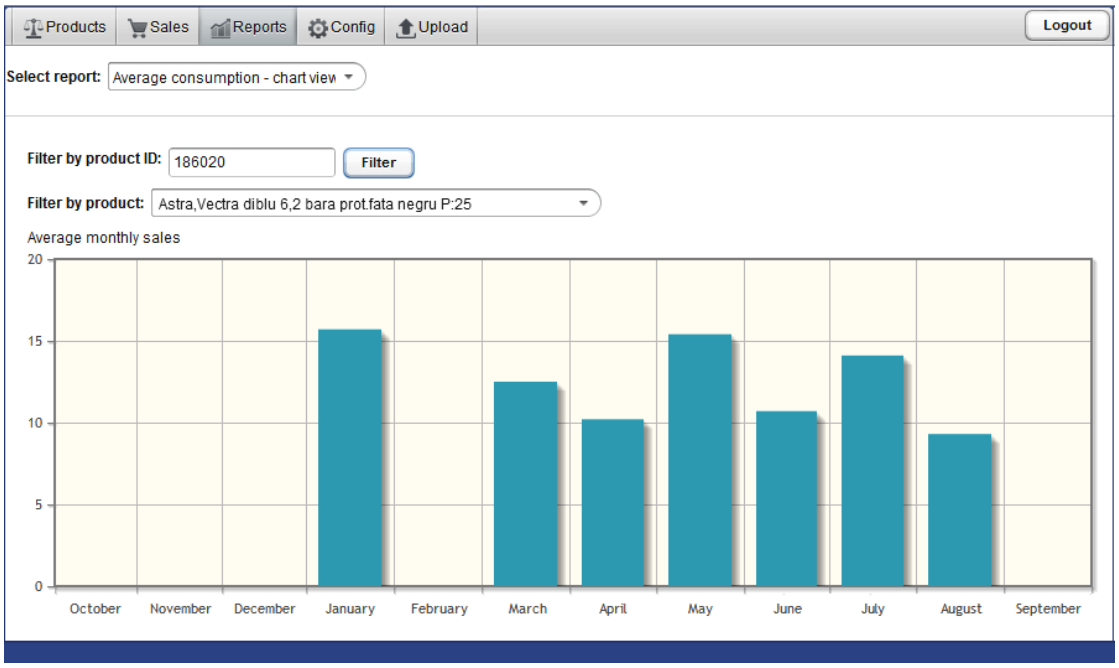
Látható, hogy például az 190925-ös kódú termékből 1 darab a napi átlag eladás a kijelölt hónapban és 775 darab áll raktáron, így ez a mennyiség még körülbelül 744 napra elegendő. Olyan termékek is vannak, amelyekből még több mint 1000 napra elegendő a készlet, sőt olyanok is, amelyekből egyáltalán nem történt eladás, mégis nagy mennyiség áll raktáron fölöslegesen.



PRODUCTID	NAME	OCTOBER	NOVEMBER	DECEMBER	JANUARY	FEBRUARY	MARCH	APRIL	MAY	JUNE	JULY	AUGUST	SEPTEMBER
186020	Astra,Vectra diblu 6,2 bara prot.fata negru P:25	0	0	0	15.7	0	12.5	10.2	15.4	10.7	14.1	9.3	0

7. ábra: Jelentések megjelenítése táblázatos formában

A Sales fülön (6. ábra) tekinthetőek meg az eladásokkal kapcsolatos információk. Az adatok rendezhetőek minden oszlop szerint, valamint szűrhetőek a termék kódja, neve, annak beszállítója, illetve egy periódus alapján. A szűrések akár kombinálhatóak is.



8. ábra: Jelentések megjelenítése grafikon segítségével

Egy külön fülön láthatóak a jelentések: a termékek átlag eladása havi lebontásban, illetve adott termék esetében grafikonon. Lehetőség szűrésre, a táblázat esetében termék kód, valamint beszállító alapján, grafikonos megjelenítésnél kód, illetve név alapján.

A 7. ábra az 186020 kódú termék eladási adatait mutatja az utolsó eladás dátumától visszamenőleg egy évre (a minta adatbázis nem tartalmazott 2012-es adatokat az első hónapok értéke ezért 0). A 8. ábrán ugyanezek az adatok tekinthetőek meg grafikonon.

Az adatok importálása a Config fülön belül tehető meg, ahol a felületen megjelenik az adatbázis dátuma is, azaz az utolsó eladás időpontja. Az importáláshoz szükséges állományok feltöltését az Upload fül teszi lehetővé, amely jelzi is a felhasználóknak, hogy a cég adatbázistípusának megfelelően milyen állományok feltöltésére van szükség. A drag and drop mechanizmusnak köszönhetően felhasználó a fájlkezelő rendszeréből egyszerűen behúzhatja az állományokat a felületre.

Következtetések és továbbfejlesztési lehetőségek

Az OptInv projekt jelenlegi formájában egy prototípus, de már most is segítségére lehet a vállalkozásoknak készleteik optimalizálásában. A fejlesztés során a tesztadatbázist szolgáltató partnercég is folyamatosan ellenőrizte az adatokat, és tesztelte az alkalmazást. A visszajelzések alapján a projekt jelen állapotában is használható, sok olyan információt nyújt, amelyekre egy átlagos könyvelési program nem képes. A továbbfejlesztett verziók további adatokat, illetve funkcionalitásokat nyújthatnak. Néhány továbbfejlesztési lehetőség:

- további könyvelési programok adatbázisaiból való adatimportálás támogatása;
- automatizált importálás meghatározott időpontokban;
- rendeléseket nyilvántartó modul integrációja;
- mobilalkalmazás megvalósítása, automatikus értesítések küldése fontosabb készletváltozások esetén;
- további statisztikai számítások és jelentések;
- termékek automatikus kategorizálása, előrejelzések (klasszikus algoritmusok és új módszerek – pl. neurális hálózatok – alkalmazása);
- rendelések optimalizálása (pl. javaslatok termékek csoportosítására rendelések esetében).

Hivatkozások

[1] Scrum Hivatalos Weboldal, https://www.scrum.org/Portals/0/Documents/Scrum%20-Guides/2013/Scrum-Guide.pdf, utolsó megtekintés dátuma: 2014.04.13.

[2] Mercurial Hivatalos Weboldal, http://mercurial.selenic.com/, utolsó megtekintés dátuma: 2014.04.13.

[3] Tim O'Brien, Manfred Moser, John Casey, Brian Fox, Jason Van Zyl, Eric Redmond, Larry Shatzer, Maven: The Complete Reference, 2010

[4] Martin Fowler, Continuous Integration, http://www.martinfowler.com/articles/-continuousIntegration.html, utolsó megtekintés dátuma: 2014.04.12.

[5] Arun Gupta, Java EE 7 Essentials, O'Reilly Media, 2013

[6] Java Enterprise Edition Hivatalos Dokumentáció, http://docs.oracle.com/javaee/7/-tutorial/doc, utolsó megtekintés dátuma: 2014.04.18.

[7] Richard Monson-Haefel, Bill Burke, Enterprise JavaBeans 3.0 5th Edition, O'Reilly Media 2006

[8] EclipseLink Wiki, http://wiki.eclipse.org/EclipseLink/Development/Indigo/-Multi-Tenancy, utolsó megtekintés dátuma: 2014.04.12.

[9] Shaun Smith, Multi-Tenancy & Extensibility előadása, http://devoxx.com/-pages/viewpage.action?pageId=5341639, utolsó megtekintés dátuma: 2014.04.12.

[10] Marko Grönroos, Book of Vaadin: Vaadin 7 Edition, 2013