

**Szegedi Tudományegyetem**  
**Informatikai Intézet**

# **SZAKDOLGOZAT**

**Urban Roland**

**2020**

**Szegedi Tudományegyetem  
Informatikai Intézet**

**CRM rendszer megvalósítása Spring Boot  
keretrendszerrel**

**Szakdolgozat**

Készítette:  
**Urban Roland**  
gazdaságinformatikus  
hallgató

Témavezető:  
**Dr. Bodnár Péter**  
adjunktus

**Szeged  
2020**

## FELADATKIÍRÁS

A feladat egy CRM rendszer megtervezése és implementálása. A webes alkalmazásban tárolt adatok egy tetszőlegesen választott adatbázisban legyenek elhelyezve. A projekt tetszőleges nyelven elkészíthető.

A rendszerrel szemben támasztott főbb követelmények az alábbiak:

- Képesnek kell lennie a hibajegyek feltöltésére, tárolására és annak megjelenítésére.
- Meg kell tudnia különböztetni a hibajegyeket státuszuk szerint.
- Lehesse regisztrálni, amivel személyre szabott adatok érhetőek el.
- A hibajegy prioritásának kezelése.
- A hiba továbbítása a megfelelő csoport számára.
- Keresés a hibák és az ügyfelek közt
- Statisztikai kimutatások készítése.
- A vezetői felületen az új felhasználók regisztrációjának elfogadása vagy elutasítása.
- Képes legyen kezelni a hibajegyekhez rendelt adatokat:
  - név
  - bejelentő
  - bejelentés típusa
  - a hiba prioritása
  - a bejelentés címe
  - leírás
  - dátum
- A hibabejegyzés élelciklusának kezelése, nyomon követése.
- Támogassa a felhasználók kijelentkezését a rendszerből.

# **TARTALMI ÖSSZEFOGLALÓ**

# **TARTALOMJEGYZÉK**

# 1. BEVEZETÉS

Manapság az Internet mára egy világméretű hálózattá nőtte ki magát, amit több millió ember használ. Ahogy növekedett, úgy jelentek meg újabb szabványok, technológiák és ajánlások. Napjainkban, nem csak szükséges, hanem elengedhetetlen, a használat mind a hétköznapi felhasználóknak, mind a vállalatoknak is.

A web-es technológiák már régóta érdekelnek így amikor a szakdolgozati témaválasztásra került a sor, számomra egyértelmű volt, hogy olyat jelölök meg ami része ennek a nagy rendszernek. Mindig is fontos volt, hogy olyan technológiákat ismerjek meg, melyek napjainkban is helytállnak és a jövőben is helyt fognak. A Szegedi Tudományegyetem

folytatott tanulmányaim alatt megismerkedtem rengeteg ilyennel, de ezek közül kiemelném a Java-t, melyet a leggyakrabban használtam nem csak az egyes kurzusokhoz, hanem hobbi projektekhez is.

A dolgozat megírásának további pozitívuma lehet számomra, hogy nagyobb rálátásom lesz az alkalmazás-tervezésre, illetve a fejlesztésre, így bízom benne, hogy ezen ismeretek birtokában nagyobb lehetőségem lesz e területen elhelyezkedni

Szakdolgozatom témája egy CRM rendszer megtervezése és implementálása. A dolgozat első részében magáról a CRM rendszerről írok, majd a feladat megoldása során felhasznált technológiákat ismertetem és részletezem. Ezután a tervezési folyamatról írok hosszabban. Ezt követően az implementációs folyamat részletezése következik. Végül a szakdolgozatot egy összegzés zárja.

## 2. ÜGYFÉLKAPCSOLAT-MENEDZSMENT

### 2.1 A CRM általános jellemzői

Az ügyfélkapcsolat-menedzsment (Customer Relationship Management, CRM) tulajdonképpen új ügyfélkapcsolatok létrehozásával, ezek fenntartásával, valamint az ügyfelekről szóló információk felhalmozásával és megőrzésével foglalkozik.

Egy CRM rendszer segítségével a saját ügyfelekről egy információs adatbázist lehet létrehozni, amelybe a szervezethez tartozó minden felhasználó rögzíthet új értékesítési lehetőségeket, feladatokat. Ha az ügyfelekkel kapcsolatos minden információ egy helyen van, rendszerezetten tárolva, akkor a rendszert használók könnyebben, szervezettebben és hatékonyabban működhetnek együtt az ügyfelekkel és egymással is.



1. ábra  
CRM rendszer összetétele  
(Forrás [3])

A legtöbb CRM rendszer építőelemei:

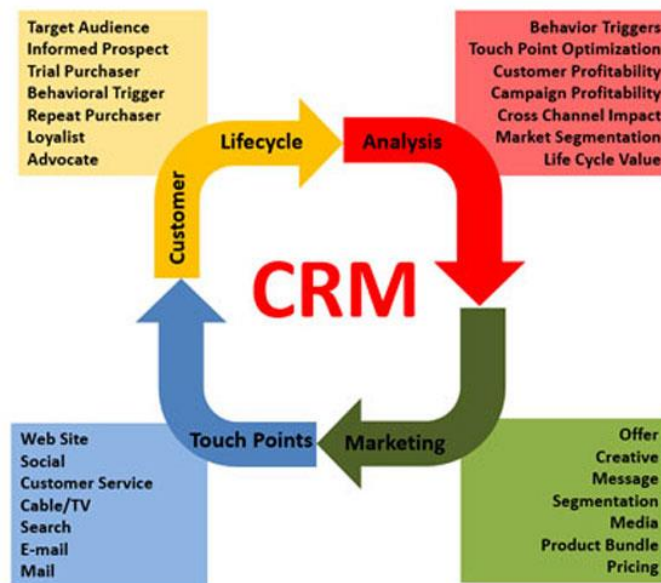
- **Ügyfelek:** lehetnek természetes személyek vagy kapcsolattartók abban az esetben, ha az ügyfél egy szervezetet jelent.
- **Adatlapok:** ezeken kerülnek tárolásra az egyes ügyfelekhez köthető értékesítési lehetőségek, segítségükkel követhető és mérhető az üzletkötés folyamata.
- **Teendők:** az ügyfelekkel történt kapcsolattartási események (e-mail küldés, telefonhívás stb.) feljegyzései, valamint a jövőbeni feladatok, emlékeztetők, amelyek az adatlapokon kerülnek rögzítésre.

Ezekből létrejön egy adathalmaz, ami lehetővé teszi többek között egy személyre való keresést, láthatók a hozzá kapcsolódó múltbeli értékesítések és jövőbeni lehetőségek, valamint az összes vele történt interakció.

Az alábbi területekre alkalmazható ez a rendszer:

- call centerek fejlesztése, átszervezése, telepítése, létesítése
- e-business
- értékesítési módszerek
- tréningek, gyakorlati támogatás
- ügyfél és piac szegmentáció
- értékesítési hatékonyság, stratégia, csatornák
- ügyfélelégedettség mérése

## 2.2 A CRM rendszer körforgása



2. ábra  
CRM körfolyamata  
(Forrás[4])

Mint már fentebb említettem a CRM alkalmazások központjában az ügyfelekről begyűjtött adatok állnak, hiszen ezek alapján tud a vállalkozás feléjük nyitni. Ezen adatok az ügyfelek általános adataira, termék vagy szolgáltatásokra, a velük megkötött megállapodásokra és az általuk előnyösnek vélt szállítási és egyéb szolgáltatásokra vonatkozhatnak. Ezek természetesen függenek a vállalkozás típusától, így egy banknak fontos lehet, hogy az ügyfél mennyit utazik külföldre, míg a szakácsnak nem biztos, hogy ez az információ releváns lenne. Az adatokat először be kell gyűjteni a klienstől, majd ezeket analizálni kell. Amennyiben ez a folyamat sikeresen megtörtént úgy a cég már könnyen el tudja dönteni, hogy az adott ügyfélnek mire van szüksége, így könnyebben bevonhatja az ügyfelet, és elkezdheti a kapcsolatépítést. Ezután a kapcsolatot fenn kell tartania, amihez még több adatra van szüksége az ügyfélről. Ekkor újra elkezdődik az információszerzés, melyet ismét analízis követ és az eljárás egy körforgásként működhet tovább.

### 3. FELHASZNÁLT TECHNOLÓGIÁK

A projekt elkészítésénél használt különböző komponensek és technológiák a következők:

A rendszert az Spring Tool Suite(STS)nevű nyílt forráskódú, platformfüggetlen szoftverkeretrendszer segítségével fejlesztettem, Java programozási nyelven.



A projektet az Apache Maven plugin segítségével hoztam létre különböző modulokká felosztva és a Maven segítségével épül fel az alkalmazás.

A webalkalmazást a Spring Boot keretrendszer használatával fejlesztettem.

Adattárolásra PostgreSQL relációs adatbázist használtam.

A webalkalmazást Apache Tomcat szerveren fejlesztettem, teszteltem.

A webes felület a Bootstrap front-end keretrendszerrel, Bootstrap icons betűtípus és ikonkészlettel valósítottam meg.

A projekt hosztolásához a Herokut választottam, lényegében ez egy platform szolgáltatás (PaaS) webalkalmazásokhoz.

A különböző diagramokat High-chart api segítségével jelenítettem meg.

A relációs adatok kezelésére JPA használtam.

A projektet készítése során az Git verziókövető rendszert használtam

### 3.1 Spring és Spring Boot

Ahhoz, hogy a teljes projektet egy nagy egésként tudjam kezelni, szükségszerű volt egy keretrendszerre használata. Ebbe a rendszerbe elhelyeztem magát az alkalmazást, így lehetőséget kaptam arra, hogy különböző szolgáltatásokat vegyek igénybe, amelyek megkönnyítették és felgyorsították számomra a fejlesztést.

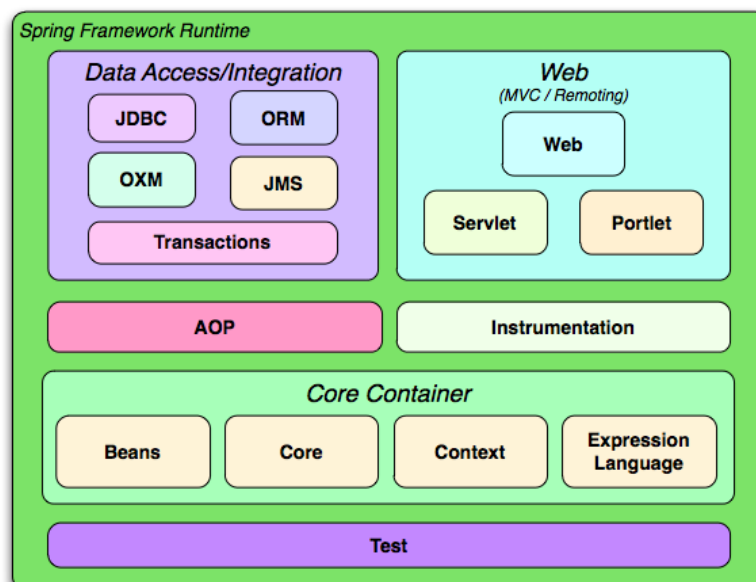
Először a Spring-et mutatom be, ezután tovább haladva a Spring Bootot is, hiszen az alkalmazásomat ennek a keretrendszernek a használatával valósítottam meg.

A Spring egy nyílt forráskódú keretrendszer, amely az IOC (Inversion of Control, fordított vezérlés) tervezési mintára alapszik Java nyelven. A Spring gyakorlatilag egy IOC konténer, az IOC a Dependency Injection (függőség injektálás) támogatásával végzi el. Szoftverfejlesztés során az egyes elemek működése gyakran függ más elemek által nyújtott szolgáltatásoktól. Jellemzően tudniuk kell, hogy mely komponensekkel kommunikáljanak, hogy azok hol találhatóak, és hogy miként kommunikáljanak velük. Ha egy ilyen szolgáltatás valamilyen módon megváltozik, akkor azt valószínűleg elég sok helyen át kell írni. A szokásos módszer a kód struktúrálására az, hogy a szolgáltatások elérésének módját az alapvető logika részeként implementáljuk. Egy másik módszer, hogy elkerüljük az előzővel járó problémákat, hogy csak függőségeket deklarálunk a szükséges szolgáltatásokhoz, és valamilyen módon egy külső kódrészlet felel majd ezek megtalálásáért és inicializálásáért. Ez engedélyezi, hogy a szolgáltatások

megváltozása esetén az ezeket használó kódrészleteket ne kelljen módosítani. Ezt a típusú módszert hívjuk Dependency Injection-nek.

A Spring egyetemes megoldásokat kínál a fejlesztés során felmerülő problémákra, a gyors és eredményes alkalmazásfejlesztésre, illetve rétegelt alkalmazások készítésére. A Spring rétegelt összetételű, ami azt jelenti, hogy a különböző fajtájú feladatok elvégzésére különböző modulok vannak. Ezek a modulok hat kategóriába sorolhatók (3. ábra). Ilyen például az adatbáziskezelés, webes szolgáltatások, tesztelési megoldások, amelyek mind külön modulokban találhatók.

Ennek a modularitásnak az a jelentősége, hogy nem kell az alkalmazást teljes mértékben a Spring keretrendszerrel függővé tenni, nem kell hozzákapcsolni magunkat. Szabadon



3. ábra  
Spring modulok csoportosítása  
(Forrás[4])

dönthetünk arról, hogy mely modulokat használjuk, ami konkrétan kell, illetve, ha valamelyik modul nem megfelelő a számunkra, akkor egyszerűen egy másik keretrendszer könyvtárát használjuk a Spring által kínált megoldás helyett.

Tény az, hogy a Spring megkönnyíti a fejlesztés folyamatát a modulok megalkotásával, de maga a szoftver eltérő beállításait, mint például a Maven függőségek hozzáadását és az XML fájlok konfigurálását a fejlesztőnek kell kiviteleznie. Erre a problémára kínál megoldást a Spring Boot

Ez a keretrendszer a Spring család egyik tagja, amely olyan plusz funkciókkal rendelkezik a Springhez képest, amelyekkel még eredményesebbé, hatékonyabbá és

könnyebé teszi a fejlesztést. Az a célja, hogy a szoftverfejlesztők ne a program konfigurálásával foglalkozzanak, hanem koncentráljanak a megírandó kódra, és hogy azt az időt, amit a beállítások módosításával töltenének, azt az üzleti logikára és az implementálására fordíthassák .

### 3.2 Maven

Az Apache Maven egy csomagoló keretrendszer Java projektekhez. Egyik legfontosabb feladata, hogy automatizál folyamatokat. A Maven egyrészt leírja, hogy miként is épül fel a projekt, másrészt pedig meghatározza a projekt függőségeit(dependency) más moduloktól, vagy külső függvénykönyvtáraktól, amely csomagokat a build folyamat során automatikusan le is tölt. Mind ezen információk egy xml fájlban vannak eltárolva, melynek neve pom.xml. A külső package-ket dinamikusan tölti le egy, vagy több repository-ból, ez általában a központi Maven repository. A letöltött csomagokat a Maven helyi gyorsítótárban tárolja.

A már említett pom.xml file (pom = Project Object Model) tartalmazza az összes olyan adatot, ami elengedhetetlen egy program buildeléséhez. Ez többnyire magában foglalja a projekt nevét, tulajdonosát és a függőségek listáját. A kiterjedtebb projekteket általában szétszedik különálló modulokká, ahol minden modulhoz külön tartozik egy pom fájl. Ilyenkor annak a projektnek a pom.xml fájlja, ami magában foglalja a modulokat, lesz a gyökere az alkalmazásnak, ennek segítségével könnyedén lehet buildelni az összes modult. Az én projektemnél csak egy modult használtam.

### 3.3 PostgreSQL

A PostgreSQL egy nyílt-forráskódú relációs adatbázis-kezelő rendszer (RDBMS). A relációs adatbázis az adatokat táblákba rendezi, szerkezete: az adatok azonos összetételű sorokként kerülnek be egy táblába, ez a rekord. A felépítést, azaz a megőrzendő adatokat az oszlopok határozzák meg, ez pedig a mező. A tábla így egy egyedtípust reprezentál. Ha egy mezőt elsődleges kulcsnak jelölünk ki, akkor annak értéke egyértelműen meghatározza a rekordot, nem ismétlődhet. Ezt kiaknázva megadhatjuk ezt a mezőt egy másik táblában idegen kulcsnak jelölve, majd lekérdezéseknél egyesíthetjük a táblákat a kulcsértékek alapján. Minden relációs adatbázis-kezelő rendszer, így a PostgreSQL is az SQL (Structured Query Language, azaz struktúrált

lekérdező nyelv) nyelvet használja az adatbázis létrehozására, módosítására és az adatok lekérdezésére. A PostgreSQL biztosít grafikus felületet az adatbázisok kezelésére. Én a munkám során parancssort és a pgAdmin-t használtam, ami egy PostgreSQL adatbázisok kezelésére írt grafikus felületet biztosító eszköz.

### 3.4 JPA (Java Persistence API)

Mint ahogy fentebb is említettem a JPA a relációs adatokat kezeli. egy keretrendszer a Java programozási nyelvhez. Lényegében az adatok automatizálásának áramlása az adatbázisból a programba és vissza.

A kulcsszó itt a perzisztencia (tartós fennmaradás). Ezt olyan adatokra használjuk, mely túléli az őt létrehozó folyamatot. A Java perzisztenciát kicsit másképpen határozzuk meg, ebben az esetben arról van szó, hogy a tárolás a Java programozási nyelv segítségével történik. Java-ban az adatok perzisztálására különböző módok is léteznek, többek között ilyen a:

- szerializációság,
- JCA,
- XML,
- JDBC adatbázisok.

A fejlesztés során azért döntöttem a JPA mellett, mert nagyban megkönnyítette az implementálást azáltal, hogy automatizált bizonyos folyamatokat, illetve az annotációk segítségével egyszerűbben meg lehet mondani, hogy milyen attribútumok legyenek elérhetőek a későbbiekben.

### 3.5 GIT

A GIT [12] egy nyílt-forráskódú verziókövető rendszer (version control system, VCS). A verziókövető rendszerek kezelik a fájlokat és mappákat, valamint eltárolják a rajtuk megvalósított változtatásokat. Ez nagymértékben megkönnyíti a csapatban végzett fejlesztési munkákat, mivel nyomon követhetjük a saját, illetve a mások által végzett módosításokat, hiba esetén akár mikor visszatölthetjük a rendszer egy korábbi állapotát. Bármilyen hálózaton üzemeltethetünk GIT szerveret, ennek okán egymástól távol is eredményesen dolgozhatunk. Az, hogy másfajta fejlesztők egymástól távol képesek ugyanazt az adathalmazt kezelni és módosítani, jelentősen megkönnyíti az együttműködést. Mivel a munka verziókezelt, nem kell attól tartani, hogy valami súlyos

hibát, vagy minőségromlást okoz változtatásunk a rendszerben, ez esetben csupán vissza kell töltenünk egy régebbi verziót.

Némely verziókövető rendszerek egyúttal szoftver konfigurációs rendszerek is (software configuration management system, SCM). Ezeket a rendszereket arra tervezték, hogy tudják kezelni a forráskódot. Efféle például, hogy felismerik és képesek kezelni a különböző programnyelveket, valamint rendelkeznek szoftver fejlesztést segítő beépített eszközökkel is. A GIT azonban nem tartozik ezek közé, hanem egy általános verziókövető rendszer, amellyel bármilyen adathalmazt kezelhetünk. A verziókövető rendszer központja az ún. repository, ami a központilag tárolt adathalmazt jelenti. Ez legtöbbször fájlrendszer formájában tárolja az adatokat, faszerkezetbe van rendezve a fájlok és mappák hierarchiája. Akármennyi kliens kapcsolódhat a repository-hoz, olvashatja és írhatja a fájlokat. Eddig nem túl sok mindenben tér el egy fájlszervertől, amiben különbözik az az, hogy a repository nyilvántartja az összes változtatást, emlékszik a fájlok minden verziójára. A verziókezelő rendszerek másik alapvető eleme a munkapéldány (working copy), ez a verziókezelt fájlok lokális másolata az egyes kliensek gépén. A munkapéldányt a kliens először letölti a központi repository-ból, majd azon végzi el a változtatásait, fejlesztéseit, ezután hozzáadja a lokális tárolóhoz majd végül visszatölti azt a központi-ba, amennyiben nem történt olyan változás, ami ütközik egy másik kliens változtatásával. Ha az utóbbi megtörténik akkor először fel kell oldani a konfliktusokat, majd ezután válik lehetővé a feltöltés. Ennek megoldásában is segítenek az GIT eszközök. Szoftverfejlesztésnél legtöbbször van egy törzs (trunk), ami a szoftver fő verzióit tartalmazza, továbbá vannak a másféle ágak (branch), amelyeken fejlesztik a módosításokat, új funkciókat hoznak létre és csak utána vezetnek ezeket át a fő vonalra.

## 4.SPECIFIKÁCIÓ

A hibajegyek elbírálási folyamata során a rendszerben a vele kapcsolatba kerülő felhasználók különböző szerepkörökben különböző feladatokat látnak el. Ezek a szerepkörök határozzák meg az alkalmazás funkcionalitását. Most az egyes szerepkörök részletezése következik:

Vezető – Ezzel a jogosultsággal rendelkező felhasználó az alap hozzáféréseken kívül elérheti a kimutatásokat, különböző statisztikákat, illetve módosítani tudja a többi felhasználó jogkörét, valamint akár ki is tudja törölni a rendszerből.

Ügyintéző – Alapértelmezetten minden új felhasználónak ez lesz a szerepköre.

Elsődleges feladatuk a hibajegyek rögzítése, továbbítása, módosítása, lezárása, törlése.

Fejlesztő –

Tesztelő –

Hardverszerelő –

	Regisztrálás	Bejelentkezés	Új hiba bejegyzés rögzítése	Hibabejegyzés továbbítása	Hibabejegyzés lezárása	Statisztikák kimutatások	Fiók aktiválása	Felhasználónév, jelszó módosítás	Keresés az ügyfelek között	Keresés a hibajegyek között	Kijelentkezés
Vezető	x	x	x	x	x	x	x	x	x	x	x
Ügyintéző	x	x	x	x	x		x	x	x	x	x
Fejlesztő	x	x		x	x		x	x		x	x
Tesztelő	x	x			x		x	x		x	x
Hardverszerelő	x	x			x		x	x		x	x

4. ábra  
Szerep-funkció mátrix

Egy hibajegy lehetséges állapotai a következők:

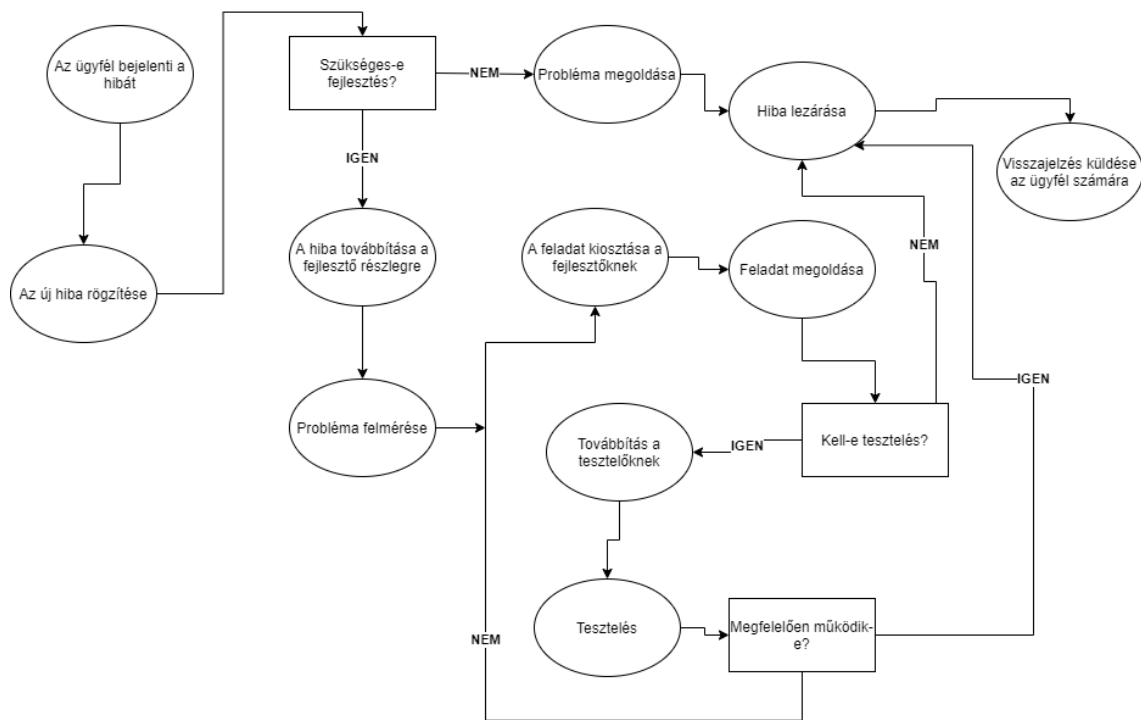
- Nyitott
- Zárt

- Folyamatban
- Felfüggesztett

- Kritikus hiba
- Gyorsfelmérés
- Extra fontos
- Nagyon fontos
- Fontos
- Normál

A továbbított hibabejegyzések a következő sorrendben lesznek kezelve a továbbiakban:

Az 5. ábra szemlélteti a hiba életciklusát, segítségével könnyebben megérthető, hogy a rendszer miként is kezeli a hibajegyeket.

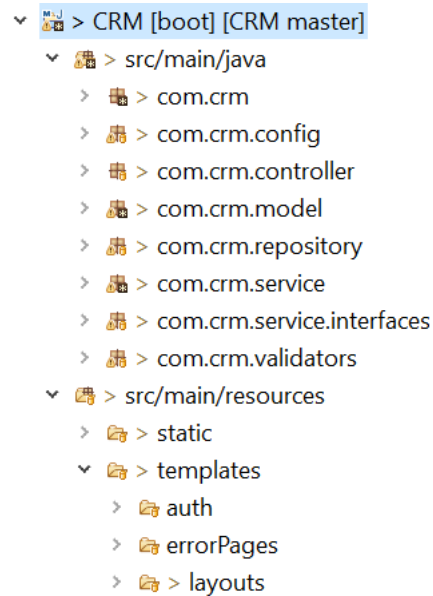


5. ábra  
A hiba életciklusa

## 5. TERVEZÉS

### 5.1 Modulok

A projekt Maven modulokba vannak szervezve. Az alkalmazás moduláris felosztása a következő:



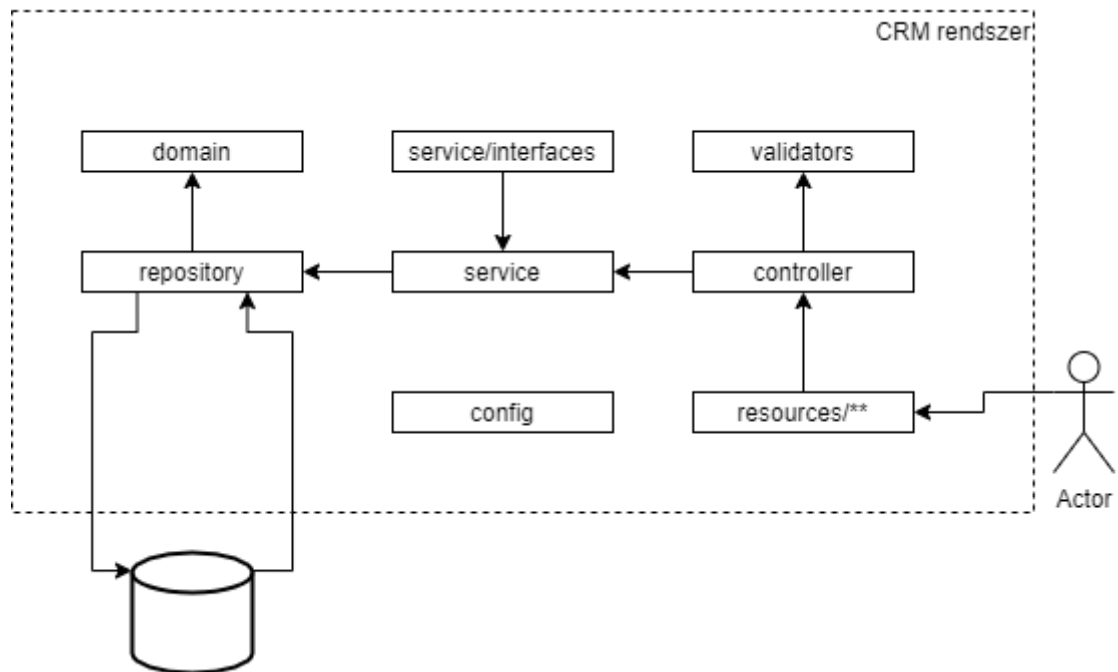
8. ábra  
alkalmazás csomag szerkezete

- config: Ez a modul felelős a biztonságért, itt található a Spring Security. Alapvetően két osztályt tartalmaz. *WebConfig* nézet konfiguráció a saját bejelentkező oldal útvonalának megadása. Illetve a *SecurityConf* ahol a jogosultságok kezelését valamint a jelszó titkosító algoritmus megadása a feladata.
- controller: Ez a réteg felelős a webes felület (font-end) és háttérrendszer (back-end) kommunikációjához. Szintén két osztályból áll *ErrorPageController* itt a különböző hibaoldalakat kezelni. *HomeController* lényegében bizonyos end point-okra különböző nézeteket ad vissza. Pontosabban a klienstől érkező kéréseket elkapja, majd meghatározza, hogy milyen szolgáltatásokat kell végrehajtania és visszaadnia. Hozzáfér a service modulhoz is interfészeken keresztül.



- model: Ezen modul tartalmazza a *pojo*-kat(plain old java object), más szóval az alap java osztályokat. A tárolandó adatszerkezeteket reprezentálják.  
Általánoságban elmondható, hogy ezek az osztályok rendelkeznek getter/setter metódus párral. Ezekre azért van szükség, mert elsősorban az *ORM*-ek(Object Relational Modeling, pl: Hibernate, JPA) masszívan használják őket.
- repository: Azért felelős, hogy a kód képes legyen kommunikálni a *entites*-ekkel(*pojo*). Az adatbázisból objektumokat fog várni, ezek a repository-k nem osztályok, hanem interfészek és mindegyik kiterjeszti a *CrudRepository*-t. Ennek köszönhetően már előre meg vannak írva bizonyos metódusok, ami gyorsabb fejlesztést tesz lehetővé.
- service/interfaces: Ez a modul tartalmazza azon interfészeket, amelyek a controller felé biztosítandó szolgáltatásokat írják le.
- service: Réteg felelős az üzleti logika megvalósításáért. Pontosabban a service/interfaces package-ben megírt metódusok megvalósítása a feladata. Hozzáfér a model, repository modulokhoz, valamint itt elérhető egyéb funkciók is (pl: email küldés).
- validators: Ezen modul tartalmazza az űrlapok validálásához szükséges logikát. Ezt a modult a controller fogja használni.
- resources/templates: Ebben a rétegben találhatóak a html fájlok. Amiket a felhasználó elérhet amennyiben megfelelő jogosultsággal rendelkezik.
- resources/templates/auth: Itt található a bejelentkezési oldal.
- resources/templates/errorPages: Ebben a modulban vannak a különböző hibaoldalak.
- resources/templates/layouts: Itt találhatóak a fragmentek. Ezekre azért van szükség mert egy adott oldalt többször is felhasználunk, így csak egyszer kell megírni.

- resources/static: Ebben a modulban vannak a statikus fájlok. Mint pl: javascript, css, illetve a képek.

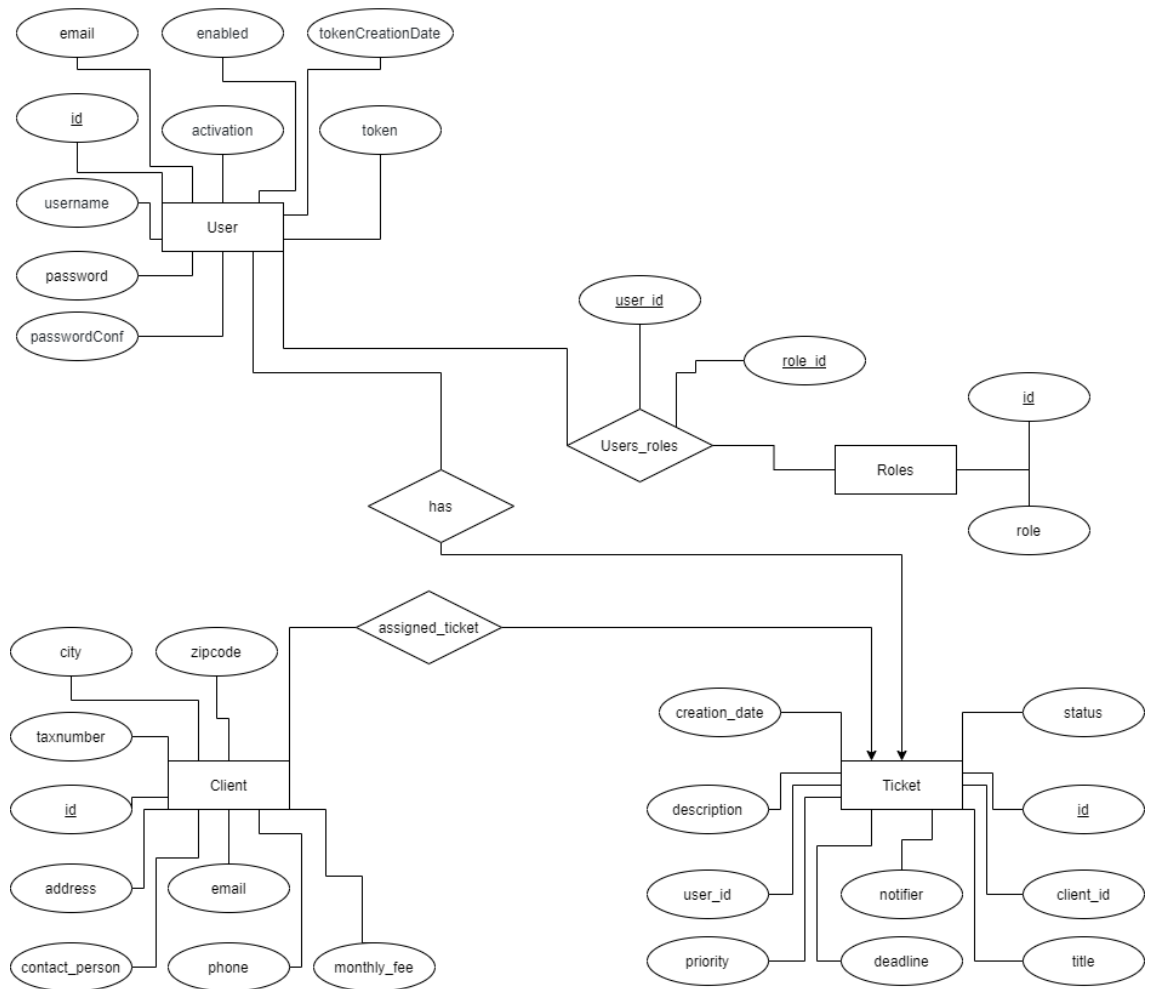


6. ábra  
Moduláris felépítés

## 5.2 Adatbázis tervezés

A tervezés folyamán egyik legfontosabb feladat az adatbázis megtervezése. Ezt az egyed kapcsolat diagrammal szemléltetem. Ez az ábra a tárolandó adatok és köztük lévő kapcsolatok grafikus ábrázolására szolgál. Ez a modell a fejlesztés során többször is módosításra került, ezáltal jelentős átdolgozási munkákat okozva.

Az adattárolási modell végleges EK modelljét szemlélteti a 7. ábra



7. ábra  
EK modell

### 5.2.1 Felhasználó

Látható, hogy az adattárolás három fő egyede a felhasználó (*User*) a hibajegy (*Ticket*) és az ügyfél (*Client*). A felhasználó egyed kulcsa a felhasználó azonosítója (*id*), ezért annak egyedinek kell lennie a rendszerben, valamint fontos megemlíteni a jelszót, amelyet titkosítva tárol el az adatbázis (*password*). Ezenkívül tartalmazza az alapvető adatokat, mint felhasználónév, email, aktivációs kód(*activation*), a profil aktív-e(*enabled*), *token* ami lehetővé teszi, hogy a felhasználó módosítja jelszavát egy bizonyos ideig, (*tokenCreationDate*) előbbi változónak a létrehozási ideje.

### 5.2.2 Hibajegy

A hibajegy egyednek egy egyedi azonosító lesz a kulcs tulajdonsága (*id*). Tárolásra kerül továbbá a státusza (*status*), az ügyfél azonosítója(*client\_id*), a hibajegy címe (*title*), határideje (*deadline*), aki bejelentette a hibajegyet (*notifier*), prioritása (*priority*), felhasználó, aki legutóbb módosította a hibajegyet (*user\_id*), a hiba részletes leírása (*description*), a létrehozási dátuma (*creation\_date*).

### 5.2.3. Ügyfél

Az ügyfél egyednek egy egyedi azonosító lesz a kulcs tulajdonsága (*id*). Továbbá tárolásra kerül az az összeg, amit havonta fizet (*monthly\_fee*), telefonszáma (*phone*), email címe (*email*), az a személy neve, akivel a kapcsolatot fel lehet venni (*contact\_person*), az ügyfél pontos címe(*address*), város (*city*), irányítószám(*zipcode*), valamint az adó azonosítója (*taxnumber*).

### 5.2.4. További egyedek

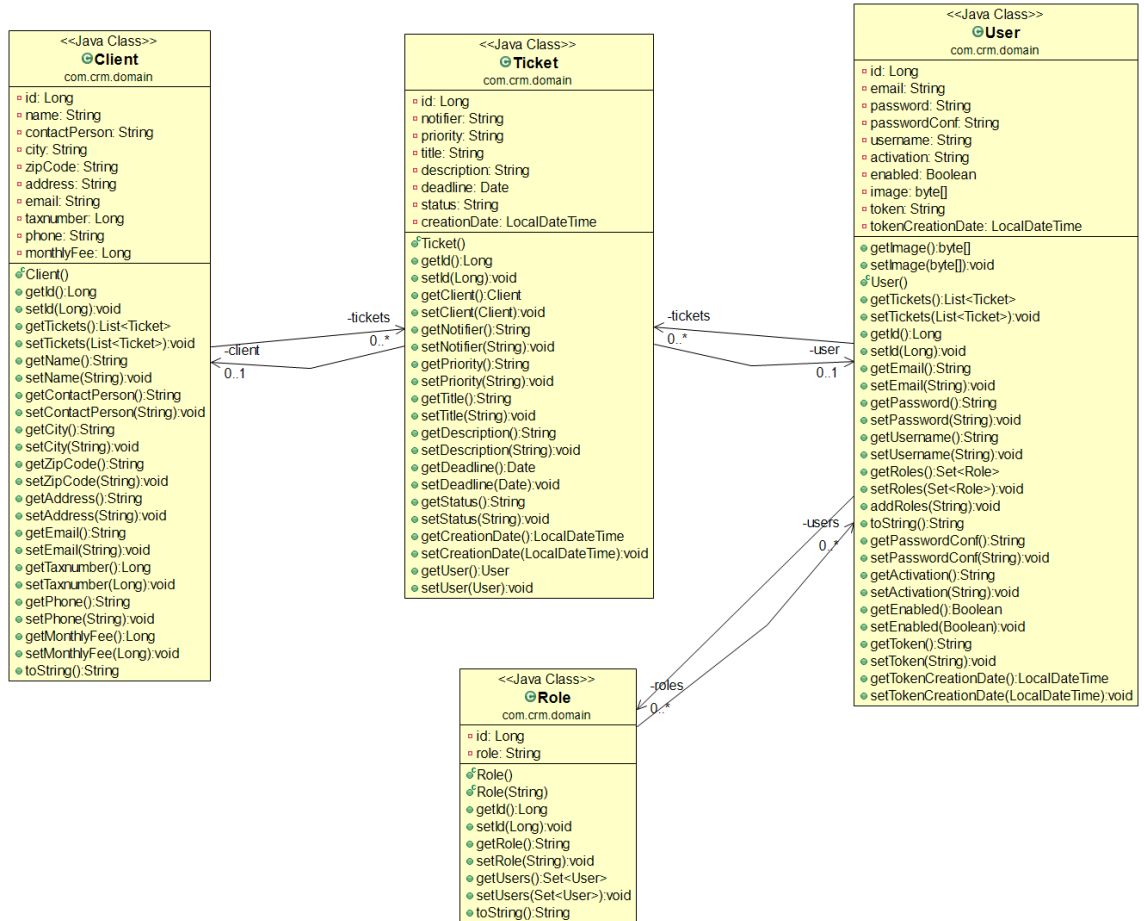
A Roles tábla tartalmazza a jogosultságokat, amivel minden felhasználó rendelkezik. A Users\_roles egy összekötő tábla, ami a user és roles táblákat kapcsolja össze. Alapvetően két attribútuma van a felhasználó azonosítója és a szerepkör azonosítója. Erre az átláthatóság, illetve a könnyebb kezelés miatt van szükség.

### 5.2.5 A felhasználók és a hibajegyek közötti kapcsolat

Mivel egy felhasználó rögzíthet több hibajegyet is és egy hibajegyet egyszerre csak egy felhasználó rögzíthet így a köztük lévő kapcsolat egy a többhöz.

### 5.3 Osztályok tervezése

A 9. ábra szemlélteti a pojo osztályokat és a köztük lévő kapcsolatot.



9. ábra  
POJO osztálydiagram

## 6. IMPLEMENTÁCIÓ

### 6.1. Entitások

A projekt implementációját ennél a modulnál kezdtem, segítségemre volt a JPA. Az entitások úgy kerültek kialakításra, hogy ahány tábla készült az adatbázisban, annyi entitás is lett előállítva. Ezeket az osztályok a model csomagban helyezkednek el.

```
▼ > CRM [boot] [CRM master]
  ▼ > src/main/java
    > > com.crm
    > > com.crm.config
    > > com.crm.controller
    ▼ > com.crm.model
      > > Client.java
      > > Role.java
      > > Ticket.java
      > > User.java
    > > com.crm.repository
    > > com.crm.service
    > > com.crm.service.interfaces
    > > com.crm.validators
```

Az alábbi táblázat összefoglalja, hogy mely entitások lettek létrehozva (táblázat 1).

osztálynév	leképezés
User	USERS
Ticket	TICKET
Role	ROLES
Client	CLIENT
users_roles	USERS_ROLES

1. táblázat – Létrehozott entitások

Az entitások szerkezete nagyban hasonlít egymásra, eltérések, a tulajdonságok nevei és típusai között van.

Egy entitás általános szerkezete a következő:

- Az osztálydefiníció előtt meg kell adni az `@Entity` annotációt, ezzel jelezve a keretrendszer számára, hogy ez egy JPA entitás. Valamint opcionális karaméterként egy `@Table` annotációt, amivel a tábla nevét tudjuk megadni.
- Az entitásban meg kell adni az adott tábla mezőit privát adattagokkal, rájuk pedig egy `@Column` annotációt rakni, amelyben meg lehet adni, hogy mely mezőre mutasson, valamint, hogy lehet-e null az értéke. Az azonosító adattagnál(ID) pedig ki kell tenni az `@Id` valamint a `@GeneratedValue(strategy`

= GenerationType.IDENTITY) annotációkat ezzel jelezvén, hogy ez lesz az entitás azonosítója.

- Az adattagok felett meg lehet adni annotációk segítségével a kapcsolatok típusait is @ManyToMany(több a többhöz), @OneToOne(egy az egyhez), @ManyToOne(több az egyhez)
- Minden adattaghoz írni kell egy publikus getter, setter metódust

## 6.2 Adathozzáférési interfész (DAO)

A projektben a komponensek konfigurációját annotációkkal valósítottam meg, mivel számomra ez tűnt a legegyszerűbbnek és egyben a leghatékonyabbnak is.

Az interfészek megtervezése során az egyedek alapján választottam szét a funkciókat. Ennek értelmében négy interfészt hoztam létre a repository csomagjába, ezek: a felhasználók adatainak kezelését végző UserRepository, a hibajegyekkel kapcsolatos adatmozgatási funkciókat leíró TicketRepository, az ügyfelekkel kapcsolatos adatok kezelésére ClientRepository és a szerepköröket végző RoleRepository.

### 6.2.1 DAO interfészek és metódusok

Először a DAO-k interfészeit nézzük meg közelebbről majd pedig az implementációit (2.táblázat)

Interfész	Implementáció
ClientRepository	ClientServiceImpl
RoleRepository	Nincs
TicketRepository	TicketServiceImpl
UserRepository	UserServiceImpl

2.táblázat - DAO interfészek és implementációk

Ami feltűnhet, hogy a RoleRepository-nak nincs implementációja ez azért van mert a UserServiceImpl-ben van megvalósítva. Valamint az interface és az implementáció között be lett iktatva egy másik interface ami növeli a projekt átláthatóságát. Az is látható, hogy egy elnevezési konvenciót alkalmaztam az a teljes projekten belül érvényesül. A névadási konvenció a következő: az osztálynevek eleje valamilyen

entitásra utal majd, hogy mely réteghez tartozik, valamint, hogy interfészről vagy implementációról van-e szó.

Az alábbiakban nézzük Repository-k metódusait, valamint azok feladatait.

#### **ClientRepository** (3.táblázat)

metódus	leírás
findAll()	visszaadja egy listában az összes ügyfelet
save(Client)	elmenti a hibajegyet

3.táblázat - ClientRepository metódusai és leírásai

#### **RoleRepository** (4. táblázat)

metódus	leírás
findByRole(String)	visszaad egy szerepkört
findByRoleName(String)	visszaad egy Role-nak egy azonosítóját

4.táblázat - RoleRepository metódusai és leírásai

#### **TicketRepository** (5.táblázat)

metódus	leírás
findAll()	visszaadja az összes hibajegyet
saveTicket(Ticket)	elmenti a megadott hibajegyet
updateTicket(String,String,String,String,String, Date, User, Long)	módosítja a hibajegyet a paraméterben megadott értékekre
deleteById(Long)	kitörli a hibát azonosító alapján
ticketGroupedByStatus()	státusz alapján csoportosítja a hibajegyeket
ticketsGroupedByPriority()	prioritás alapján csoportosítja a hibajegyeket
ticketGroupedByMonths()	hónap alapján csoportosítja a hibajegyeket

5. táblázat – TicketRepository metódusai és leírásai

#### **UserRepository** (6.táblázat)

metódus	leírás
registerUser(User)	felhasználó regisztrálása



findByEmail(String)	megkeresi a paraméterül adott email címet(ha létezik)
allEmail()	visszaadja az összes email címet
userActivation(String)	aktíválja a felhasználót
forgotPassword(String)	a felhasználónak kiküld egy email-t, amin keresztül meg tudja változtatni a jelszavát
resetPassword(String,String)	megváltoztatja a felhasználó jelszavát
updateUserName(String, String)	módosítja a felhasználónevet
updateUserPassword(String,String,String)	módosítja a felhasználó jelszavát
findById(Long)	megkeresi a felhasználót, ha létezik
findAll()	visszaadja az összes felhasználót
updateUserRole(Long,Long)	módosítja a felhasználó szerepkörét
findByRoleName(String)	megkeresi a megadott szerepkört név szerint
deleteUsers_Roles(Long)	törli a felhasználó szerepkörét
deleteById(Long)	törli a felhasználót azonosító alapján

6. táblázat – UserRepository metódusai és leírásai

### 6.3 Service réteg

Az alkalmazást az MVC tervezési minta alapján hoztam létre, annyi kiegészítéssel, hogy a Model és a Controller réteg közé készítettem egy Service és egy Configuration réteget is.

A Service réteg elsődleges feladata az üzleti logika megvalósítása. A Service-ben definiálva van a Repository @Autowired jelzéssel, ami a dependency injection egy típusa. Pontosabban konstruktor alapú DI valósít meg. A Service osztályban az osztálydefiníciók előtt be kell szűnni a @Service annotációt.

Ennek a rétegnek az az előnye, hogy „tehermentesíti” a controller-t. Valamint átláthatóbb, jobban strukturált és biztonságosabb lesz tőle maga az alkalmazás. Minden táblának van egy külön Service részlege kivéve a Role osztályt, ezek a service csomagban találhatóak meg. Valamint a strukturáltság miatt a Service osztályokban megvalósított metódusokat kigyűjtöttem külön-külön egy interface-be. Ez megtalálható a service/interfaces csomagban.

```

> CRM [boot] [CRM master]
  > src/main/java
    > com.crm
    > com.crm.config
    > com.crm.controller
    > com.crm.model
    > com.crm.repository
    > com.crm.service
      > ClientServiceImpl.java
      > EmailServiceImpl.java
      > TicketServiceImpl.java
      > UserDetailsImpl.java
      > UserServiceImpl.java
    > com.crm.service.interfaces
      > IClient.java
      > IEmail.java
      > ITicket.java
      > IUser.java
    > com.crm.validators

```

Service osztályok részletesen (7. táblázat)

Interface	Implementáció
IUser	UserServiceImpl
ITicket	TicketServiceImpl
IClient	ClientServiceImpl
IEmail	EmailServiceImpl
UserDetails	UserDetailsImpl

7. táblázat – Service interfacek és implementációik

Akár csak, mint a Repository rétegnél, itt is bemutatásra kerül, hogy milyen metódusokat tartalmaznak az osztályok. Mivel a Repository rétegnél már bemutatásra kerültek a metódusok, nem részletezném, itt már csak azt mutatom meg, hogy melyik metódus mely korábbi metódusra hivatkozik.

A UserRepository műveleteit az IUser kezeli (8. táblázat)

A IUser metódusai a következők:

metódus	DAO művelet
registerUser	userRepository.save(User)
findByEmail	userRepository.findByEmail(String)

allEmail	userRepository.allEmail()
userActivation	userRepository.userActivation(String)
forgotPassword	userRepository.forgotPassword(String)
resetPassword	userRepository.resetPassword(String,String)
updateUserName	userRepository.updateUserName(String,String)
updateUserPassword	userRepository.updateUserPassword(String,String,String)
findById	userRepository.findById(Long)
findAll	userRepository.findAll()
updateUserRole	userRepository.updateUserRole(Long,Long)
findByRoleName	userRepository.findByRoleName(String)
deleteUsers_Roles	userRepository.deleteUsers_Roles(Long)
deleteById	userRepository.deleteById(Long)

8. táblázat – IUser metódusai és azok metódushívásai

A TicketRepository műveleteit az ITicket kezeli (9. táblázat)

A ITicket metódusai a következők:

metódus	DAO művelet
findAll	ticketRepository.findAll()
saveTicket	ticketRepository.saveTicket(Ticket)
updateTicket	ticketRepository.updateTicket(String, String, String, String, String,Date,User,Long)
deleteById	ticketRepository.deleteById(Long)
ticketsGroupedByStatus()	ticketRepository.ticketsGroupedByStatus()
ticketsGroupedByPriority()	ticketRepository.ticketsGroupedByPriority()
ticketGroupedByMonths()	ticketRepository.ticketGroupedByMonths()

9. táblázat – ITicket metódusai és azok metódushívásai

A ClientRepository műveleteit az IClient kezeli (10. táblázat)

A IClient metódusai a következők:

metódus	DAO művelet
saveClient	clientRepository.saveClient()
findAll	clientRepository.findAll()

10. táblázat – IClient metódusai és azok metódushívásai

A UserDetails egy speciális osztály, amelyet a Spring Security használ. Ide kapcsolódik a bejelentkezési funkciók kezelése, jelszó titkosítás, felhasználó akíválása.

Ezt az osztály maga a keretrendszer adja így neki nincs külön Repository rétege.

Viszont a service rétegben egészítettem ki a saját implementációval.

A UserDetails műveleteit az UserDetailsImpl kezeli (11. táblázat)

A UserDetailsImpl metódusai a következők:

metódus	leírás
getAuthorities()	itt lehet megadni a saját szerepköröket a security számára
getPassword()	a felhasználó jelszava
getUserEmail()	a felhasználó email címe
getUsername()	a felhasználó felhasználóneve
getName()	a felhasználó felhasználóneve
getId()	a felhasználó azonosítója
getRole()	a felhasználó szerepköre
isAccountNonExpired()	a felhasználó fiókja lejárt-e
isAccountNonLocked()	a felhasználó fiókja lezárt-e
isCredentialsNonExpired()	a felhasználó jelszava lejárt-e
isEnabled()	a felhasználó fiókja aktiválva lett-e








11. táblázat – UserDetailsImpl metódusai és azok metódushívásai

#### 6.4 Controller réteg

Az adattárolási modell és a felület közötti réteg az MVC architektúrában. Alapvető feladata a bejövő kérések kezelése és a megfelelő válaszok visszaadása. Ezen osztályok felépítése, szintén hasonlítanak egymáshoz. Jellemzőik:

- @Controller annotáció használata az osztálydefiníció előtt
- Ezekbe az osztályokba be kell kötni a megfelelő Service-eket, @Autowired jelzéssel ellátni
- Valamint a service eljárásokat kezelő metódusokat a @RequestMapping jelzéssel láttam el.

A projekten két Controller osztály található a Home és az ErrorPage utóbbi a hibaoldalakat kezeli, pl: nincs ilyen oldal vagy belső hiba történt, előbbi pedig minden más.

```
▼  > CRM [boot] [CRM master]
  ▼  > src/main/java
    >  > com.crm
    >  com.crm.config
    ▼  com.crm.controller
      >  ErrorPageController.java
      >  HomeController.java
```

## **A jelenlegi rendszer analízise**

A jelenleg használt rendszert egy kisvállalkozásnál használták. A célja az volt, hogy az ügyfelek esetleges hibáit begyűjtse, tárolja és megoldja ezeket. A jelenlegi rendszer egyik legnagyobb hibája, hogy teljesen papíralapú. E tulajdonsága nem teszi lehetővé a különböző hibák gyors és hatékony megoldását. Illetve az egyre növekvő adatok dinamikus kezelése igen nehézkessé vált. A vállalkozás belátta, hogy hosszú távon ez a rendszer nem kifizetődő így beruháztak egy saját CRM rendszerbe. Ez okból kérték, hogy tervezzek egy rendszert, ami kielégíti az igényeiket. Kisvállalkozás révén külön kérésük volt, hogy az egyszerű felhasználói felülettel mellett az új felhasználók regisztrációjának aktiválását a vezetők hagyják jóvá. A jelenlegi rendszer működése az alábbi folyamatokra bontható.

## **Hibák bevitele és feldolgozása**

Az az ügyfél, aki valamilyen hardveres vagy szoftveres hibát észlelt felhívta a vállalkozás ügyfélszolgálatát, az ügyintézőnek elmondta a problémát. Ez rögzítésre került papír formájában. Amennyiben az ügyintéző nem tudott segíteni a megoldásban abban az esetben tovább küldte az illetékeseknek. Ők elemezték a hibát majd visszacsatolást küldtek az ügyintézőnek, aki értesítette az ügyfelet

## ***Irodalomjegyzék***

- [1] [https://www.soulware.hu/szakmai-reszletek/a-crm-reszletebben/?gclid=Cj0KCQjwka\\_1BRCPARIsAMlUmErp3rORyrxHa-gWCO4X\\_ODosv66Lzhs\\_bacdrquNuRyX57OUhmBs4aAsMcEALw\\_wcB](https://www.soulware.hu/szakmai-reszletek/a-crm-reszletebben/?gclid=Cj0KCQjwka_1BRCPARIsAMlUmErp3rORyrxHa-gWCO4X_ODosv66Lzhs_bacdrquNuRyX57OUhmBs4aAsMcEALw_wcB)
- [2] <https://hu.wikipedia.org/wiki/CRM>
- [3] <https://www.perfectviewcrm.com/what-is-crm/>
- [4] <https://www.resonantanalytics.com/services/customer-relationship-management/>
- [5] <https://spring.io/projects/spring-boot>
- [6] <https://docs.spring.io/spring/docs/3.0.0.M3/reference/html/ch01s02.html>
- [7] [https://hu.wikipedia.org/wiki/A\\_f%C3%BCgg%C5%91s%C3%A9g\\_befecskendez%C3%A9se](https://hu.wikipedia.org/wiki/A_f%C3%BCgg%C5%91s%C3%A9g_befecskendez%C3%A9se)

- [8] [https://hu.wikipedia.org/wiki/Apache\\_Maven](https://hu.wikipedia.org/wiki/Apache_Maven)
- [9] <http://maven.apache.org/index.html>
- [10] <https://hu.wikipedia.org/wiki/PostgreSQL>
- [11] <https://www.postgresql.org/>
- [12] <https://hu.wikipedia.org/wiki/Git>
- [13] <https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>
- [14] [https://hu.wikipedia.org/wiki/Java\\_Persistence\\_API#Java\\_Persistence\\_Query\\_Language](https://hu.wikipedia.org/wiki/Java_Persistence_API#Java_Persistence_Query_Language)

## *Nyilatkozat*

Alulírott ..... szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet ..... Tanszékén készítettem, ..... diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat / diplomamunkámat a Szegedi Tudományegyetem Informatikai Intézet könyvtárában, a helyben olvasható könyvek között helyezik el.

Dátum

Aláírás