



# Szoftverfejlesztés Java Enterprise platformon

**Készítette:**

Kosárkó Ákos

Programtervező informatikus Bsc szak

**Témavezető:**

Tajti Tibor

tanársegéd

EGER, 2016

# Tartalomjegyzék

|                                                                                  |           |
|----------------------------------------------------------------------------------|-----------|
| <b>1. Az alkalmazás bemutatása</b>                                               | <b>5</b>  |
| 1.1. Az alkalmazás működtetésének célja . . . . .                                | 5         |
| 1.2. Képernyők . . . . .                                                         | 5         |
| 1.2.1. Főoldal . . . . .                                                         | 6         |
| 1.2.2. Regisztrációs oldal . . . . .                                             | 6         |
| 1.2.3. Bemutatkozás . . . . .                                                    | 6         |
| 1.2.4. Szolgáltatások . . . . .                                                  | 6         |
| 1.2.5. Kapcsolat . . . . .                                                       | 7         |
| 1.2.6. Számlák . . . . .                                                         | 7         |
| 1.2.7. Hibabejelentés . . . . .                                                  | 7         |
| 1.2.8. Bejelentett hibák . . . . .                                               | 7         |
| 1.3. A szoftver elkészítésére használt technológiák, és az azt támogató eszközök | 7         |
| 1.3.1. Java Enterprise Edition . . . . .                                         | 8         |
| 1.3.2. WildFly Application Server . . . . .                                      | 8         |
| 1.3.3. Oracle Database 11g Express Edition . . . . .                             | 8         |
| 1.3.4. IntelliJ IDEA . . . . .                                                   | 9         |
| 1.3.5. Vaadin Framework . . . . .                                                | 9         |
| 1.3.6. Maven . . . . .                                                           | 9         |
| 1.3.7. Git . . . . .                                                             | 9         |
| <b>2. A Java Enterprise Edition</b>                                              | <b>11</b> |
| 2.1. Mi a Java EE? . . . . .                                                     | 11        |
| 2.2. Arhitektúra . . . . .                                                       | 12        |
| 2.3. Komponensek . . . . .                                                       | 12        |
| 2.3.1. Komponensek bemutatása . . . . .                                          | 12        |
| 2.3.2. Komponensek csomagolása . . . . .                                         | 12        |
| 2.4. Konténerek . . . . .                                                        | 13        |
| 2.4.1. Konténerek leírása . . . . .                                              | 13        |
| 2.4.2. Konténerek által nyújtott szolgáltatások . . . . .                        | 14        |
| 2.5. Annotációk és deployment descriptorok . . . . .                             | 14        |
| 2.5.1. Alszakasz címe . . . . .                                                  | 15        |

|                                        |           |
|----------------------------------------|-----------|
| <b>3. Az alkalmazás</b>                | <b>16</b> |
| 3.1. Az adatbázismodell . . . . .      | 16        |
| 3.2. A projekt összeállítása . . . . . | 18        |
| 3.2.1. Alszakasz címe . . . . .        | 20        |
| <b>4. Összegzés</b>                    | <b>21</b> |
| 4.1. Szakasz címe . . . . .            | 21        |
| 4.1.1. Alszakasz címe . . . . .        | 21        |

# Bevezetés

2015 elején foglalkoztatni kezdett, hogy munkába álljak és pénzt keressek. Ezen célért az interneten böngészve rátaláltam egy iskolaszövetkezet hirdetésére, amely biztos munkát ígért az általuk kínált képzés elvégzése után. A képzés tárgyaként a JAVA EE volt megjelölve, amiről első körben nem tudtam, hogy micsoda, annyit sejtettem, hogy a Java programozási nyelvhez kapcsolódhat. Második nekifutásra utánaolvastam az interneten, az jött le, hogy ez a Java programozási nyelv rengeteg külső osztálykönyvtárral kibővített "változata". Ma már tudom, hogy ez ennél jóval több. Visszatérve az iskolaszövetkezet hirdetésére, kapva az alkalmon, jelentkeztem a hirdetésükre, és a felvételi interjú után alkalmasnak találtak arra, hogy részt vegyek a képzésükön.

Így kerültem kapcsolatba a Java Enterprise Edition technológiával, amely, mint később világossá vált számomra, az iparban széles körben elterjedt és számos helyen használt eszköz. A szakdolgozat elkészítésével és az ehhez írt alkalmazás elkészítésével célom a technológia általam már ismert részének bemutatása, a technológia által nyújtott és számomra új megoldások megtalálása, és a gyakorlás. Mindezeket egy fiktív telekommunikációs cég számára készített alkalmazás, amellyel online ügyintézését tesznek lehetővé a régi és új ügyfeleik számára.

# 1. fejezet

## Az alkalmazás bemutatása

### 1.1. Az alkalmazás működtetésének célja

Az alkalmazás működtetésének célja lehetővé tenni a szolgáltató ügyfelei számára, hogy online intézhessék az ügyeiket. Ezen ügyintézésbe a cég szolgáltatásaira való előfizetés, számlabefizetés és hibabejelentés tartozik, emellett az alkalmazást a szolgáltató ügyintézői szintén használhatják, akik az ügyintézői jogosultságuk miatt más tartalomhoz férnek hozzá, mint az ügyfelek. Ők látják az összes bejelentett hibajegyet, amikhez a hiba elhárításának megoldását fűzik hozzá a hibabejelentés lezárásaként.

### 1.2. Képernyők

Az alkalmazással végezhető műveletek természetesen láthatóak kell legyenek a használók számára, erre a képernyők szolgálnak. Az alkalmazás webes böngészőkön keresztül érhető el a felhasználók számára, a képernyők is abban fognak megjelenni nekik. Az alábbiakban a felhasználók számára megjelenő képernyők leírását adom meg. A képernyők két csoportba bonthatók:

- Az egyik csoportba azok a képernyők tartoznak, amelyek bárki számára (azaz a nem bejelentkezett felhasználók számára is) megjelennek. Az ezeken a képernyőn elérhető információk és műveletek nincsenek egyénekhez kötve, egyedi információkat nem tartalmaznak. Ezzel együtt vannak olyan bárki számára elérhető oldalak, amelyek bejelentkezés után bővebb tartalommal rendelkeznek.
- Ennek ellenkezője áll fenn a bejelentkezett felhasználók számára elérhető oldalak esetén. Ezeken az oldalakon olyan információk és műveletek is elérhetőek, amelyek egyéniak, azaz bejelentkezett felhasználónként különböznek. Ilyen például a számlák képernyő, ahol nyilvánvalóan a felhasználó saját számlai kerülnek megjelenítésre. Ennek adatvédelmi okai vannak: egy előfizető számláit csakis ő maga

láthassa, az alkalmazás többi felhasználója ne. A bejelentkezett felhasználók típusoktól függően (ügyfél illetve ügyintéző) más felületeket érnek el: egy ügyfél hibabejelentő oldalt ér el, míg az ügyintéző a bejelentett hibákat listáztathatja ki, és kezelheti.

### **1.2.1. Főoldal**

Az alkalmazás kezdőoldala a főoldal, amely a bárki számára megjelenő oldalakhoz közé tartozik. Ez az oldal egy rövid üdvözlő szöveget jelenít meg, a bárki számára elérhető menüpontok gombjaival együtt.

### **1.2.2. Regisztrációs oldal**

Az oldalon történő ügyintézéshez egy saját felhasználói fiókra szükséges. Ezt a fiókot a "Regisztrációs oldal"-on hozhatja létre a leendő ügyfél, amelyre a képernyő jobb felső sarkában lévő login-box "Regisztráció" linkre kattintással jut el. A fiók létrehozásához személyes adatainak megadására van szükség, amely a form mezőinek kitöltésével végzendő el. A megadott adatok a regisztráció véglegesítése előtt egy validáción kell hogy keresztül menjenek. A validáció célja jelen esetben az, hogy elkerüljük olyan értékek megadását, amik nem lehetnek valósak, például egy számjegy a névben.

### **1.2.3. Bemutakozás**

A "Bemutakozó oldal"-on a szolgáltató hosszabb bemutatkozása olvasható, amely bővebb információkat ad a működéséről, felépítéséről és az általa nyújtott szolgáltatásokról.

### **1.2.4. Szolgáltatások**

A "Szolgáltatások" felület egy fontos felület a képernyők között. Ezen a felületen találja meg az oldal látogatója a cég által nyújtott szolgáltatások listáját és azok jellemzőit, úgy mint az ár, internetszolgáltatás esetén a sebesség, stb. Mivel a cég többféle szolgáltatást is nyújt (telefon, internet, kábelTV) ezért az oldal áttekinthetősége és a szolgáltatások megjelenítésének struktúrája kiemelkedő fontosságú. Éppen ezért a szolgáltatások listája szolgáltatástípusonként megjelenítve áll rendelkezésre az ügyfelek számára. A bejelentkezett felhasználók itt adhatják hozzá a bevásárlókosarukhoz a szolgáltatásokat, amelyekre elő kívánnak fizetni. A bevásárlókosárba tett szolgáltatások megrendelését a "Megrendel" gomb megnyomásával eszközölheti az ügyfél, ami után a szolgáltatási végpont adatait kell hogy megadja. A megrendelt szolgáltatások az itt megadott címre kerülnek bekötésre.

### **1.2.5. Kapcsolat**

A "Kapcsolat" oldalon a cég elérhetőségei olvashatóak, úgy mint a székhely, ügyfélszolgálati telefonszám és levelezési cím.

### **1.2.6. Számlák**

Ez az oldal az előfizetések megléte miatt kiállított számlákat listázza ki. Az ügyfél minden egyes befizetetlen számláját kilistázásra kerül ezen a felületen, számlanévvvel, összeggel és befizetési határidővel együtt. A számla befizetésére bankkártyás fizetéssel van mód, amelyhez a bankkártyán szereplő adatokat a számlához tartozó befizetés gomb megnyomásával betöltődő oldalon kell megadni. Ezen az oldalon - természetesen egy kitalált, nem létező - pénzügyi szolgáltató felé történő kérés fut le, amely a bankkártyás fizetést szimulálja. Ez a fizetési kísérlet kétféle eredménnyel záródhat: Az ügyfél bankszámláján van elegendő pénz a számla kiegyenlítésére, és ez esetben a számla befizetett státuszba kerül. A másik eset az, amikor az ügyfél bankszámláján nem áll rendelkezésre elegendő pénz a számla befizetésére, így az befizetésre váró számlaként marad a számlák között. Mivel nincs valós pénzügyi szolgáltató a befizetések mögött, ezért a befizetés sikerességét vagy sikertelenségét egy véletlen szám generálással határozom meg, amelynek értéke dönti el, hogy sikeres vagy sikertelen legyen a befizetés. Mindkét esetben újra a "Számlabefizetés" oldal töltődik be újra.

### **1.2.7. Hibabejelentés**

Az aktív szolgáltatással rendelkező felhasználóknak lehetőségük nyílik a szolgáltatással kapcsolatos hibák bejelentésére. Ezt a "Hibabejelentés" oldalon tehetik meg. Az észlelt hiba részleteit egy form kitöltésével adhatják meg, amely form beküldésével az rögzítésre kerül, és az ügyintézők számára láthatóvá válik.

### **1.2.8. Bejelentett hibák**

Ez egy az ügyintézők számára elérhető felület, amin a még ki nem javított hibákról szóló bejelentések érhetőek el. Amennyiben egy hiba kijavításra került, azt az ügyintézők a megoldást eredményező munkálatok rögzítésével nyugtázhatják.

## **1.3. A szoftver elkészítésére használt technológiák, és az azt támogató eszközök**

Egy alkalmazás elkészítése összetett folyamat. Ennek során nem elég a specifikációt ismerni, és az alkalmazott programozási nyelv használatában járatosnak lenni, ezek

mellet számos olyan eszközt érdemes igénybe venni, amely gyorsítja, hatékonyabbá, vagy éppen "biztonságosabbá" teszi az alkalmazás fejlesztését azáltal, hogy a szoftver-készítés folyamat egyes állomásain a forráskód állapotát eltárolja.

### **1.3.1. Java Enterprise Edition**

A szakdolgozatom témája a Java EE technológia bemutatása, így ez az a technológia, amiről részletesen írni fogok. Bevezetésül pár dolgot írok itt a technológiáról.

A Java Enterprise Edition API-k egy halmaza, amely API-k lehetővé teszik nagy skálázhatósággal rendelkező üzleti alkalmazások elkészítését. 13 API van, amelyek a technológia részeiül szolgálnak. Egy alkalmazás elkészítéséhez nem szükséges minden API-t felhasználni. Például abban az esetben, amikor az alkalmazásunkban nincs szükség a kliensek közötti üzenetküldésre, abban az esetben a JMS -Java Message Service - használata nélkülözhető.

Természetesen vannak olyan API-k amelyek használata szinte nélkülözhetetlen a JEE technológia felhasználásával készített alkalmazások során. Ezen API-k az EJB és a JPA API-k, amik az üzleti logika és az adatbázisműveletek implementálására alkalmasak.

Az általam készített alkalmazás egy olyan alkalmazás, amelyik az API-k egy részét fogja használni.

### **1.3.2. WildFly Application Server**

Az elkészült alkalmazások futtatásához egy olyan környezet szükséges, amely ismeri és kezeli az alkalmazás által használt API-kat. Ezeket a környezeteket az alkalmazás-szerverek. Feladatuk az alkalmazás számára biztosítani az általa használt API-kat, és erőforrás kezelést, erőforrás (ki)osztást végezni. Ilyen erőforrás (ki)osztás alatt értem például az alkalmazásszerver azon műveleteit, amelynek során az erőforrás konténerekbe annyi EJB példányt tárol, amennyi a kliensoldalról érkező kéréseket (optimálisan) képes kiszolgálni.

Én a WildFly alkalmazásszervert fogom használni, amely a JBoss alkalmazásszerver community verziójának új neve a 8.0-s verzió óta. Ez az alkalmazásszerver alkalmas a Java EE technológiával készült alkalmazások futtatására, mert implementálja annak összes standardját.

### **1.3.3. Oracle Database 11g Express Edition**

Az alkalmazás a felhasználókról, szolgáltatásokról, stb adatokat tárol. Erre az Oracle ingyenesen használható adatbázisát, az Oracle Database 11g Express Edition-t használja.



### 1.3.4. IntelliJ IDEA

A fejlesztői környezetnek az IntelliJ IDEA nevű eszközt választottam. Ezen választásom oka az, hogy tapasztalt programozóktól azt hallottam, hogy a forgalomban lévő fejlesztői környezetek közül ez a tapasztalataik alapján az egyik legjobb. Ezen választás hozománya, hogy egy újabb fejlesztői környezetet ismerek meg, a főiskolai tanulmányaim során használt Microsoft Visual Studio és NetBeans mellett.

### 1.3.5. Vaadin Framework

Az alkalmazás frontend részét Vaadin Framework használatával fogom elkészíteni. Ennek oka az, hogy Java-alapú frontend készítést tesz lehetővé. Ez azt jelenti, hogy a böngészőben megjelenő oldalakat nem HTML használatával kell összeállítani, hanem a framework által nyújtott Java objektumokkal, és az így összeállított oldalak HTML5-ként generálódnak ki, amelyet a böngészők képesek megjeleníteni.

### 1.3.6. Maven

A szoftverfejlesztési iparban jelenleg sok programozó és sok cég van jelen. Előfordul, hogy az egyikük elkészít egy funkcióhalmazt implementáló kódot, és azt mások által felhasználhatóvá teszi. Az is előfordulhat, hogy egy specifikációt, elvbeli működést (pl JPA) több cég is implementál, még hozzá másként, majd kiadja azt másoknak felhasználásra. Ezen esetekben ezeken az implementációkat osztálykönyvtárak formájában érjük el egy központi repositoryban. Amennyiben használni szeretnénk ezen osztálykönyvtárakat, elérhetővé kell tennünk ezeket a projektünk számára, mivel az függ azokról. Ez esetben jön jól a Maven, amellyel a projektünk függőségei könnyen letölthetők az internetről és hozzáadhatók a projektünkhöz. Ehhez mindössze egy megfelelő módon összeállított, projektfüggőségeket leíró fájlra van szükségünk.(pl pom.xml) Ez a fájl tartalmazza a projektben használt függőségek elérési helyét és azonosítóját, így azok egy áttekinthető, struktúrált és kényelmes módon integrálhatóak a projektünkbe.

### 1.3.7. Git

A program elkészítése során nagy segítséget nyújt egy verziókezelő rendszer használata, amely lehetővé teszi a fejlesztés során az alkalmazás aktuális állapotának mentését. Ennek több előnye is van. Egyik előnye az, hogy a fejlesztés egy pontjából több irányba is elágaztathatjuk a fejlesztés további menetét. Ez amiatt hasznos, mert ha két megrendelő hasonló alkalmazást kér, akkor az alkalmazások funkciói közötti átfedéseket egy fejlesztési vonalon kezelhetjük és implementálhatjuk, a különbségeket pedig egy-egy külön ágon "adhatjuk" hozzá az átfedéseket tartalmazó részhez. Másik előnye a verziókezelésnek, hogy amennyiben azt találjuk, hogy az alkalmazás jelenlegi állapota

nem a kívánt állapottal egyenlő, akkor könnyen átállhatunk egy olyan mentett állapot-ra, amelyik egy kívánt állapot. Harmadik előnye, hogy amennyiben osztott verziókezelő rendszert használunk, úgy a forráskódból több példány is van, így az esetben, ha az egyik példány elveszne, úgy tényleges veszteség nem ér minket, mivel a projekt kódja máshol is tárolva van.

Én egy a github-on létrehozott repositoryt és a Git nevű verziókezelő rendszert használom az alkalmazás elkészítése során.

## 2. fejezet

# A Java Enterprise Edition

### 2.1. Mi a Java EE?

A jelen korban gyakori eset, hogy olyan szoftvereket kell készítenünk, amelyek a felhasználók számára rengeteg funkciót biztosítanak, és emellett megbízhatónak és biztonságosnak is kell lenniük. Ahhoz, hogy ezeket az igényeket kielégítsük, és az alkalmazásunk viszonylag rövid idő alatt elkészüljön, az szükséges, hogy a szoftver elkészítéséhez szükséges építőkockát közül ne az összeset mi írjuk meg, vagyis felhasználhassunk olyan eszközöket, építőkockákat, amiket mások már elkészítettek, ezáltal mi már nagyrészt csak ezek összeillesztésére kell, hogy időt és energiát fordítsunk, ami összetett alkalmazások esetében még így is jelentős mennyiségű. Például, teljesen felesleges minden egyes alkalmazásunkban elkészíteni egy típust, ami primitív típusú elemeket tartalmaz, hiszen ezt már másvalaki(k) elkészítették, és a műveleteit optimalizálták. Ezek a listák, mint egy alkalmazás építőkockái. Ez, és sok más építőkocka API-ként a Java Standard Edition-jében található meg összegyűjtve. Ennek mintájára, robosztus méretű alkalmazások elkészítéséhez nagy méretű építőkockákat is elkészítettek, amelyek összetett műveletek elvégzésére alkalmasak. Ezekkel az építőkockákkal néhány 10 sor általunk írt programkóddal elvégezhető egy entitás adatbázisba történő perzisztálása, alkalmazások közötti üzenetküldés, e-mail küldés, és egyéb összetett művelet, amelyeket ha mi kellene, hogy megírjunk, nagy mértékben megnövekedne a fejlesztéshez szükséges időtartam. Ezeket a nagy építőkockákat gyűjtötték össze, mint API-k, amiből létrejött a Java EE. Ez az enterprise edition a standard edition teljes egészét felhasználhatja, illetve ha egy Java EE alkalmazást készítünk, mi is használhatjuk benne a Java SE tartalmát.

## 2.2. Arhitektúra

A Java EE különböző konténerekben implementált specifikációk halmaza. A konténerek Java EE futtatói környezetek, amelyek különböző szolgáltatásokat biztosítanak az általuk kiszolgált komponenseknek. Ilyen szolgáltatások például az életciklus menedzselés, a függőség befecskendezés, és így tovább. Ezek a komponensek jól definiált szerződéseket használnak a Java EE infrastruktúrabeli kommunikációhoz, és más komponensekkel történő kommunikációhoz egyiránt. Deploy-olás (az alkalmazásszerverre történő kihe-lyezés) előtt úgynevezett package-ekbe csomagolandóak, amik típusát és kiterjesztését a célkonténer típusa határozza meg.

Konténerek Kép ide!!!

Az alábbi ábrán a platform négy konténere látható, a köztük lévő kapcsolódási protokollok megjelölésével.

## 2.3. Komponensek

### 2.3.1. Komponensek bemutatása

A platform négy komponenstípust tartalmaz:

- Az "Applet"-ek grafikus felhasználói felülettel(Graphic User Interface - GUI) rendelkező alkalmazások, amelyek webes böngészőkben futnak.
- Az "Application"-ök olyan alkalmazások, amelyek kliens gépeken futnak. Ezek tipikusan GUI-k, amelyeknek a Java EE középső rétegének összes eszközéhez elérése van.
- A "Web Applicaton"-ök, amelyek servletekből, JSP-ekből, JSF-ekből, stb állnak a web konténerben futnak, és a kliensekből érkező kéréseket szolgálják ki.
- Az "Enterprise Application"-öket az EJB konténerben futtatjuk. Ezek az alkalmazások EJB-ekből (Enterprise Java Beans), Java Message Service-ből, Java Transaction API-ből, asszinkron hívásokból, RMI/IIOP-ből állnak. Az EJB-k konténer-menedzselt komponensek, amelyek az üzleti logikát írják le és implementálják. Lokálisan, vagy RMI-n keresztül távolról érhetőek el.

### 2.3.2. Komponensek csomagolása

Ahhoz, hogy egy komponenst kihelyezhessünk a szerverre, először be kell, hogy csomagoljuk egy standard formátumba. A Java SE a Java Archive (jar) fájlokat használja erre, amelyek több fájl (Java osztályok, deployment descriptorok, erőforrások, külső könyvtárak) egyetlen fájlba csomagolásai. Ez a csomagolás ZIP formátum alapú.

Minden egyes modul számára meghatározott, hogy milyen típus csomagolt fájlba kell csomagolni:

- Az application client modul java osztályokat és más erőforrás fájlokat tartalmaz, amiket egy .jar fájlba kell csomagolni.
- Az EJB modulok egy vagy több bean-t tartalmaznak, és esetükben szintén a .jar kiterjesztés a megkövetelt.
- A web application module számos elemet tartalmazhat: servleteket, JSP-eket, JSF-eket, Java-Scriptet, HTML fájlokat, CSS-t, stb. Mindezek egyetlen Web-Archive fájlba kerülnek, ami egy .war kiterjesztésű állomány.
- A negyedik modultípus az enterprise modul, ami bármennyi (akár 0-t is) web modul-t(.war), bármennyi EJB modul(.jar) tartalmazhat, és mindezeket egy Enterprise Archive .ear kiterjesztésű fájlban fogja össze.

## 2.4. Konténerek

### 2.4.1. Konténerek leírása

A Java EE infrastruktúra logikai egységekbe került felosztásra, amiket konténereknek nevezünk. Minden egyes konténernek speciális szerepe van, má-más szolgáltatásokat tesznek elérhető a komponensek számára. A konténerek elrejtik a technikai komplexitást. Attól függően, hogy milyen típusú alkalmazást akarunk fejleszteni, más és más konténer(ek)e)t kell használnunk. Például, ha egy webes alkalmazást fejlesztünk, akkor elegendő egy JSF réteget és egy EJB Lite réteget fejleszteni, amik a web konténerbe deployolva működőképeseek. Négy különböző konténer létezik:

- Az Applet konténer applet komponensek futtatására alkalmas. Homokozó biztonsági modell-t használ, így a kód egy homokozó környezetben fut, és a "homokozóm kívüli játék" nem megengedett. Ez azt jelenti, hogy a konténer megelőzi azt, hogy az applet a lokális gépen lévő erőforrásokhoz -képekhez, fájlokhoz- hozzáférjen.
- Az Application Client Container Java osztályokat, könyvtárakat és egyéb fájlokat tartalmaz, amik ahhoz szükségesek, hogy a Java SE alkalmazás egy EJB konténerbeli vagy egy Web konténerbeli komponenssel képes legyen kommunikálni.
- A Web konténer web komponensek futtatásának és menedzselésének alapjául szolgáló szolgáltatásokat biztosítja. Például ez a konténer felelős az EJB-k példányosításáért, inicializálásáért és a servlet hívásokért. HTTP és HTTPS protokollon keresztül érhető el. Ez az a konténer, amit a web oldalak generálására használunk, amik ezután eljutnak a böngészőbe.

- Az EJB konténer a Java EE alkalmazás üzleti logikáját tartalmazó enterprise beanek futtatásának menedzseléséért felelős. Új példányokat hoz létre belőlük, kezeli az életciklusukat.

### 2.4.2. Konténerek által nyújtott szolgáltatások

A konténerek különféle alapszolgáltatásokat nyújtanak az általuk futtatott komponensek számára. Így a fejlesztés során elég az üzleti logika implementálására fókuszálnunk, a technikai problémákat ezen szolgáltatások orvosolják(pl Java entitásosztály mappelése adatbázisbeli rekorddá). Az, hogy ezen alapszolgáltatások közül melyik érhető el egykonténerben, az konténerenként változik. Néhány szolgáltatás a teljesség igénye nélkül:

- Java Persistence API(JPA): ez egy standard API az objektum relációs leképzéshez(ORM). Az API által definiált Java Persistence Query Language (JPQL) által objektumokat olvashatunk fel az alkalmazás által használt adatbázis(ok)ból.
  - Java Message Service(JMS): Asszinkron üzenetek általi kommunikációt tesz lehetővé komponensek között. Megbízható point-to-point üzenetküldést ad mint például a publish-service modell.
  - Java Naming and Directory Interface(JNDI): Arra használjuk, hogy neveket kössünk objektumokhoz és ezután ezen nevek használatával találjuk meg ezeket az objektumokat egy dictionary-ben(lookup). Ez a lookup használható datasource-ok, bean-ek és más erőforrások esetén.
  - JavaMail: Ezzel az API-val e-mail üzeneteket küldhetünk az alkalmazásunkból.
- kép ide

## 2.5. Annotációk és deployment descriptorok

Ahogy az előbb írtam, a konténerek szolgáltatásokat nyújtanak a komponensek számára. Ezek a szolgáltatások alapbeállításokkal rendelkeznek. Például egy entitás osztály egy String típusú tagváltozója varchar2(255) típusú mezőbe fog leképződni az adatbázisban. Abban az esetben, hogyha ezt meg kívánjuk változtatni, meta-adatként kell megadnunk a forráskódban, vagy egy külső fájlban. Forráskódban egy annotációt használva jelezzük az alapbeállításon való változtatást, amikor pedig egy külső fájlban tesszük ezt meg egy deployment descriptor-t alkalmazunk. Ez a deployment descriptor egy XML fájl, amiben a meta-adatok leírása található. A Java EE 6 óta, amikor is beletettek az annotációkat a platform eszközei közé, az általános trend az annotációk

használata, mivel ez jóval kevesebb írást igényel, illetve a meta-adat azon a helyen kerül rögzítésre, ahol kifejti a hatását, így egyszerű kezelhetőséget nyújt. Hátránya, hogy egy annotációban megadott érték megváltoztatása a kód újra fordítását igényli. Deployment descriptor esetében elegendő az xml fájlban átírni az értéket. A két meta-adat megadási mód közül a deployment descriptor rendelkezik magasabb precedenciával. Tehát ha egy meta-adat értéket annotációval is megadtunk, illetve deployment descriptort is alkalmazunk, de abban más értéket állítottunk be, akkor az felülírja az annotációban megadott értéket, és az fogja kifejteni a hatását.

### **2.5.1. Alszakasz címe**

## 3. fejezet

# Az alkalmazás

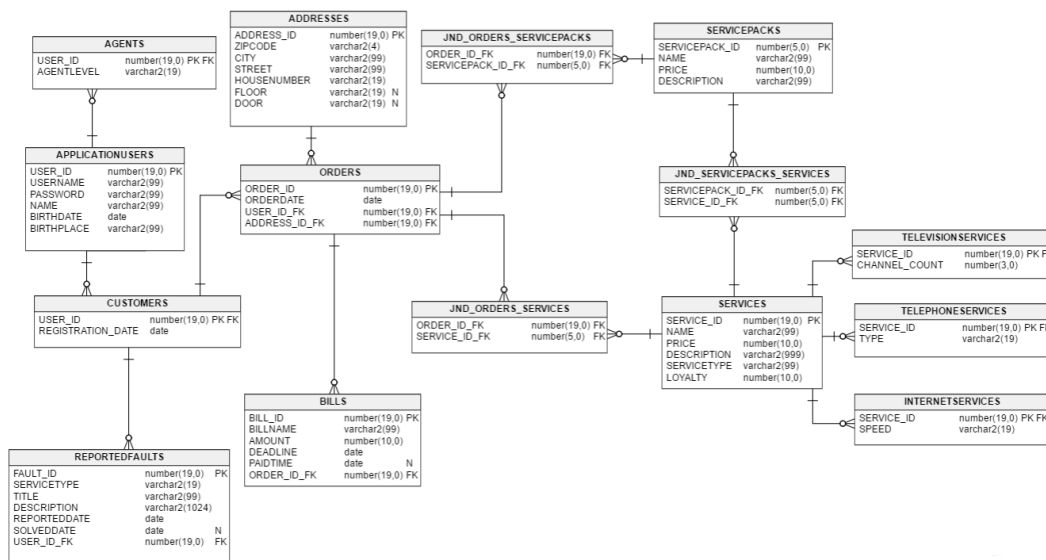
### 3.1. Az adatbázismodell

Az adatbázismodell 15 darab táblából áll. Ezek közül 12 forgalmi tábla. Forgalmi táblának azokat a táblákat nevezzük, amelyek egy entitásra jellemző adatokat tartalmaznak. Például, az általam használt táblák közül az APPLICATIONUSER tábla egy forgalmi tábla, mert egy felhasználó adatait(neve, felhasználóneve, jelszava, stb) tartalmazza. Egy jól megtervezett adatbázismodellben a modell összes táblája közvetlenül kapcsolódik legalább egy másik táblához, és közvetve az összeshez. Ez azt jelenti, hogyha gráfként ábrázoljuk a modellt, amiben a táblák a csúcsok, és két csúcs akkor van összekötve a gráfban, ha az általuk reprezentált táblák között bármilyen(egy-egy, egy-több, több-több) kapcsolat van, és a gráf bármely pontjából vezet út bármely másik pontjába. A forgalmi táblák mellett kapcsolótáblákról is beszélünk. Ezek azok a táblák, amelyek két több-több kapcsolatban álló tábla közé ékelődnek be, méghozzá azért, hogy eltárolják, hogy az egyik tábla mely entitása mely másik táblabeli entitásokhoz kapcsolódik, és fordítva. Az alkalmazásomban által használt adatbázisban egy kapcsolótábla a JND\_ORDERS\_SERVICES. Ez a tábla az ORDERS és a SERVICES tábla között teremt kapcsolatot, melyhez a két oszlopát használja fel. Az ezekben szereplő értékek a idegen kulcsok, melyek elsődleges kulcsok a kapcsolat két oldalán álló táblákban. Egy sorbeli két mezőérték mutatja meg, hogy az egyik tábla mely rekordja a másik tábla mely rekordjához kapcsolódik. Például, ha az ORDER\_ID\_FK mező értéke 9, és a SERVICE\_ID\_FK-é 12, egy másik rekordban pedig 9 és 14, egy harmadikban pedig 10 és 14 ez a két érték, akkor az azt jelenti, hogy a 9-es ID-jű Order rekord a 12 ID-jű, és a 14-es ID-jű Service rekordhoz is kapcsolódik, a 14-s ID-jű Service rekord pedig a 9-es és 10-es ID-jű Orderhez.

A modell ER diagrammja a következő képen látható:

Az adatbázis felhasználókról, bejelentett hibákról, rendelésekről és a rendelésekhez kapcsolódó adategyüttesekről tárol értékeket. Ez utóbbiak a megrendelt szolgáltatások





és/vagy szolgáltatáscsomagok, a rendeléshez tartozó cím amelyre a megrendelt szolgáltatásokat bekötik, és a számlák, amelyek a megrendelés véglegesítése után rögtön le is generálódnak.

A felhasználói adatok az APPLICATIONUSER, az AGENTS és a CUSTOMERS táblában szerepelnek. Az APPLICATIONUSER tábla egy "őstábla", amely mindazokat a mezőket tartalmazza, amik az Agent típusú és a Customer típusú felhasználók leírására szolgálnak. Az Agent-ök egyedi jellemzőit az AGENTS tábla mezői jelentik. A Customer-ök egyedi adatai pedig a CUSTOMERS táblába kerülnek bele. Hogy mi miatt van ez, és hogy miért jó ez, arról egy másik részben fogok írni.

A megrendelések jellemzőit az ORDERS táblában találjuk. A megrendeléshez cím (város, utca, stb) tartozik, amelyek egy a számukra fenntartott táblában – amely az ADDRESSES – kapnak helyet. Ennek oka, hogy amennyiben egy címre több megrendelés is érkezne, elég a címet összeállító adatokat egyszer eltárolni, így elkerüljük az adatok fölöslegesen sokszori eltárolását, és ezáltal gazdaságosabban használjuk fel a tárhelyünket. Egy megrendeléshez nem a cím részleteit (utca, házszám, stb), hanem a cím rekord elsődleges kulcsát tároljuk el idegen kulcsként, amivel hivatkozhatunk rá.

A szolgáltató az általa nyújtott szolgáltatásokból szolgáltatáscsomagokat rak össze. Egy szolgáltatása akár több csomagban is szerepelhet. Nyilvánvalóan egy szolgáltatáscsomag több szolgáltatásból áll. A szolgáltatások közös adatait a SERVICES tábla, a szolgáltatáscsomagok adatait a SERVICEPACKS tábla tárolja. A szolgáltatáscsomagok és a bennük lévő szolgáltatások összerendelését a JND\_SERVICEPACKS\_SERVICES kapcsolótábla tárolja. A különböző típusú (Internet, KábelTV, Telefon) szolgáltatások típusjellemző adatait az INTERNETSERVICES, a TELEVISIONSERVICES és a TELEPHONESERVICES táblákban találjuk. Az elv ugyanaz, mint a felhasználói adatok tárolására használt táblák kialakításánál.

A szolgáltatásokat és szolgáltatáscsomagokat kapcsolótáblák kötik a megrendelésekhez. Ezek a táblák az idegen kulcsokon kívül mást nem tartalmaznak, neveik: JND\_ORDERS\_SERVICES és JND\_ORDERS\_SERVICEPACKS.

A modell utolsó táblája a BILLS nevű tábla. Ezen táblabeli rekordok tartalmazzák a megrendelések után létrejövő számlák adatait. Számlanév, az összeg, amiről a számla szól, a befizetési határidő, amely minden esetben a következő hónap 5-e, és a befizetés ideje. Ezen utolsó mezőt, ami egy Date típusú PAIDTIME nevű mező használatom fel arra, hogy egy számláról megállapíttassam, hogy be van-e már fizetve. Amennyiben ezen mező értéke null, úgy tudjuk, hogy még befizetetlen számláról van szó, ha pedig egy dátum érték szerepel benne, egyértelmű, hogy befizetett. Az előbb leírtak alapján a NOTNULL megszorítást erre a mezőre nem raktam rá. A rekord ORDER\_ID\_FK idegen kulcsa elárulja, hogy melyik megrendelésről szól a számla.

## 3.2. A projekt összeállítása

A projekt vázának egy Vaadin archetípust<sup>1</sup>, nevezetesen a vaadin-archetype-application archetípust használok. Ez az archetípus egy kiindulási alapot szolgáltat, amelyhez az EJB-k, entitások és egyéb Java objektumok, illetve a konfigurációs fájlok, mint például a persistence.xml, hozzáadása az alkalmazás elkészüléséhez vezet. Ahhoz, hogy ezen archetype alapján egyszerűen létrehozzam a projektet, egy új projektet hozok létre, amely egy maven-es projekt, és a projekt létrehozása során megadom, hogy archetype alapján jöjjön létre a projekt. Ehhez meg kell adni az archetype jellemzőit: groupId, artifactId, és a verzió. Mindezek a Vaadin honlapján elérhetők. Jelen esetben ezek a következők: groupId: com.vaadin, artifactId:vaadin-archetype-application version: 7.6.2 . Az archetype kiválasztása után a saját alkalmazásunk groupId-ját és artifactID-ját kell megadnunk. Én groupId-nak a com.szakdolgozat, artifactId-nak pedig a szer-t választottam. Ezen adatok megadása után az IDE letölti a maven központi repository-jából a archetype-ot, mint függőséget, és ez alapján legenerálja a projektet. Ezután rendelkezésünkre áll egy kiindulási projektterv, ami mindössze egy UI java objektumot, egy gwt.xml-t és néhány a kinézetre(forntend) vonatkozó .scss-t tartalmaz, egy struktúrált mappaszerkezetben.

Ezután az alkalmazás IDE-ből deployolhatóságát állítottam be. Ehhez egy plugin-t használok, amely alkalmazáserver specifikus. Ez a plugin mavenrel könnyen hozzáadható az alkalmazásunkhoz. A

### 3.1. kód. WildFly plugin

```
1 <plugin>
2   <groupId>org.wildfly.plugins</groupId>
```

---

<sup>1</sup><https://vaadin.com/maven#dependencies>

```

3      <artifactId>wildfly-maven-plugin</artifactId>
4      <version>1.1.0.Alpha6</version>
5      <configuration>
6          <port>10000</port>
7          <name>szer.war</name>
8      </configuration>
9 </plugin>

```

sorok pom.xml-hez hozzáadásával a plugin letöltésre kerül, és a projekt részévé válik. Ezután a Maven toolboxban megjelenik a pluginok között, és a wildfly:deploy opció indításával máris deployolásra kerül az alkalmazáserverünkre. A port és name konfigurációs attribútumok beállításával finomhangolhatjuk a deployolás tulajdonságait. Jelen esetben a 10000-es porton futó administration consol-ú serverre fog történni a deploy - melynél jelen esetben maga a server a 8090-es porton fut - és szer.war néven fog az alkalmazás deployálásra kerülni. Ez utóbbi azt vonja magával, hogy az alkalmazást a localhost:8090/szer URL-en fogjuk elérni.

Ezt követően az első lépés az adatbázis elérést lehetővé tévő persistence.xml fájl hozzáadása a projekthez, amelyet az src mappa alatt néhány mélységben található resources almappában létrehozandó META-INF mappában kell elhelyezni. Ez az xml file tartalmazza az adatbázis eléréséhez szükséges információkat. Jelen esetben a DataSource nevét, amely az adatbáziskapcsolat kialakításához szükséges információkat tárolja: az adatbázis címe, a bejelentkezéshez szükséges felhasználónév és jelszó. Emellett ami számunkra fontos ebben az xml-ben, az a table generation strategy. Egy jól megválasztott stratégiával gondoskodhatunk az adatok biztonságáról. Én az update értéket választottam. Ez a stratégia az alkalmazás indulásakor megvizsgálja az adatbázis séma tartalmát, és amennyiben nem talál benne a projektben definiált entitás számára táblát, létrehozza azt. Abban az esetben pedig, amikor egy entitásosztály tagváltozóinak típusa és/vagy a tagváltozókhoz beállított adatbázisbeli jellemzők az adatbázisbeli táblafelépítéssel nincsenek szinkronban, úgy a tábla módosításra kerül, hogy annak jellemzői az entitásosztályban beállított jellemzőket tükrözzék. Például, ha egy Java osztály, ami egy entitást ír le rendelkezik egy int típusú és mennyiség nevű tagváltozóval, aminek az entitáshoz tartozó táblában nincs megfelelő mező, úgy a tábla egy "ALTER TABLE ..." DDL utasítás automatikus lefutásával módosul, és bővül egy mennyiség nevű mezővel.

Az ORM műveleteket a Hibernate, egy JPA implementáció fogja végezni. Ez az entitymanager-en keresztül goldozik. Ahhoz hogy ezt használni lehessen, függőségként hozzá kel adnunk a projektünkhöz. Ezt a pom.xml szerkesztésével tehetjük meg:

### 3.2. kód. Hibernate függőségek

```

1 <dependency>
2     <groupId>org.hibernate</groupId>
3     <artifactId>hibernate-core</artifactId>

```

```

4         <version>4.0.1.Final</version>
5     </dependency>
6     <dependency>
7         <groupId>org.hibernate</groupId>
8         <artifactId>hibernate-validator</artifactId>
9         <version>4.2.0.Final</version>
10    </dependency>
11    <dependency>
12        <groupId>org.hibernate.common</groupId>
13        <artifactId>hibernate-commons-annotations</
14            artifactId>
15        <version>4.0.1.Final</version>
16        <classifier>tests</classifier>
17    </dependency>
18    <dependency>
19        <groupId>org.hibernate.javax.persistence</
20            groupId>
21        <artifactId>hibernate-jpa-2.0-api</artifactId>
22        <version>1.0.1.Final</version>
23    </dependency>
24    <dependency>
25        <groupId>org.hibernate</groupId>
26        <artifactId>hibernate-entitymanager</
27            artifactId>
28        <version>4.0.1.Final</version>
29        <exclusions>
30            <exclusion>
31                <groupId>dom4j</groupId>
32                <artifactId>dom4j</artifactId>
33            </exclusion>
34        </exclusions>
35    </dependency>

```

A fenti kód alapján a maven a központi repositoryból letölti a szükséges JAR-okat, és hozzáadja a projektünkhöz. Így már fogjuk tudni használni a Hibernate-t, amivel az objektumorientált programunk entitásainak világa és a "rekord orientált" adatbázisunk rekordjainak világa közötti átjárást tesszük lehetővé.

### 3.2.1. Alszakasz címe

## 4. fejezet

# Összegzés

### 4.1. Szakasz címe

#### 4.1.1. Alszakasz címe

# Irodalomjegyzék

[1] SZERZŐ: Cím, Kiadó, Hely, évszám.