Recent Innovations in Mechatronics (RIiM) Vol. 5. (2018). No. SI.
DOI: 10.17667/riim.2018.si/30.

Árubeszerzési folyamatot segítő vállalatirányítási rendszer Spring keretrendszer használatával

Klein Ádám
Mérnökinformatikus
Debreceni Egyetem, Informatikai Kar
Debrecen, Magyarország
kleinadam96@hotmail.com

Beatrix Papp
London South Bank University
School of Law and Social Sciences,
London, United Kingdom
pappb@lsbu.ac.uk

Erdei Timotei István
Mechatronikai Tanszék
Debreceni Egyetem, Műszaki Kar
Debrecen, Magyarország
timoteierdei@eng.unideb.hu

***Absztrakt*— Kis- és nagyvállalatok mindennapi folyamatainak egyaránt fontos részét képezi az árubeszerzés. Egy vállalat fejlődése során általában az import folyamatok mennyisége exponenciálisan növekszik a vállalat terjedésével egyhuzamban. Ennek következtében a hagyományos telefon és e-mail alapú szervezés egyre elavultabbá válik. A kommunikáció megnehezedik a felek között, az információ áramlása a köztes csatornákon nem megfelelő minőségű, fennakadásokhoz vezet. A nem megfelelő import a vállalat gyártási és működési folyamatait akadályozhatja, ezzel direkt és indirekt módon is kárt és veszteséget okozva a vállalatnak. Ezek következtében felmerül az igény egy olyan vállalatirányítási rendszerre, amely biztosítja a megfelelő kommunikációt a felek között, biztosítja az import folyamatok nyomon követhetőségét és biztosítja az ehhez tartozó adatok integritását. A projekt célja egy árubeszerzési folyamatot segítő vállalatirányítási rendszer létrehozása volt, nyílt keretrendszer használatával.**

Kulcsszavak—Spring, Maven, Thymeleaf, MySQL, Java, JavaScript

I. BEVEZETŐ

A vállalatirányítási rendszereknek komplex problémák megoldásaival kell megbirkózniuk, biztosítaniuk kell a folyamatos elérhetőséget, jogköröket kell definiálniuk a biztonság érdekében, magas absztrakciót kell megvalósítaniuk és könnyen átláthatónak, használhatónak kell lenniük.

Egy ilyen alkalmazás megvalósításához tökéletes alapot nyújt a Spring keretrendszer [1], amely platform független, nyílt szabványú, Java [2] nyelven íródott keretrendszer.

A projekt ezt a keretrendszert használja, adatbázis szinten MySQL [3] relációs adatbázissal dolgozik, a front-end pedig a Thymeleaf [4] szerver oldali „*Template engine*” segítségével generálódik.

A rendszer egy alapvető import folyamatot kezel és biztosítja hozzá a kommunikációs csatornát. Autentikáció nélkül nem használható az applikáció, mivel három jogosultságot kezel.

Ezek az alábbiak:
- Adminisztrátor
- Beszerző
- Import-fuvarszervező.

Az applikáció lehetővé teszi az egyes jogköröknek megfelelő műveletek végrehajtását, majd ezeknek a műveleteknek az ellenőrzését, nyomon követését. A kutatás/fejlesztésnek a Debreceni Egyetem adott otthont [14].

II. A RENDSZER SZERVER OLDALA

A Spring keretrendszer egy jelentős nyílt forráskódú Java/J2EE alkalmazás fejlesztői keretrendszer. A legnépszerűbb keretrendszernek számít 30%-os használati mutatóval [1].

A Spring komponensei lehetővé teszik az egyszerű webalkalmazások fejlesztésétől kezdve a komplex Enterprise applikációk fejlesztését is.

A fő koncepciók, amit a Spring keretrendszer követ a következők: a kontroll megfordítása (IoC), a függőség befecskendezése (DI), aspektorientált programozás (AOP), Java perzisztencia (JPA) [5].

Az alkalmazás a Spring egy modulját, a Spring Boot-ot használja. A Spring Boot koncepciója az automatikus konfiguráció, ami lehetővé teszi a standalone alkalmazások fejlesztését, amiket konfiguráció nélkül lehet „csak futtatni”. Tartalmaz egy servlet container-t, (ami legtöbbször Apache Tomcat [6]), automatikusan végrehajtja a futtatás build fázisát, majd az így létrejött snapshot-ot konfigurálva futtatja azt a servlet container-ben, így a fejlesztőnek gombnyomásra létrejön a futatni kívánt alkalmazás [7].

Az adatokat MySQL relációs adatbázisban kerülnek tárolásra. Az adat relációs tekintetben: S_1, S_2, ... S_n (nem feltétlenül különböző) tömb, R egy reláció ezen az n darab tömbön, ha az egy n elemű rendezett tömbökből álló tömb, amik mindegyikének első eleme S_1-ből származik, második eleme S_2-ből és így folytatva tovább. S_j-re hivatkozhatunk, mint j-edik tartományára az R relációnak. Ebből kifolyólag az R reláció n fokú. Elsőfokú relációkat unáris, másodfokú

relációkat bináris, harmadfokú relációkat ternáris kifejezésekkel illetjük, magasabb fokon pedig *n*-ed fokúként hivatkozunk rájuk [8].

A Spring keretrendszer az adatbázissal előre definiált modelleket használva a Spring Data JPA-t használva kommunikál.

A modelt POJO-kként (Plain Old Java Object) definiáljuk, ezek lesznek leképezve az adatbázisba. Az adatbázisból elérni ezeket több féle módon van lehetősége a fejlesztőnek.

Egy ilyen módszer a natív SQL scriptek írása, amiket a *@Query* annotációval kell ellátni. Mivel natív nyelvről van szó, ez teljesen adatbázis függő, a lekérdezéseket annak a függvényében kell megírni, hogy éppen milyen adatbázist használ az alkalmazás.

Továbbá a fejlesztőnek lehetősége van ezekben az interfacekben saját metódusok definiálására. Ezeket a metódusokat nem kell implementálni, mivel a Spring Data JPA futási időben implementálja őket a szignatúrájuk alapján. Több ilyen interface létezik, ezeket a *@Repository* annotációval látjuk el. Egy példa egy ilyen repositoryra az applikációban látható az 1. ábrán.

```
package repositories;

import org.springframework.data.domain.Page;

@Repository
public interface FreightOrderRepository extends PagingAndSortingRepository<FreightOrder, Long>{
    public Page<FreightOrder> findAllByStatusOrderByArrivalDateAscIdDesc(Pageable pageable, OrderStatus status);
    public Page<FreightOrder> findAllByUserAndStatus(Pageable pageable, User user, OrderStatus status);
    public Page<FreightOrder> findAllByCoordinatorAndStatus(Pageable pageable, User user, OrderStatus status);
    public Page<FreightOrder> findAllByUser(Pageable pageable, User user);
    public Page<FreightOrder> findAllByCoordinator(Pageable pageable, User user);
}
```

1. ábra PagingAndSortingRepository definiálása az alkalmazásban

Ugyan a Spring Data JPA repositoryk használatával és egy adatbázis connector függőség megadásával sikeresen tudjuk lekérdezni az adatbázist, de a Spring Data JPA csak egy specifikációt biztosít az objektum-relációs leképezésekhez. Mint korábban említve volt, a Spring POJO-k (Plain Old Java Object) használatával tudja kezelni az adatokat, ezek viszont az objektum orientált Java nyelvből kifolyólag olyan objektumok, amik inkompatibilisek az adatbázisban lévő rekordokkal.

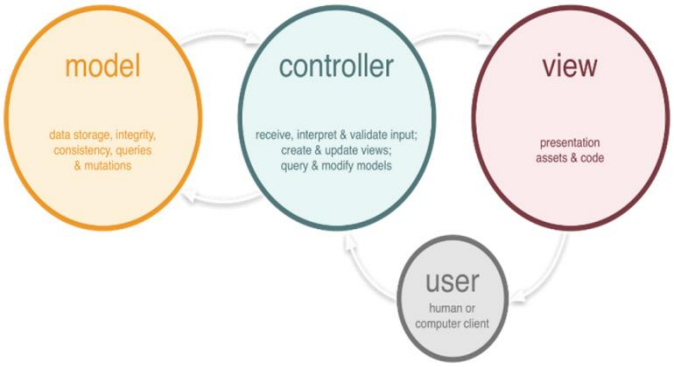
Ehhez szükséges egy objektum-relációs leképezéseket implementáló eszköz, a Hibernate ORM-re [9].

A Hibernate segítségével Java objektumokat annotálva tudunk adatbázis entitásokat létrehozni. Egy entitást minden esetben az *@Entity* annotációval kell ellátni, ami egy osztályszintű annotáció.

A mezőket rengeteg többféle annotációval kell ellátni, *@Id, @Basic, @Temporal, @Version, @Transient* stb. Ezek fogják jelenteni egy-egy mező adatbázisba való leképezésének tulajdonságait, típusát, a kényszereket (constraint).

Továbbá a Hibernate ORM biztosítja az entitások közötti kapcsolatok definiálását. Ezeket is annotációkkal kell megvalósítani, az alábbi annotációk felhasználásával, *@OneToOne*, *@OneToMany*, *@ManyToOne*, *@ManyToMany*. Opcionálisan meg lehet adni egy két adatbázis entitás közötti kapcsolat típusát, ami lehet egyirányú vagy kétirányú.

Az alkalmazás (Modell-Nézet-Vezérlő) szerkezeti mintát alkalmaz. Az MNV szerkezetet először Trygve Reenskaug fogalmazta meg 1970-ben. Szerinte „az MNV célja az, hogy áthidalja a rést a felhasználó mentális modellje és a digitális modell között ami a számítógépen létezik”, ahogyan a 2. ábrán látható [10].



2. ábra MNV (MVC) tervezési minta [10]

A „Modell” a rendszernek azon része, amely ellátja mindazon feladatokat, amik az adatokhoz kapcsolódnak (érvényesítés, hitelesítés, adatforrás strukturálás).

Egy modellt általában az adatabsztrakció, érvényesítés és hitelesítés szempontok alapján határoznak meg. A modell tartalmazza azokat a metódusokat, amik az adatforrásokkal való munkát végzik (módosítás, törlés stb.)

A „View” felelős a grafikus felhasználói interfész megjelenítéséért. Meghatározza, hogy hogyan jelenik meg az adat a felhasználó számára. A felhasználótól való adatbegyűjtést is biztosítja.

A „Controller” kezeli azokat az eseményeket, amik kiváltódhatnak annak következtében, hogy egy user vagy egy rendszerfolyamat kommunikál az applikációval (kérések). A vezérlő elfogadja a kéréseket, előkészíti az adatokat és meghatározza a formátumot a válaszhoz.

A projekt-ben létrehozott alkalmazásban az MNV architektúrát [11] a Spring MVC modul kezeli. A Spring MVC a *DispatcherServlet* servletre épül, ami a HTTP kéréseket a *@Controller* annotációval ellátott vezérlő metódusok egyikéhez adja át, még pedig ahhoz, amelyik az URL és HTTP metódusnak megfelelő *@RequestMapping* annotációval van ellátva, ahogyan a 3. ábrán látható.

```
@RequestMapping(value="/available", method = RequestMethod.GET)
public String getAvailableOrders(Model model, HttpServletRequest req, @RequestParam("page") Optional<Integer
    if (!page.isPresent()) return "redirect:/available?page=1";
    Page<FreightOrder> elements = fos.getAllByStatus(new PageRequest(page.get()-1, 9), OrderStatus.AVAILABL
    model.addAttribute("orders", elements);
    model.addAttribute("totalPages", elements.getTotalPages());
    model.addAttribute("currentPage", page.get());
    return "available";
}

@RequestMapping(value="/accept/{id}", method = RequestMethod.GET)
public String getAcceptOrder(@PathVariable("id") Long id, Model model, HttpServletRequest req){
    if(fos.getOrderById(id).getStatus()!=OrderStatus.AVAILABLE){
        return "redirect:/error";
    }
    model.addAttribute("order", fos.getOrderById(id));
    return "accept";
}
@RequestMapping(value="/accept/{id}", method = RequestMethod.POST)
public String postAcceptOrder(@PathVariable Long id, final HttpServletRequest req) {
    FreightOrder order = fos.getOrderById(id);
    if(order.getStatus()!=OrderStatus.AVAILABLE) {
        return "redirect:/error";
    }
    String[] name;
    String[] cost;
    String[] pickupDate;
    String[] arrivalDate;
    String[] isDefault;

    for(int i=0; i<order.getShipmentTypes().size(); i++) {
```

3. ábra Vezérlő metódusok az alkalmazásban

Az applikáció a REST (Representational State Transfer) architekturális stílust követi, ami azt jelenti, hogy az applikáció RESTful web szolgáltatásokat tartalmaz, ahol minden szolgáltatás egy vagy több CRUD műveletet (létrehozás, olvasás, frissítés, törlés) végez el az adott erőforrásokon [12].

Az URL-eket dinamikusan kezeli, ezekből és a hozzájuk tartozó HTTP metódusokból határozza meg a „vezérlő”, hogy milyen műveletet kell elvégezni. A REST architektúrából kifolyólag az applikáció állapot független, ami azt jelenti, hogy egy-egy válasz a szerver felől nem függ semmilyen állapottól, csak az adott kérés URL-jétől.

Az 1. táblázatban látható a RESTful routing működésére.

1. TÁBLÁZAT RESTFUL ROUTING [13]

URL	HTTP metódus	Válasz
/	GET	index megjelenítése
/new	GET	új űrlap megjelenítése
/new	POST	új megrendelés létrehozása
/login	GET	login oldal megjelenítése
/login	POST	felhasználó autentikálása
/view/{id}	GET	az {id} azonosítójú rendelés megjelenítése
/confirm/{id}	POST	az {id} azonosítójú rendelés megerősítése

Az alkalmazás front-end része az RWD megközelítést alkalmazza, vagyis az oldal optimálisan jelenik meg a különböző méretű és felbontású eszközökön, ezzel könnyebb átláthatóságot és navigációt biztosít a felhasználók számára [13].

Ennek elérésére egy front end komponens könyvtár került használatra, a Bootstrap-et. Egy JavaScript fájl kezeli a „view” elrendezését, reagálva a különböző eseményekre, annak az érdekében, hogy a felhasználó minden platformon és ablak méretben ugyanazt az egyszerűen átlátható, követhető grafikus felületet kapja.

III. A FOLYAMAT LEÍRÁSA

Az alap koncepció a rendszer tervezésekor az alábbi összetevőkből állt össze:

- Az alkalmazásnak biztosítania kell a megfelelő jogköröket és védelmet a funkciók számára.
- Az alkalmazásban létrejött adatoknak nyomon követhetőnek kell lennie.
- Az applikációban az információ áramlásának gyorsnak kell lennie.
- Nagy mennyiségű adat feldolgozásának hatékony megkövetelése.

Mint korábban említve volt, a rendszerben három jogkörrel rendelkező felhasználó létezik: adminisztrátor, beszerző és import-fuvarszervező.

A munkafolyamat leírása a következő:

Egy beszerző a számára rendelkezésre álló adatok alapján kitölt egy űrlapot, ami az import megrendelés adatait tartalmazza. Egy ilyen megrendelésnek négy állapota van, amit Java enumerációkként tárolunk.

Ezt követően létrejön a megrendelés és beleíródik az adatbázisba elérhető státusszal.

Amíg a megrendelés ebben a státuszban van, a beszerzőnek nincs lehetősége módosító műveletet végrehajtania a megrendelésen.

Az elérhető megrendelések a fuvarszervezők számára jelennek meg egy menüpontban, ahol röviden összefoglalva csak a legfontosabb adatok láthatóak. Ebből a listából szabadon lehet választaniuk a megrendelőknek, azonban egy rendeléshez egyedül egy fuvarszervező tartozhat, mint koordinátor.

Amikor egy fuvarszervező befejezi az aktuális elérhető megrendelés további adatainak kitöltését és beküldi az űrlapot, ezzel „elfogadva” a rendelést, eltűnik az elérhető elemek listájából és módosul a státusza elfogadottá.

Ilyenkor a rendelés folytatását az azt létrehozó beszerzőnek kell folytatnia, a megadott adatok alapján ki kell választania egy beszállítót és meg kell erősítenie a rendelést.

Amíg ez a folyamat nem fejeződik be, a koordináló fuvarszervező csupán láthatja a megrendelés adatait, nem hajthat rajta végre más műveletet.

A megerősítést követően egy újabb lépés következik, ami ismét a koordinátor feladata, vagyis a megrendelés teljesítése, mégpedig, úgy hogy egy rendszámmal vagy valamilyen megrendelési azonosítóval látja el a beérkezett rendelési igényt, így lezárva a folyamatot. Ezek után a megrendelésen semmilyen változtatás nem léphet életbe, az adatbázisban tárolódik és igény szerint visszakereshető, megjeleníthető a komplett tartalma.

Összefoglalva a megrendelések attól a ponttól kezdve, hogy létrejönnek, módosíthatóak és nyomon követhetőek, egészen a teljesítésig, ami után csak lekérdezhetőek. Ezek az

entitások az adatbázisban az 4. ábrán látható módon jelennek meg.

id	user_id	coordinator_id	arrival_date	contact_email	contact_name	contact_phone	status
1	2	3	2018-04-08				3
2	2	3	2018-04-08	something@localhost.com	Name	555-312-5234	3
3	2	9	2018-04-21	something@localhost.com	Name	555-312-5234	3
4	5	3	2018-04-21	something@localhost.com	Name	555-312-5234	1
5	2	3	2018-04-10	something@localhost.com	Name	555-312-5234	3
26	2	3	2018-04-14	something@localhost.com	Name	555-312-5234	3
27	2	9	2018-04-21	contact@localhost.com	new name	555-312-5234	3
28	2	3	2018-04-20	something@localhost.com	Name	555-312-5234	2
29	2	NULL	2018-04-21	something@localhost.com	Name	555-312-5234	0
30	2	NULL	2018-04-21	something@localhost.com	Name	555-312-5234	0
31	2	NULL	2018-04-21	something@localhost.com	Name	555-312-5234	0
32	2	NULL	2018-04-21	something@localhost.com	Name	555-312-5234	0
33	2	NULL	2018-04-21	something@localhost.com	Name	555-312-5234	0
34	2	NULL	2018-04-21	something@localhost.com	Name	555-312-5234	0
35	2	NULL	2018-04-21	something@localhost.com	Name	555-312-5234	0
36	2	NULL	2018-04-21	something@localhost.com	Name	555-312-5234	0

4. ábra A megrendelések rekordjai az adatbázisban

IV. A FUNKCIÓK BEMUTATÁSA

A felhasználó autentikáció nélkül két oldalt érhet el, a bejelentkező és a regisztráció felületet.

Az oldalak védettségét és az autentikációt a Spring Security modul biztosítja, amivel lehetőség van user entitásokat létrehozni, azokhoz tartozó jogköröket definiálni, ezeknek megfelelően az oldalakat levédeni, illetve sikeres bejelentkezés esetén session-t létrehozni, kijelentkezés esetén pedig invalidálni a session-t.

A „user” modell egy azonosító kulcsból áll, egy e-mail címből, jelszóból, keresztnévből és vezetéknévből, illetve egy kapcsolatból, ami a user-hez tartozó jogkört definiálja. A felhasználó jelszava biztonsági okokból hash-elve tárolódik az adatbázisban.

A Spring Security beállítását egy konfigurációs fájlban lehet megadni, ami a *WebSecurityConfigurerAdapter* osztály egy leszármazott osztálya kell legyen és a megadott metódusok felülírásával lehet egyedi konfigurációt létrehozni, ahogyan a 5. ábrán látható.

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests()
            .antMatchers("/new/**", "/confirm/**").hasAuthority("PURCHASER")
            .antMatchers("/available/**", "/accept/**", "/complete/**").hasAuthority("COORDINATOR")
            .antMatchers("/",  "/orders/**", "/view/**").authenticated()
            .anyRequest().authenticated()
            .antMatchers("/resources/**", "/login", "/signup").permitAll().anyRequest().permitAll()
            .and()
            .exceptionHandling().accessDeniedPage("/accessDenied")
            .and()
            .formLogin()
            .loginPage("/login")
            .failureUrl("/login?error")
            .usernameParameter("email")
            .permitAll()
            .and()
            .logout()
            .logoutUrl("/logout")
            .deleteCookies("remember-me")
            .logoutSuccessUrl("/login?logoutSuccess")
            .permitAll()
            .and()
            .rememberMe()
            .key("keyboard cat");
}
```

5. ábra A SecurityConfig osztály, ami a konfigurációt tartalmazza

Az alkalmazás HTTP szintű biztonságot alkalmaz egyedi hitelesítéssel. Sikeres autentikáció után a felhasználó különböző funkciókat érhet el a jogkörének megfelelően.

A beszerzői jogkörrel rendelkezdő felhasználók számára a dashboard-on kívül két menüpont érhető el. Az első az új megrendelés létrehozása, ami egy űrlapot jelenít meg, ennek kitöltésével hozható létre az új megrendelés.

Ez az űrlap tartalmaz egyszerű szöveg és szám alapú inputokat, selectek-et, checkboxok-at, radio gombokat, dátum típusú inputot, illetve dinamikusan hozzáadható és törölhető inputokat.

A dinamikus input gombnyomásra JavaScript selector-ral kiválasztja azt az elemet, amihez hozzá kívánjuk adni az újabb sort, majd ugyanezt az elemet hozzáfűzi a már meglévőekhez, illetve gombnyomásra egy event listen-er használatával törlés opciója is elérhető a felhasználónak.

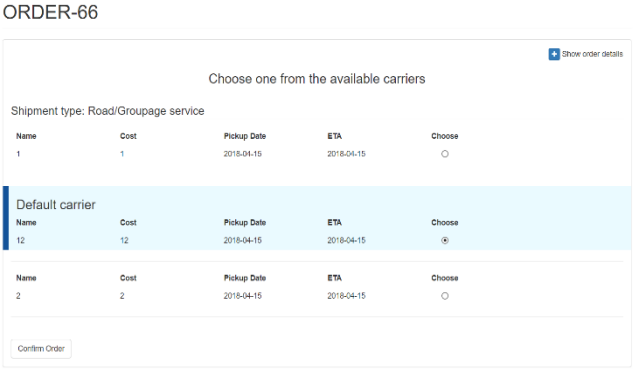
Ezen kívül lehetősége van nyomon követni az általa kiírt rendeléseket egy másik menüpontban, megjeleníteni azokat, illetve státuszukhoz megfelelő műveletet végrehajtani.

Ebben a menüpontban a megrendelések a felhasználói átláthatóság érdekében a leglényegesebb attribútumaikkal jelennek csak meg panelekben, amelyeknek a színe attól függően változik, hogy mennyire van közel az adott megrendelés teljesítésének a határideje.

Amennyiben egy megrendelés teljesítési határideje az aktuális időponttól számítva 1 héten kívül esik, a megrendelés egy zöld panelben jelenik meg. Az aktuális időponttól számítva a 3-7 napon belül teljesítendő megrendelések sárga panelben jelennek meg, a sürgős, 3 napon belüli vagy lejártrendelések pedig piros panelekben látszódnak, így a felhasználó első ránézésre is könnyen megállapíthatja, melyek azok a rendelések amelyek kiemelt figyelmet igényelnek.

A teljesített megrendelések sötétkék panelekben jelennek meg, a szervezést nem igénylő, csak árajánlatot kérő rendelések pedig halványkékben. Az alkalmazásban ez a nézet az oldal renderelé-se után, JavaScript DOM manipulációval lett megvalósítva.

Amennyiben egy rendelés megerősítésre vár, a panelekből, vagy a /confirm/{id} routeon keresztül a megrendelés adatait megjelenítve a beszerzőnek az elérhető ajánlatokból meg kell erősítenie egyet. Lehetőség van a fuvarszervező számára default vagy alapvető ajánlatot definiálni, ez kiemelve jelenik



meg a beszerző számára, ahogyan a 6. ábrán is látható.

6. ábra Egy megrendelés megerősítése

A koordinátorok vagy fuvarszervezők számára autentikáció után szintén 2 menüpont érhető el.

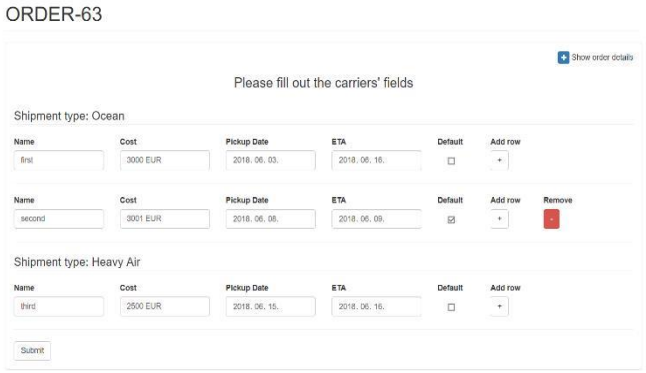
Az első menüpontban az elérhető státuszú rendelések jelennek meg szintén változó színű panelekben. Mint

korábban volt említve, egy rendeléshez egy koordinátor tartozhat minden esetben, így elfogadásra csak a legelsőnek beküldő koordinátornak lehet lehetősége.

Az elfogadni kívánt megrendelés az /accept/{id} routeon jelenik meg. A létrehozó beszerző által kitöltött információk megjelenítése után a szállítási típusokra a fuvarszervezőnek minden esetben legalább 1 beszállítói opciót kell adnia, illetve lehetősége van alapértelmezett beszállító megadására is, ami később kiemelve jelenik meg a beszerzőnek. Ezt a 7. ábra szemlélteti.

Ezek után az adott megrendelés nem elérhető a fuvarszervező számára, csupán megtekinthető, egészen addig, amíg a megadott beszállítók közül a rendelést létrehozó beszerző nem választja ki a megfelelőt.

Továbbá a fuvarszervező feladatai közé tartozik a megrendelés lezárása, amit a fuvar leszervezése után kapott beszállító által megadott rendszám vagy valamilyen azonosító megadásával tud végrehajtani. Ez után a pont után a megrendelés lezártnak minősül, a létrehozó beszerzőnek, a koordináló fuvarszervezőnek és az adminisztrátornak lehetősége van megtekinteni a teljesen kitöltött űrlapot.



7. ábra Egy megrendelés elfogadása

A rendeléseket, és minden más adatbázis entitást egy azonosító kulcs mezővel látunk el, aminek a maximum értéke 2^{64} -1 lehet, így ennyi darabkülönböző rekord is létezhet az adatbázisunkban entitásonként. Skálázhatóság szempontjából ez optimális, de ekkora méretű adatbázist már csak hatalmas számítási kapacitással lehet elfogadhatóan kezelni.

A rendszer alapját képezik az űrlapok, amelyek különböző státuszhoz kapcsolódva különböző formában jelennek meg. Ezeknek az űrlapoknak a tartalma alapján, és egyéb adatok alapján generálódnak és módosulnak az entitások, amik az adatbázisban tárolódnak.

Ebből kifolyólag rendkívül fontos a validáció, azaz a megfelelő inputok vizsgálata. Az űrlapokat csak az inputoknak megfelelő típusú értékekkel lehet elküldeni, a kötelezően kitöltendő mezőket szintén vizsgáljuk.

Mivel ezek az űrlapok, és az egész grafikus felület böngészőben található eszközökkel könnyen manipulálható, illetve http kérések küldésére használható böngésző kiegészítők segítségével megkerülhető, nem elég egyszerűen a front enden vizsgálni az adatokat, fontos részét képezi az alkalmazásnak a szerver oldali adatok hitelesítése is.

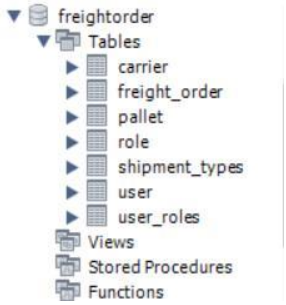
Egy ilyen példa az űrlapok kitöltésénél használt CSRF (Cross-Site Request Forgery) tokenek használata, így megvédve az autentikált felhasználókat a rosszindulatú szoftverek ellen, oly módon, hogy ellehetetleníti a nem böngészőből kitöltött űrlapok beküldését.

Valamint minden módosító műveletnél a vezérlő metódusokban egy további hitelesítés előzi meg az adatbázissal való kommunikációt, ezzel is biztonságosabbá téve az alkalmazást.

V. AZ ADATBÁZIS FELÉPÍTÉSE ÉS A TÁBLÁK KÖZÖTTI KAPCSOLATOK

A rendszer ismertetése után az adatbázisban létrejött táblák és rekordok bemutatása is kiemelten fontos. Az alkalmazás tetszőleges MySQL sémára futtatható, az *application.properties* fájlban szükséges megadni a JDBC kapcsolatra vonatkozó adatokat, ezek az adatbázist elérő felhasználó felhasználóneve, jelszava, illetve a MySQL séma URI-ja. Továbbá a Hibernate adatbázis inicializációt is tud végezni, ennek a típusát is meg lehet adni opcionálisan.

Az alkalmazás adatbázisának felépítése és a benne lévő táblák a 8. ábrán láthatóak.



8. ábra Az adatbázis

Az applikáció a rendeléseket a *freight_order* táblában tárolja rekordokként. Mint korábban említve lett, ez tartalmaz BIGINT, VARCHAR, INT, DATE, BIT típusú oszlopokat. A Hibernate ORM segítségével ez a tábla több másik táblával is kapcsolatban áll, ezek a következők:

A *pallet* táblával egy a többhöz kapcsolat áll fent, ahol a rendelés a kapcsolat tulajdonosa, a *pallet* táblában vannak definiálva a rendeléshez tartozó raklapok adatai.

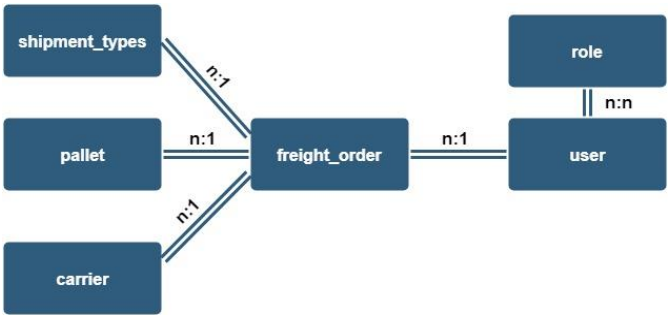
A *shipment_types* táblával szintén egy a többhöz kapcsolat áll fent, ahol szintén a rendelés a kapcsolat tulajdonosa, ebben a táblában a rendeléshez tartozó lehetséges szállítási típusok jelennek meg, ez egy egyszerű elem kollekcióként van definiálva a rendelés Java osztályában.

A *carrier* táblával ugyanilyen egy a többhöz, tulajdonosi viszonnyal rendelkezik, itt a beszállítók adatai vannak tárolva.

A *user* táblával több az egyhez való kapcsolat áll fent, ebben az esetben két mezőnél is létezik ez a kapcsolat, a létrehozó user és a koordinátor user is egy-egy mezőként van jelen a rendelés entitásban.

Továbbá a *user* és a *role* tábla között kétirányú több a többhöz kapcsolat áll fent, ezeket egy külső, harmadik táblában a *user_roles* táblában tároljuk, ahol a userre vonatkozó jogkörök vannak definiálva.

Az adatbázis felépítése és a kapcsolatok a 9. ábrán vannak szemléltetve.



9. ábra A táblák közötti kapcsolatok

VI. Összegzés

Összességében a projekt során sikerült egy standalone vállalatirányítási rendszert megalkotni, ami hatékonyan és átláthatóan működik. A strukturált felépítésnek köszönhetően, könnyen módosítható/átszervezhető, ezzel biztosítva nagyobb flexibilitást, az adott vállalat számára. A felépítésénél fő szempont volt, hogy könnyen illeszthető legyen más vállalatok struktúrájához, ezzel megkönnyítve annak bevezetését.

A rendszer kis és középvállalkozásoknak segítheti a hatékonyabb működést.

VII. Köszönetnyílvánitás

A publikáció elkészítését az EFOP-3.6.1-16-2016-00022 számú projekt támogatta. A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósult meg.

Köszönetemet szeretném kifejezni Erdei Timotei Istvánnak és Dr. habil. Husi Géza Tanár Úrnak a projekt során való segítségért és útmutatásáért.

VIII. Hivatkozások

[1] Spring keretrendszer (2018) [Online]. Available: https://spring.io/

[2] Java (2018, April 17) [Online]. Available: https://www.java.com/en/

[3] MySQL (2018, April 19) [Online]. Available: https://www.mysql.com/

[4] Thymeleaf (2017, November 5) [Online]. Available: https://www.thymeleaf.org/index.html

[5] Z. Jovanovic, D. Jagodic, D. Vujicic, R. Sinisa„Java Spring Boot Rest WEB Service Integration with Java Artificial Intellgence Weka Framework" International Scientific Conference Gabrovo, 2017 November, pp II-270.

[6] Apache Tomcat (2018. May 11.) [Online]. Available: http://tomcat.apache.org/

[7] Z. Jovanovic, D. Jagodic, D. Vujicic, R. Sinisa„Java Spring Boot Rest WEB Service Integration with Java Artificial Intellgence Weka Framework" International Scientific Conference Gabrovo, 2017 November, pp II-271.

[8] E. F. Codd, „A relational model of data for large shared data banks", Communications of the ACM Volume 13 Issue 6, 1970. June, pp. 379.

[9] Hibernate ORM (2018) [Online]. Available: http://hibernate.org/orm/

[10] Trygve Reenska, (1978-79), [Online]. Available: http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html

[11] D.-P. Pop, A. Altar, „Designing an MVC Model for Rapid Web Application Development", Procedia Engineering Volume 69, 2014, March 25, pp. 1173-1175.

[12] S. Segura, J. A. Parejo, J. Troya, A. Ruiz-Cortés, „Metamorphic Testing of RESTful Web APIs", IEEE Transactions on Software Engineering, 2017. October, pp 1.

[13] K. H. Jin, „Teaching Responsive Web Design to Novice Learners", the 18th Annual Conference", 2017. September

[14] Zs. Molnár, T. I. Erdei, N. C. Obinna, G. Husi, „A novel design of an air-cushion vehicle and its implementation," MATEC Web Conf. Volume 126,