SZAKDOLGOZAT

Nagy Zsolt

Debrecen
2009

Debreceni Egyetem

Informatika Kar

Java alkalmazások

Témavezető	Készítette:
Espák Miklós	Nagy Zsolt
Egyetemi Tanársegéd	Informatika – matematika

Debrecen

2009

Tartalomjegyzék

1. Bevezetés ... 4
2. Felépítés ... 6
  2.1. Üzleti logika ... 6
  2.2. A program funkciói, működése ... 8
  2.3. Alkalmazott eszközök ... 9
    2.3.1. Fejlesztői környezet ... 9
    2.3.2. Az adatbázisszerver ... 17
    2.3.3. Az alkalmazásszerver ... 25
    2.3.4. A megjelenítés ... 33
    2.3.5. A perzisztens réteg ... 35
  2.4. Fejlesztési Fázisok ... 36
    2.4.1. Adatbázis táblák létrehozása ... 36
    2.4.2. Adatbázis táblák reprezentálása kód szinten ... 39
    2.4.3. Üzleti logikai rész leprogramozása ... 40
    2.4.4. Felhasználói felület (GUI) és mögöttes kód kialakítása ... 42
  2.5. A rendszer felállása ... 46
3. Összegzés ... 55
4. Irodalomjegyzék ... 56

1. Bevezetés

Szakdolgozatom témája a Java alkalmazások fejlesztésének általános menete, körülményeinek egy konkrét példán keresztüli bemutatása. Az ehhez szükséges eszközök tulajdonságainak és szerepének áttekintése. Témaválasztásom több tényezőnek is köszönhető.

Egyrészt, mindenképp olyan témába szerettem volna mélyebbre ásni magam, ami modern, az informatika területén mondhatni mindennapi. A rendszerfejlesztést ebbe a kategóriába sorolom, hiszen informatikai rendszerek hálózzák be környezetünket. Például vehetjük az oktatási minisztérium komplett és összetett rendszerét, melyet minden, az oktatási folyamatban részt vevő intézmény elér, és a saját azonosítójával az ehhez a profilhoz beállított jogosultsági szinttel képes operálni az adatokon, központi statisztikáktól kezdve a legegyszerűbb adatmódosításig, szolgáltatások széles körét tárja a felhasználók elé. Az állami rendelkezésben lévő nyilvántartó alkalmazások, az orvosi rendszerek, az infrastruktúránkat irányító rendszerek is mind hasonló példák. De nyílván ezek felépítése nagyon bonyolult. Elég a kisebb vállalatok irányító és nyilvántartó rendszereire is gondolni. Ez az irány az informatikában bár már nem éli virágkorát – hisz az általánosabb funkciójú sablon alkalmazások kielégíthetik egy nem túlságosan specializált igénykör szükségleteit, és ezekből a termékekből nagy választék van a piacon viszonylag elérhető áron – mégis az ehhez szükséges eszközök újabbnál újabb típusai jelennek meg, a választék egyre bővül. Ezek a tendenciák sarkallt arra, hogy közelebbről megvizsgáljam a webes alkalmazások fejlesztésének menetét és körülményeit.

Emellett a jövőbeli céljaimat is szem előtt tartottam. Matematika-informatika pedagógia szakon hallgatok mellyel, és ez a sajnálatos nagy átlag, nem pedagógiai törekvéseim vannak, de ez egy másik dolgozat témája. A jövőben mindenképp szeretnék egy olyan munkahelyet találni, ahol az egyetemen tanult ismeretek jó alapot nyújtanak a szakmai fejlődéshez és egy valamire való szakmai karrier befutásához. A Hajdú-Bihar megyében található munkalehetőségeket informatikai végzettségűek számára előnyösebbnek tartom más szakmabeliek lehetőségeinél. A maszekok eltűnésének és a multinacionális vállalatok térnyerésének következtében egyre inkább nyugat-európai színvonalú az ebben a megyében folyó informatikai szolgáltatás, és ezen ipar térnyerésének köszönhetően ezt a tendenciát egyre inkább javulónak vélem. Az egyetemi tanulmányok, bár remek alapot nyújtanak a kezdeti megpróbáltatásokhoz és a különböző informatikai területek ismereteinek

elsajátításához, erre a színvonalú munkára nem készítenek fel. Ezen okokból kifolyólag azt gondoltam, hogy a pályakezdési céljaimat mindenképp támogatná egy ilyen témájú szakdolgozat és ismeretbővítés.

A szakdolgozati témámmal a célom az volt, hogy egy általánosabb rálátást nyerjek a webes fejlesztések menetére. Főként az előkészületek és a fejlesztést átfogó struktúra, mint irányelv áttekintésére törekedtem. Egy konkrét példán keresztül mutatom be a megvalósítás menetét, fázisainak vizsgálata így szemléletesebb, mint csupán elméleti keretek között maradni.

2. Felépítés

Az alkalmazásfejlesztés főbb lépései és struktúrája a következők

2.1. Üzleti logika

Ugyanakkor az Egyetem mellett végzett munkám is erre a döntésemre vezetett. Egy oktató és tanácsadó cégnél végeztem csoportvezetői és egyéb feladatokat. A cégvezető – aki szintén jártas informatikában, és tisztában van egy ilyen rendszer előnyeivel – kért fel egy társaságot, hogy készítsenek neki egy vállalati nyilvántartási alkalmazást. Különösképp távolabbi elképzelései miatt, ugyanis országos hálózatot kíván kiépíteni, ahol elengedhetetlen egy átfogó, a vállalat munkáját szemléltető program. Ekkor döntöttem el, hogy megpróbálok egy hasonló fejlesztést létrehozni én is önállóan.

Ennek a rendszernek a célja, hogy egy jól ellenőrizhető nyilvántartási rendszert nyújtson a vállalati hierarchiát modellezve:

- On-line webes felülettel rendelkezzen, hisz bárhonnan – Internet kapcsolattól függően– el kell tudni érni, ezzel szolgálva a mobilitást.
- Űrlapos feltöltést tegyen lehetővé, hisz az alkalmazást nem informatikusok, hanem általános informatikai intelligenciával rendelkező alkalmazottak fogják használni.
- Adatbázis támogatással rendelkezzen, mivel a vállalati adatokat tudni kell tárolni és elérni, azok közt megfelelő szelektálásokat végrehajtani.

Egy ilyen rendszer nagyban meg tudja könnyíteni egy vállalat munkáját, ezt testközelből is tapasztalhattam. Legtöbbször az ellenőrzésen bukik meg a hatékonyság, bármilyen szolgáltatást nyújtó vállalkozás alkalmazotti teljesítményét tekintjük is. Egy olyan vállalatnál, ahol az alkalmazottak hierarchia szinteket valósítanak meg – azaz az irányítás réteges, elhatárolt, egyes pozíciókról mégis átfogó, az egész vállalati működést, mint egységet kezeli – mindenképp elengedhetetlen. Ebben az esetben, de ez a zubbony sok intézményre adoptálható, nagyon fontos az átfogó statisztikák és az egyéni teljesítmények összehasonlítása, ellenőrzése.

A munkahelyi hierarchia a következőképp néz ki:

A cég *tulajdonosát* most, mint szakmai segítőt tekintem, aki a vállalat aktív munkájában nem vesz részt, de ellenőrzést végezhet, számon kérheti a vállalat bármely dolgozójának munkáját, eredményeit, tevékenységét. Így Ő a hierarchiában nem kap szerepet.

Az *ügyvezető* szerepe a cég/vállalat irányítása, annak koordinálása, a különböző területi régiók összefogása, azok munkájának irányítása. Az ügyvezető, mint láncszem a hálózat vezetőjével áll kapcsolatban. Minden régiónak van ügyvezetője.

A különböző régiók, mint egymástól elszeparált területek vezetője a *hálózatvezető*, aki az adott kommuna vezetésével van megbízva, közvetlen kapcsolatot képvisel a részlege, mint önállóan működő vállalat, és az *ügyvezető* között. A *hálózatvezető* továbbá közvetett kapcsolatot jelent az alkalmazottjai és az *ügyvezető* között, a hozzá érkező, de a vállalat működésére vonatkozó elvárásokat ő továbbítja a megfelelő láncszemnek.

A *hálózatvezető* emellett követlen kapcsolatban áll azon *csoportok vezetőivel*, akik a vállalat működésében elválasztható, elkülönült feladatkörrel rendelkeznek. Ezen csoportok egymástól kvázi függetlenül tevékenykednek/tevékenykedhetnek, leszámítva hogy a vállalat működését a különálló csoportok együttes munkája szolgáltatja. A csoportok tehát a vállalat kisebb szegmensét, a vállalati fürt egy-egy részfürtjeit reprezentálják. Hasonló hierarchia épül fel a *csoportvezetők* és az - alkalmazotti szempontból a vállalati hierarchia legalsó lépcsőfokát képző – un. *ajánló partnerek* között.

Ezen felépítés alapján az *ajánló partnerek* közvetlen kapcsolatban állnak saját *csoportvezetőjükkel*, közvetett kapcsolatban *hálózatuk vezetőjével* és a *tulajdonossal* is.

A hierarchia még nem teljes, ugyanis szerepe van a vállalat működéséhez elengedhetetlen *ügyfelek* definiálására is. Ügyfél az, aki kapcsolatba lép a vállalat bármely tagjával, és a vállalat szolgáltatásait valamilyen formában igénybe veszi. Az ügyfél és a vállalat kapcsolatba kerülésekor közreműködő munkatárs kiemelt szerepet játszik a további szolgáltatások véghezvitelének folyamatában. Ebből kifolyólag a vállalat bármely alkalmazottja rendelkezhet saját ügyféllel, még a hálózatvezető is. Így az ügyfelek bármely hierarchiaszinten megjelenhetnek. Az ügyfeleknek állapota is van, ami az aktivitását vagy passzivitását jelöli meg. Ennek szintén üzleti funkciója van (Pl. a passzív ügyfelek nem részesülnek az aktuális akciókat tartalmazó körlevelekből). Szakdolgozatom célja nem egy komplett rendszer összeállítása, hanem egy általánosabb szféra – a fejlesztés körülményeinek megragadása és elemzése, így a kódrészletbe az ügyfeleket nem szándékozom beletenni.

Összegezve a hálózati hierarchiát:

Ügyvezető - Hálózatvezető – Csoportvezető – Ajánló Partner.

A munkafolyamatot, mely a rendszer működése szempontjából ebben a vázlatszerű megvalósításban nem fontos, nem részletezem. A tevékenység egyetlen fontos fázisa, amelyre a rendszer is épül, az ellenőrzés. Meghatározó momentum, hogy a tulajdonos képes legyen a hálózatok összesített munkásságát szemügyre venni, vagy külön bármely alkalmazott adatait elkérni, teljesítményét, statisztikáit ellenőrizni. E követelményekből áll össze a rendszer, a szükséges tevékenységek és a megfelelő sajátosságok.

Ezt az igényt a következő felépítésű program elégíti ki:

2.2. A program funkciói, működése

A program funkciói és működése a következő pontokban foglalható össze:

Képes legyen az alkalmazottak adatait tárolni, azokat a munkahelyi hierarchia megfelelő helyére beilleszteni.

Bejelentkezési felületet, egyéni felhasználói nevet és jelszót biztosítson, a bejelentkezés „állapotát" figyelje, biztonságos használatot szolgáltatva.

Kezelje az egyes munkatársak jogköreit a korábban szemléltetett módon. Láthatósági, jogosultsági szinteket állítson be, korlátozzon bizonyos műveleteket, ezzel biztosítva az adatvédelmet és a lehetőséget az egyes feladatkörök elvégzéséhez szükséges folyamatokhoz. Módosíthatóak legyenek a saját adatok mellett a hierarchiának megfelelő alacsonyabb szinteken lévő munkatársak adatai is. Listázni szintén csak hierarchiának megfelelően „lefelé" lehetséges.

A fejlesztés szempontjából mellékes tényező, de mégis szükségesnek tartom megjegyezni a rendszer fizikai szükségleteit. Mivel szerveroldali alkalmazásról van szó, az alkalmazás a műveleteket a szerveren lévő erőforrásokon végzi el. Így egy valós esetben mindenképp szükséges egy szerver bérlése, hogy a működéshez elengedhetetlen programokat azon futtatni tudjuk, ez által biztosítva a felhasználók kiszolgálását. Ezek a programok az alkalmazásszerver, az adatbázisszerver, a megjelenítésért felelős interfész, a futtató környezet és a kiszolgáló. Védelme szempontjából viszont nem biztonságos az adatbázist a többi alkalmazással egy helyen tárolni, ajánlatos egy külön, az alkalmazás helyétől fizikailag különálló, tűzfallal védett és egy adott fix IP címről – az alkalmazás IP-jéről – elérhető gépen

elhelyezni azt. Az alkalmazás esetleges feltörésekor az általunk tárolt, bizonyára értékes adatok nem kerülnek veszélybe, vagy illetéktelenek kezébe.

Mindezen szempontokat figyelembe véve a fejlesztés első fázisa az üzleti logika kiépítését és az igények felderítését követően a szükséges eszközök meghatározása.

2.3. Alkalmazott eszközök

2.3.1. Fejlesztői környezet

A fejlesztői környezet szerepe a következő: a konkrét fejlesztés ezen történik, erre épül az összes eszköz használata. Korábbi tanulmányaim során a NetBeans fejlesztői környezetet használtam. Több beépített modulja és falhasználó barát felülete hamar elsajátítható. Most mégis az Eclipse környezetet részesítem előnyben, az utóbbi időben ezt használtam gyakrabban.

Eclipse

„Az Eclipse platform bárminek lehet az integrált fejlesztői környezete (Integrated Development Environment (IDE), de nem kifejezetten egy termékre van specializálva.

Habár az Eclipse platform sok beépített funkcionalitással rendelkezik, ezek közül a legtöbb csak általános. További olyan, a platform kiterjesztéséhez elengedhetetlen eszközökre volt szükség, hogy új összetett típusokkal lehessen dolgozni, már létező összetett típusokkal új dolgokat lehessen végrehajtani, és az általános funkcionalitást egy specifikusabbra lehessen összpontosítani.

Az Eclipse platform felfedező, integráló, futtató modulok mechanizmusára épül, melyet bővítménynek (plug-in) nevezünk.

Az eszköz kiszolgáló ír egy eszközt, mint különálló bővítményt, mely a munkaterület (Workspace) fájljain operál, és az eszköz specifikus felhasználói felületet (User Interface - UI) a munkalapon (Workbench) alakítja.

A platform felállásakor a felhasználó egy elérhető bővítményekből álló IDE-vel találja magát szemben.

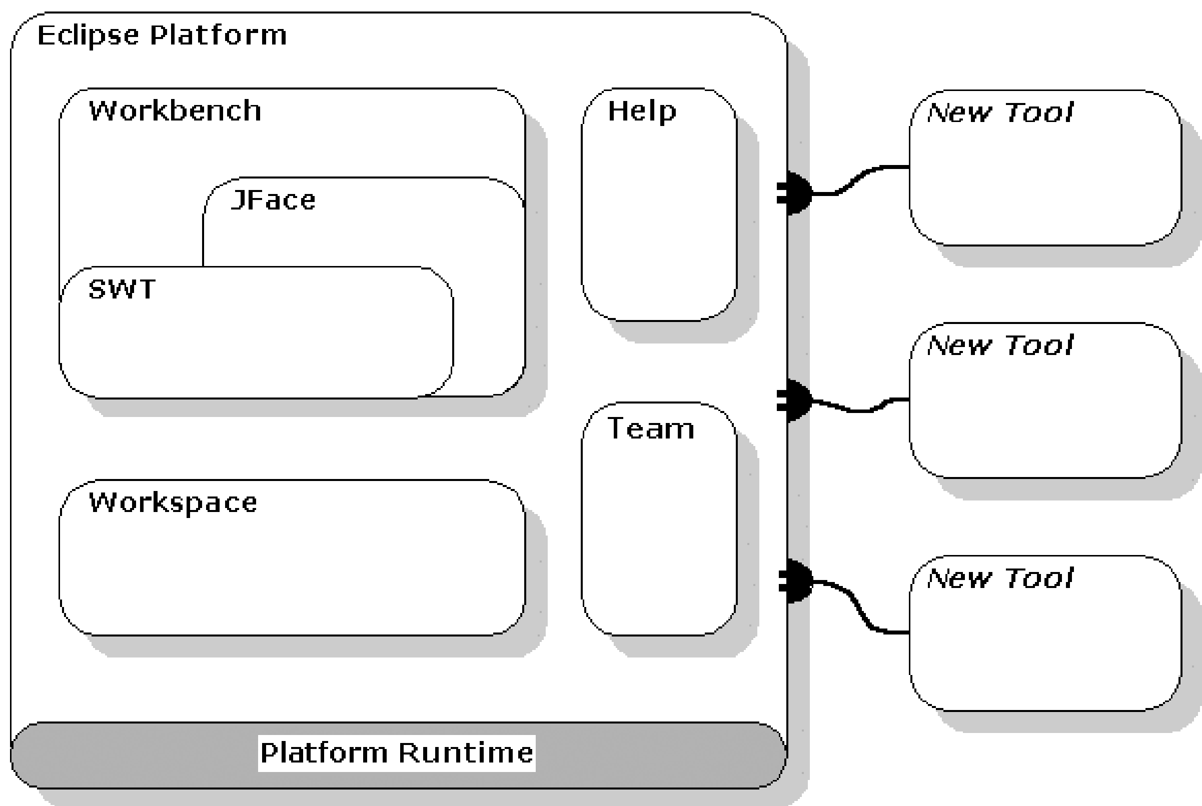
A felhasználói tapasztalat minősége jelentősen függ attól, hogy milyen jól vannak az eszközök a platformba integrálva, és hogy mennyire dolgoznak jól együtt a különböző eszközök.

Eclipse platform technikai áttekintése

Az Eclipse Platform (vagy egyszerűen csak „a Platform”, amennyiben nincs névütközési kockázat) az alábbi követelmények kielégítésére lett tervezve és építve:

- Alkalmazásfejlesztéshez szükséges különböző összetevők kivitelezésének támogatása
- Eszközkiszolgálók számára korlátlan beállítások támogatása
- Tetszőleges összetevőjű típusok kezeléséhez szükséges eszközök támogatása (pl.: HTML, Java, C, JSP, EJB, XML, és GIF)
- Elősegíti az eszközök integrálását különböző összetett típusokon és eszköz kiszolgálókon belül és keresztül
- Mind a GUI és nem GUI alapú alkalmazás fejlesztési környezetének támogatása
- Széles körű operációs rendszereken való futás, beleértve a Windows és a Linux rendszereket
- Kihasználja a Java programozási nyelvhez használt leíró eszközök népszerűségét

Az Eclipse Platform elsődleges feladata, hogy használható mechanizmusokkal rendelkező eszköz kiszolgálót nyújtson, és olyan követendő szabályokat biztosítson, mely teljességgel integrált eszközökhöz vezet. Jól definiált API-n, (Application Programming Interface - felhasználói program interfész) osztályokon és metódusokon keresztül világítja meg ezeket a mechanizmusokat. A Platform használható keretrendszert és építő elemeket biztosít az új eszközök fejlesztésének megkönnyítése érdekében.



Az Eclipse Platform szerkezete: főbb összetevői

Platform futási ideje és beépülő-modul szerkezete

A beépülő-modul az Eclipse Platform legkisebb egysége, mely fejleszthető és szállítható is külön-külön. Általában egy kis eszközt írnak, mint egyedüli beépülő-modult, míg egy összetett eszköz funkcionalitása több különböző bővítményen keresztül tevődik össze. Kivéve egy kis Kernelt, ismertebb nevén Platform Runtime-ot, minden egyéb Eclipse Platform funkcionalitása beépített-modulokban található meg.

A bővítmények Java-ban vannak kódolva. Egy tipikus bővítmény JAR könyvtárbeli Java kódból, néhány kizárólag olvasható fájlból, és egyéb forrásokból (képek, web sablonok, üzenet katalógusok, natív kód könyvtárakból, stb..) tevődik össze. Egyes beépített könyvtárak egyáltalán nem tartalmaznak kódot. Ilyen példa egy olyan bővítmény, mely csak és kizárólag on-line segítséget tartalmaz HTML formájában. Egyes bővítmények kód könyvtárai és kizárólag olvasható tartalmai a fájl rendszerben egy könyvtárban találhatóak, vagy a szerveren egy URI által. Létezik egy mechanizmus, mely által különböző szeletekből a saját könyvtárukból vagy URI-k által bővítményekké egyesíthetők. Ez a mechanizmus szolgáltatott

különböző nyelvi csomagokat a nemzetközi bővítményeknek. Minden bővítmény rendelkezik egy Manifest fájllal, mely más bővítményekkel való kapcsolatát mutatja meg. A kapcsolati modell egyszerű: egy bővítmény tetszőleges számú kiterjesztési pontot nevez meg, és tetszőleges számú kiterjesztés lehet egy vagy több kiterjesztési ponttal más bővítményekben. Egy bővítmény kiterjesztési pontja más bővítmények által is kiterjeszthető. Például a Workbench bővítménye a felhasználói beállítások (User Preferences) számára deklarál kiterjesztési pontot. Bármely bővítmény hozzájárul saját felhasználói beállításaihoz a kiterjesztés pontja kibővítésével. Egy kiterjesztési pont rendelkezhet megfelelő API-val. Más bővítmény implementációkat szolgáltat ezen interfész számára a kiterjesztési pont bővítésén keresztül. Bármely bővítmény szabadon definiálhat új kiterjesztési pontokat, és új API-kat szolgáltathat más bővítmények használatára. Platformindítás (Start-up) esetében a Platform Runtime feltárja az elérhető bővítményeket, beolvassa a Manifest fájlokat, és beépít a memóriába egy bővítménykezelőt (plug-in registry). A Platform a kiterjesztési pontokat nevük alapján illeszti össze a megfelelő bővítmény-deklarációkkal. Bármely probléma, mint pl. a kiterjesztési pont hiánya, észlelhető és naplózható. Az ebből származó bővítménykezelő a platformon keresztül érhető el, bővítmények hozzáadása nem lehetséges start-up után. A bővítmény Manifest fájlja tartalmaz XML-t. Egy kiterjesztési pont deklarálhat specializált XML elemi típusokat a bővítmények használatához. Ez által a bővítményt támogató bővítmények tetszőleges számú információt közölhetnek a megfelelő kiterjesztési pontot deklaráló modul számára. Mi több, a Manifest információja a bővítménykezelőből is elérhető az ehhez tartozó modul aktiválása, vagy bármilyen ehhez szükséges kód betöltése nélkül.

Ezen tulajdonság kulcsfontosságú a nagy mennyiségű installált bővítmények támogatásához, habár ezekből csak néhány szükséges az adott felhasználói munkafolyamathoz. A bővítmény kód betöltése egy elhanyagolható memória lenyomatot eredményez, és hatással van a betöltési időre. Az XML alapú bővítmény Manifest használata szintén megkönnyíti az eszközök írását, mely a bővítmény létrehozását támogatja. A Plug-In Development Environment (PDE - bővítmény fejlesztési környezet) - egy ilyen eszköz, beleértve az Eclipse SDK-t.

A bővítmény a hozzá tartozó kód szükséges futtatásakor aktiválódik. Aktiválása után a bővítmény a bővítménykezelőt használja a közreműködő beépülő modulok kiterjesztési pontjainak felkutatásához és eléréséhez.

A bővítmény deklarálja a felhasználói beállítások kiterjesztési pontját annak érdekében, hogy az összes hozzá tartozó felhasználói beállítást felfedezze, és hogy hozzáférjen a megjelenített nevekhez, ez által szerkesztve egy beállítási dialógust. Ezt a regiszterből származó információk használatával is megtehetjük anélkül, hogy bármilyen ehhez tartozó bővítményt aktiválnánk. A hozzáadott bővítmény akkor aktiválódik, amikor a felhasználó kiválasztja a megfelelő beállítást.

A bővítmény aktiválása nem történik automatikusan, de létezik néhány olyan API metódus, mely explicite aktivál bővítményeket. A bővítmény aktiválását követően az is marad a Platform leállításáig. Minden bővítmény fel van szerelve egy alkönyvtárral, mely tárolja a bővítmény specifikus adatokat; ez a mechanizmus teszi lehetővé a bővítmény számára a legfontosabb státuszok szállítását futtatások között. Tehát a Platform Runtime egy speciális kiterjesztési pontot deklarál az alkalmazások számára. Platform indításkor az alkalmazás nevét a parancssor határozza meg, az alkalmazást deklaráló bővítmény az egyetlen, mely inicializáláskor aktiválódik.

Meghatározva rögtön az elején az elérhető bővítményeket, és támogatva az alapvető jelentős információcseréket a bővítmények között – a bővítmények aktiválása nélkül – a Platform biztosíthat minden, a szövegkörnyezetben levő helyes információforrást, mely megmondja, hogy mely szövegkörnyezetben fog operálni. Ez a szövegkörnyezet a futás alatt nem változtatható, annak változtatásakor nincs szükség a bővítmények tájékoztatására szolgáló komplex életciklusú eseményre. Így elkerülendő a hosszadalmas betöltési idő csakúgy, mint a hibák gyakori forrása, a megjósolhatatlan bővítményaktiválási sorrendből származó leállás/fennakadás.

Az Eclipse számára kizárólag a standard Java Virtual Machine (Java virtuális gép - JVM) – szükséges. Minden bővítménynek létezik egy saját Java osztálybetöltője (Classloader), mely egyedül az osztályok betöltéséért felelős. Minden bővítmény explicite deklarál egy másik bővítménytől való függőséget, melyen keresztül közvetlen osztályelérést kap. A bővítménykezelő kontrollálja a nyilvános osztályok láthatóságát, és az interfészeket a könyvtáraiban. Ez az információ a bővítmény Manifest fájljában deklarálódik, a láthatósági szabályok a bővítmények Classloader-e által vannak kényszeríve futási idő alatt. A bővítmény mechanizmus az Eclipse platform felosztását szolgálja. Valójában, a különböző bővítmények

alkotják a Workspace-t, a Workbench-et stb. Még a Platform Runtime is rendelkezik saját bővítménnyel.

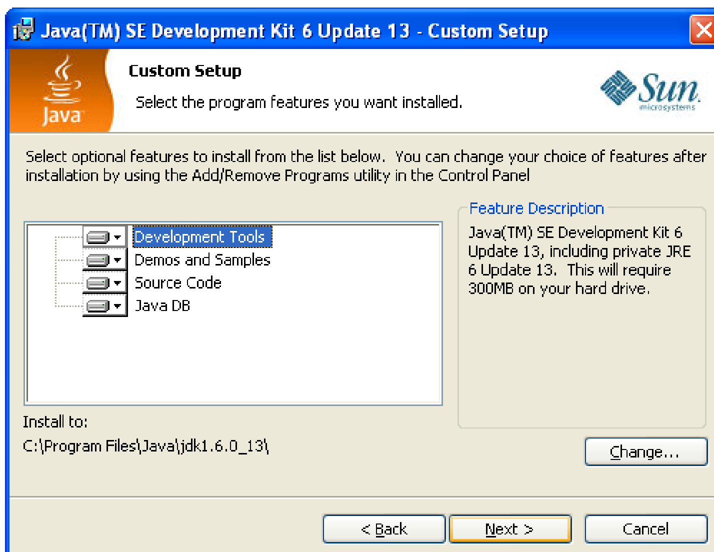
Az Eclipse Platform frissítési menedzsere (update manager) letölt és installál új tulajdonságokat vagy egy javított (upgraded) verzióját egy már létező tulajdonságnak (lehet ilyen tulajdonság egy csoport összefüggő bővítmény, melyet egyszerre installálhatunk és frissíthetünk). A frissítési menedzser az elérhető bővítményekből új konfigurációt szerkeszt, melyek a legközelebbi Eclipse Platform indításakor kerülnek használatba. Ha a javítás vagy installálás eredménye nem kielégítő, a felhasználó visszatölthet egy korábbi konfigurációt.

Az Eclipse Platform Runtime biztosítja az objektumok kiterjesztésének dinamikus mechanizmusát. Azon osztály, mely implementálja az adaptálható („adaptable") interfészeket, nyújtja a példányok nyitottságot a harmadik fél (third party – itt: megrendelő) igényeinek kiterjesztéséhez. Az objektumillesztő számára (adapter object), mely implementálja az interfészeket és osztályokat, kérdéses lehet egy adaptálható példány. Például a workspace források, adaptálható objektumok; melyekhez a workbench olyan illesztőt ad hozzá, mely a források számára megfelelő ikonokat és szövegcímkéket biztosít. Bármely fél hozzáadhatja igényeit az adaptálható objektumok már meglévő típusaihoz (mind osztályokhoz, mind interfészekhez), mivel beiktat egy megfelelő illesztő tényezőt (adapter factory) a Platformhoz. Különböző felek egymástól függetlenül bővíthetik ki ugyanazt az adaptálható objektumot, mind különböző szándékkal. Mikor igény van egy adott interfész illesztőjére, a Platform azonosítja és segítségül hívja a megfelelő factory-t annak elkészítésére. A mechanizmus csak a Java típusú objektumokat használja (nem növeli az adaptálható objektumok memória lenyomatát). Bármely bővítmény kihasználhatja ezt a mechanizmust, úgy hogy már létező adaptálható objektumokhoz igényeket társít, és új típusú adaptálható objektumokat definiál. Így más bővítmények is képesek ennek használatára és kiterjesztésére."**[1]**

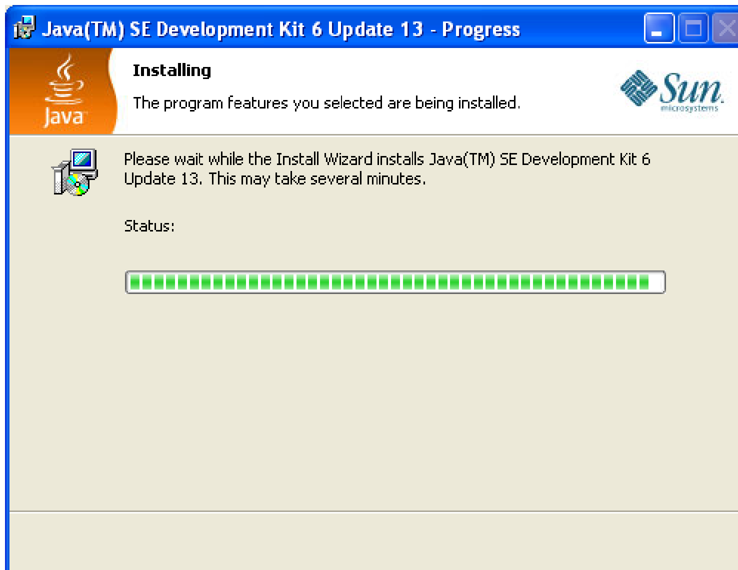
Telepítése

Az Eclipse igényel futtató környezetet (JRE). A Java SE Development Kit (JDK) 6 Update 13 a konkrét verzió, mely tartalmazza a JRE-t is (1.6.0_13 – build 03) a JBoss tools 2.1 bővítménnyel. A szokásos telepítési útvonalakat használtam, minden esetben. A környezeti változók is a szokásos beállításúak. (a JAVA_HOME Változó értéke: <*a jdk vagy jre gyökérkönyvtára*>, a Path változóhoz pedig hozzá kell adni a %JAVA_HOME%\bin utat). A telepítés gyors és egyszerű:

Elsőként a Jogi feltételeket kell elfogadni (free licenc).



Ezt követően az installációs könyvtárat, és a telepítendő tulajdonságokat kell kiválasztani. Ezeket a telepítést követően is lehetőség van megváltoztatni a programok hozzáadás/eltávolítása pontban a vezérlő pulton.



Majd felajánlja a regisztrációt, mellyel hírleveleket és értesítéseket kaphatunk újabb elérhető kiadásokról és speciális ajánlatokról.

Az Eclipse nem igényel telepítést, indításkor tölti be a szükséges modulokat, könyvtárakat. Ezt követően a környezet felajánl különböző oktató, gyorstalpaló lehetőségeket, de akár már kezdhetjük a konkrét fejlesztést is.

A JBoss eszköz telepítését az Eclipse által vihetjük végbe. A szoftver frissítés alatt az elérhető szoftverek listájából, ahol az elérhető URI-t nekünk kell megadni, ha nincs a listában, majd telepíteni.

2.3.2. Az adatbázisszerver

Az adatbázisszerver funkciói a követezők: tárolni az adatokat, és azok biztonságos elérését nyújtani a felhasználók számára. Emellett a meglévő adatokon szelektálást, módosítást, törlést biztosítson. Adatbázisszerver esetében a MySQL és a PostgreSQL került szóba. Egy bonyolult üzleti logikát megvalósító alkalmazás magasabb szintű SQL szelektálásokat is igényelhet. Mivel a felületét jobban ismerem, így a fejlesztés a PostgreSQL adatbázisszervert használtam.

PostgreSQL

„A PostgreSQL rendszer mibenlétét a hivatalos dokumentáció a következőképpen határozza meg: A PostgreSQL a POSTGRES adatbázis management rendszer egy kiegészítése, ami egy következő generációs DBMS kutatási prototípus. Megtartja a POSTGRES adatmodelljét és gazdag adattípus választékát, de a PostQuel lekérdező nyelvet az SQL egy kiterjesztett verziójával helyettesíti. A PostgreSQL szabad és a teljes forráskód hozzáférhető. A PostgreSQL fejlesztését egy csapat végzi, amelynek minden tagja megtalálható az adatbázisszerver fejlesztői levelezési listáján. A jelenlegi koordinátor Marc G. Fournier. Ez a csapat felelős minden fejlesztésért. (A PostgreSQL 1.01 alkotói Andrew Yu és Jolly Chen voltak. Sokan járultak hozzá portolással, teszteléssel, hibakereséssel és fejlesztéssel. Az eredeti Postgres kódot, amiből a PostgreSQL származik, Michael Stonebraker professzor irányítása alatt fejlesztették ki az UC Berkeley egyetem programozói, tanulói és végzett tanulói.) A szoftver eredeti neve Postgres volt. Amikor SQL funkcionalitással egészítették ki 1995-ben, a nevét Postgres95-re változtatták. 1996 végén kapta mai nevét. A PostgreSQL felhasználási feltételei nem igazán szigorúak. Jogunk van használni a szoftvert mindenféle ellenszolgáltatás (pénz, stb.) nélkül, a forráskódot módosíthatjuk, és továbbadhatjuk, de semmilyen, a szoftver használatából következő károsodásért nem vállal garanciát a fejlesztő. A PostgreSQL BSD licenc, egy klasszikus nyílt-forráskód licenc. Nem tartalmaz megszorításokat arra, hogy a forráskódot hogyan használjuk fel. Általában minden UNIX-kompatibilis operációs rendszer képes arra, hogy futtassa a PostgreSQL-t. Azokat a platformokat, amiken tesztelték a kiadást, megtalálhatjuk az installációs utasítások között.

Nem UNIX operációs rendszereken Kliensek esetében a libpq C függvénykönyvtárat (PostgreSQL elérése C programból), a psql-t és más felületeket le lehet úgy fordítani, hogy fussanak MS Windows operációs rendszereken. Ebben az esetben a kliens MS Windows-on

fut és TCP/IP segítségével kommunikál a Unix- on futó szerverrel. A „win32.mak” állomány a kiadás része, ennek segítségével lehet Win32 platformokra lefordítani a libpq-t és a psql-t.

A PostgreSQL ODBC kliensekkel is képes kommunikálni. Az ODBC (Open Database Connenctivity) a különböző adatbázisok közötti kapcsolatok szabványos nyelve.

Szerver esetében az adatbázis szerver Cygwin segítségével fut Windows NT és Win2k rendszereken.

A Cygwin DLL (dynamic link library), segítségével UNIX környezetet emulálhatunk Windows alatt, így akár Linux, Unix alkalmazásokat futtathatunk Win32 környezetben.

A legfrissebb PostgreSQL kiadás a 8.3.7-es verzió, 2009.03.16. 14:30:00-tól érhető el, a 8.4 Béta 1 2009-04-15-i dátumtól tölthető le. A tervek szerint minden évben lesz egy nagyobb fejlesztéseket tartalmazó kiadás, míg a kisebb fejlesztéseket néhány havonta adják ki. Dokumentációval kapcsolatban számos kézikönyv, man oldalak és kis teszt példák találhatóak a kiadásban a doc/ könyvtár alatt.

A PostgreSQL tulajdonságai

Számos nézőpontból lehet vizsgálni a szoftvert: képességek, teljesítmény megbízhatóság, támogatottság és ár.

Képességek: a PostgreSQL rendelkezik a nagy, kereskedelmi adatbázis-kezelő rendszerek képességeivel: tranzakciók, al-lekérdezések, triggerek, nézetek, külső kulcsok, integritás és kifinomult zármechanizmusok. Van néhány képessége, ami a kereskedelmi adatbázisokból hiányzik, mint például a felhasználó által definiált típusok, öröklődés, szabályok és verzió kontroll a zárolási viták redukálásáért.

Teljesítmény: a PostgreSQL teljesítménye hasonlít a kereskedelmi és más nyílt adatbázis szerverekéhez. Lehet bizonyos esetekben lassabb, másokban gyorsabb.

Megbízhatóság: ha egy DBMS nem megbízható, akkor használhatatlan. A PostgreSQL-nél igyekeznek jól tesztelt, stabil kódot kiadni, amiben a lehető legkevesebb hiba van. Minden kiadás előtt eltelik legalább 1 hónap béta teszt, és a kiadási történet is azt mutatja, hogy stabil kódot adnak ki, ami készen áll a produktív felhasználásra.

Támogatás: Levelezési listáik kapcsolatot teremtenek a fejlesztők és felhasználók csoportjával, akik segítenek a problémák megoldásában. A fejlesztő csoport közvetlen elérési

lehetősége, a közösség, a dokumentáció és a forráskód gyakran támogatást biztosít, mint más adatbázisoknál. Van kereskedelmi, alkalmi támogatás azoknak, akiknek szüksége van rá.

Ár: a PostgreSQL szabad bármilyen felhasználásra, akár kereskedelmire is. A termékhez hozzáadhatjuk a saját forráskódjainkat korlátozás nélkül.

A PostgreSQL első osztályú infrastruktúrával rendelkezik, amit 1996-ban indítottak el. Mindez Marc Fourniernek köszönhető, aki létrehozta és karbantartja a rendszert. A minőségi infrastruktúra nagyon fontos egy nyílt forrású szoftver esetében. Megvéd az olyan fennakadásoktól, amelyek komoly késéseket okoznak a fejlesztésekben. Természetesen ez az infrastruktúra nem olcsó. Számos havi és állandó kiadásunk van. „PostgreSQL, Inc"- ként említi, a hozzájárulások kizárólag a PostgreSQL fejlesztésre értendőek, és nem egy meghatározott cégre. Központi bizottság vagy ellenőrző cég nincs a PostgreSQL mögött. Létezik egy mag és CVS commiter csoport, de ez inkább adminisztratív, mint ellenőrző célú. A projectet fejlesztők és felhasználók közössége irányítja, amihez bárki csatlakozhat. Az ebben való tevékenység érdekében csak fel kell iratkozni a levelezőlistákra és részt venni a beszélgetésekben.

Két ODBC meghajtó érhető el: PsqlODBC és az OpenLink ODBC. A PsqlODBC a PostgreSQL kiadás része. Az OpenLink ODBC az ő szabványos ODBC kliens szoftverükkel működik.

A PostgreSQL szervert több programozási nyelvekkel lehet elérni. Én a Java (jdbc) nyelvet alkalmaztam.

A PostgreSQL használja a kernel osztott memória és szemafor API-ját. Figyelni kell, hogy megfelelően konfigurálva legyen a kernel osztott memória támogatása vagy, hogy meg legyen nagyobbítva a maximális osztott memória mérete. A pontos méret szükséglet függ az architektúrától, és hogy hány buffert és folyamatot konfigurálunk a postmasternek. Legalább 1 MB területre szükség van. A PostgreSQL szerver folyamatonként 1 szemafort igényel. A PostgreSQL Adminisztráció kézikönyvben olvashatunk részletesebb információkat az osztott memóriáról és a szemaforokról. Alapértelmezésben a PostgreSQL a helyi kapcsolatokat Unix socketekkel valósítja meg. Más gépek nem lesznek képesek csatlakozni, ha nem engedélyezzük a postmasternek, és nem állítjuk be a kiszolgáló alapú azonosítást. Ezzel válnak lehetővé a TCP/IP kapcsolatok.

Nagyobb teljesítmény elérése érdekében az indexelés feltétlenül gyorsítja a lekérdezéseket. Az EXPLAIN parancs lehetővé teszi, hogy a PostgreSQL miként interpretálja a lekérdezést és melyik indexet használja. Sok INSERT műveletet nagy kötegekben a COPY paranccsal érdemes végrehajtani. Ez sokkal gyorsabb, mint az egyedi INSERT parancsok. Azok a műveletek, amelyek nincsenek tranzakció blokkon belül, azok saját tranzakciót indítanak. Sok műveletet érdemes egy tranzakción belül végrehajtani. Ez csökkenti a tranzakció kezelés többletidejét. Az indexeket javasolt a nagy adatváltozások előtt eltávolítani, majd újra létrehozni. Számos teljesítményjavító lehetőség van. Túl sok klienshez való csatlakozás esetében szükséges lehet növelnünk a postmaster egyidejűleg futtatott szerver folyamat szám korlátját. Az alapértelmezett korlát 32 folyamat. Fontos, hogy a szerver folyamatok maximális számának növelésekor a bufferek számát is növelnünk kell, legalább a folyamatok számának kétszeresére. A PostgreSQL azért korlátozza külön a folyamatok számát, hogy a rendszert ne terhelhesse meg túlságosan.

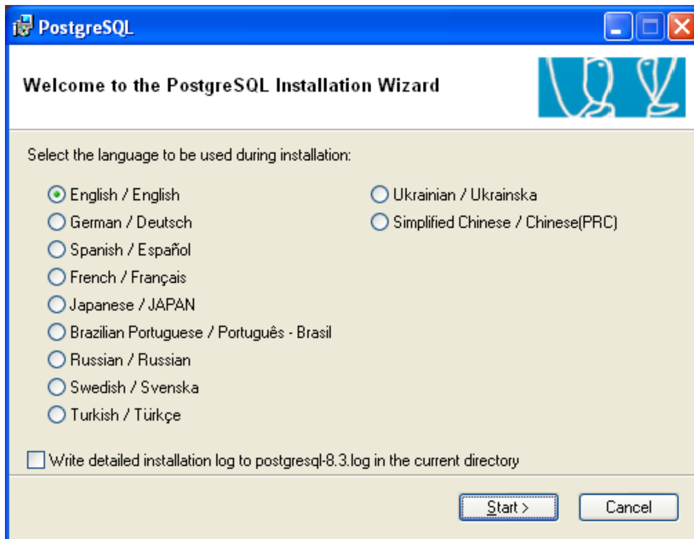
A pgsql_tmp könyvtár a lekérdezés végrehajtó által létrehozott átmeneti állományokat tartalmazza. Például ha egy rendezést kell végrehajtani egy ORDER BY kifejezés miatt, és a művelet több memóriát vesz igénybe, mint amennyit a megengedett, akkor az átmeneti könyvtárban hoz létre egy állományt a fennmaradó adat tárolására. Az átmeneti állományok többnyire törlődnek, de meg is maradhat, ha például váratlan hibával leáll a szerver egy rendezés közben. Indításkor és leállításkor ezeket az állományokat törli a postmaster.

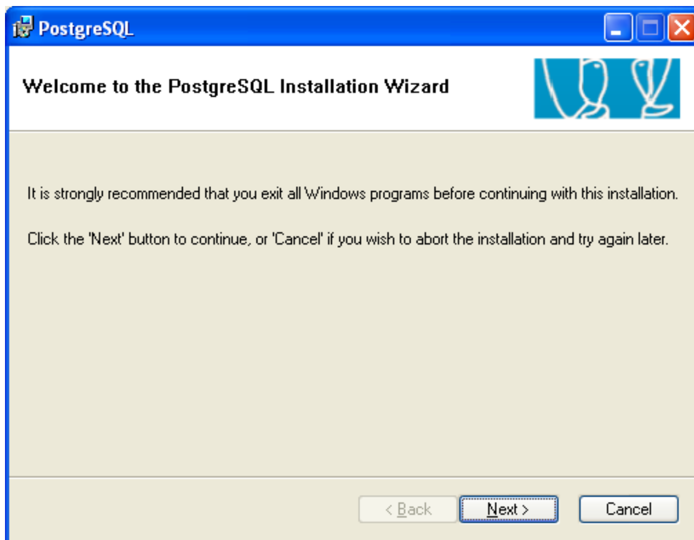
Kiadásváltáskor gyakran szükséges a dumpolás és újratöltés, de nem minden esetben. A PostgreSQL csapat csak apróbb változtatásokat hajt végre a kisebb kiadások között, így ha 7.2 verzióról állunk át 7.2.1 verzióra, akkor nem szükséges kidumpolnunk az adatbázist. A nagy kiadások esetében (például 7.2 verzióról 7.3-ra áttérésnél) változik a belső adatstruktúrák és adatállományok formátuma. Ezek a változások általában nagyon összetettek, visszafelé kompatibilitás gyakran nem lehetséges. A dump az adatot általános formátumban írja ki, majd az új formátumban lehet azt visszatölteni. Azokban a kiadásokban, amelyek között az adat formátum nem változik, a pg_upgrade program használható dumpolás és helyreállítás nélkül.”**[2]**

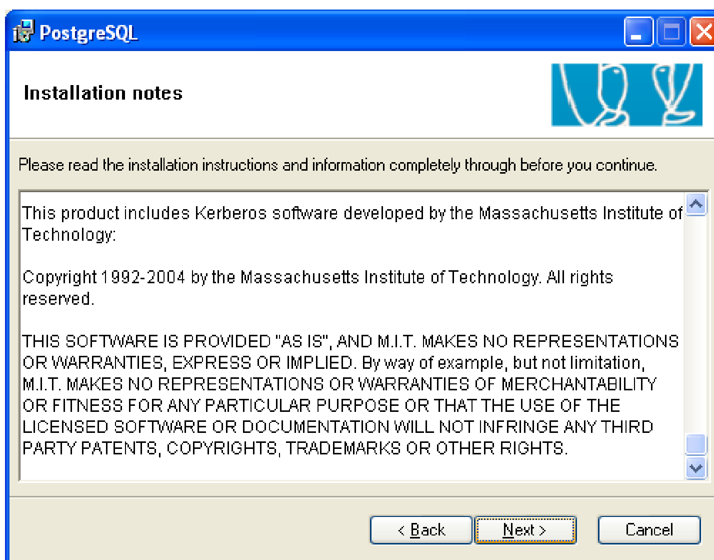
A PostgreSQL telepítése

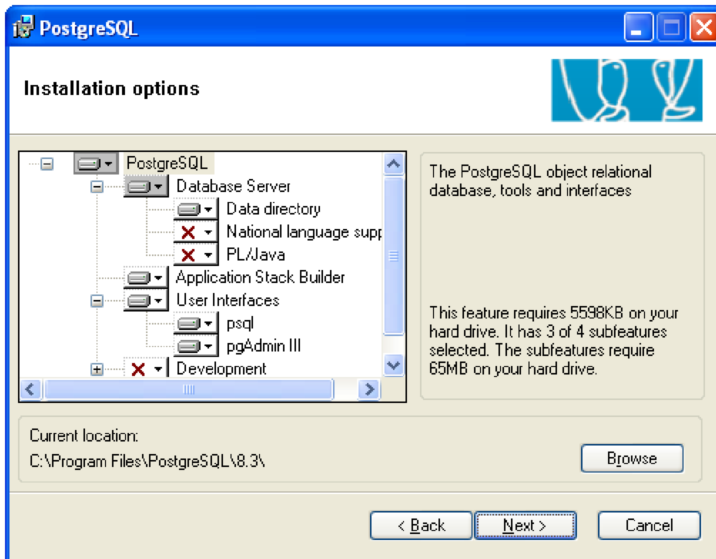
Az adott verzió a PostgresSQL oldaláról tölthető le, postgresql-8.3.7-1.zip néven.

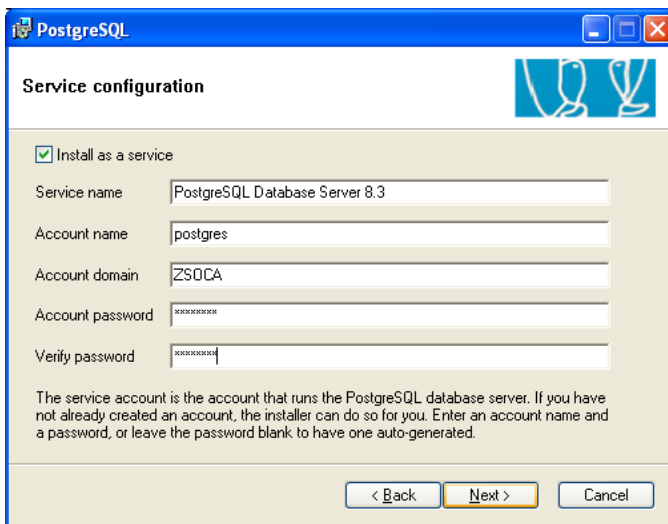
A letöltött tömörített könyvtárból indíthatjuk a telepítést. A menete a szokásos.


PostgreSQL
Welcome to the PostgreSQL Installation Wizard
Select the language to be used during installation:
English / English
German / Deutsch
Spanish / Español
French / Français
Japanese / JAPAN
Brazilian Portuguese / Português - Brasil
Russian / Russian
Swedish / Svenska
Turkish / Türkçe
Ukrainian / Ukrainska
Simplified Chinese / Chinese(PRC)
Write detailed installation log to postgresql-8.3.log in the current directory
Start >
Cancel

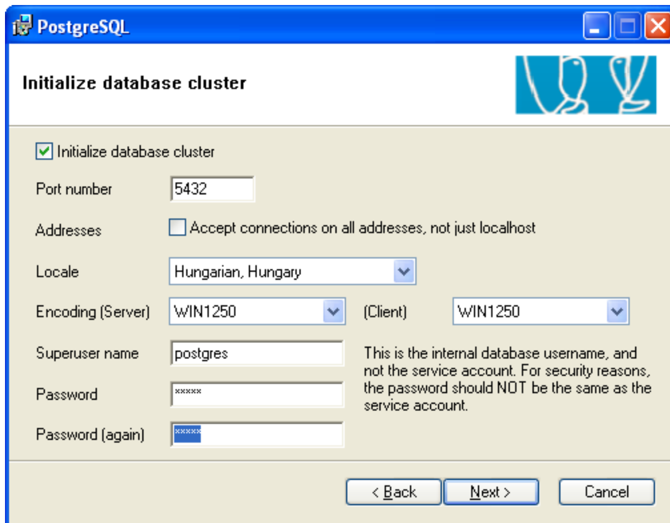

PostgreSQL
Welcome to the PostgreSQL Installation Wizard
It is strongly recommended that you exit all Windows programs before continuing with this installation.
Click the 'Next' button to continue, or 'Cancel' if you wish to abort the installation and try again later.
< Back
Next >
Cancel

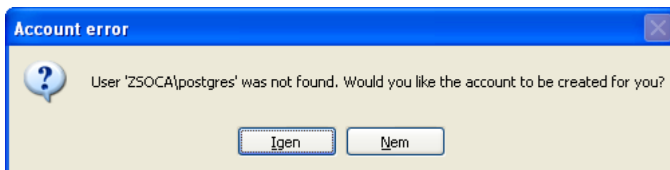

PostgreSQL
Installation notes
Please read the installation instructions and information completely through before you continue.
This product includes Kerberos software developed by the Massachusetts Institute of Technology:
Copyright 1992-2004 by the Massachusetts Institute of Technology. All rights reserved.
THIS SOFTWARE IS PROVIDED "AS IS", AND M.I.T. MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. By way of example, but not limitation, M.I.T. MAKES NO REPRESENTATIONS OR WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE LICENSED SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.
< Back
Next >
Cancel

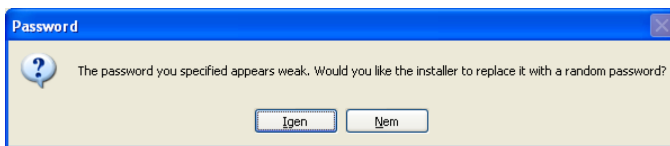




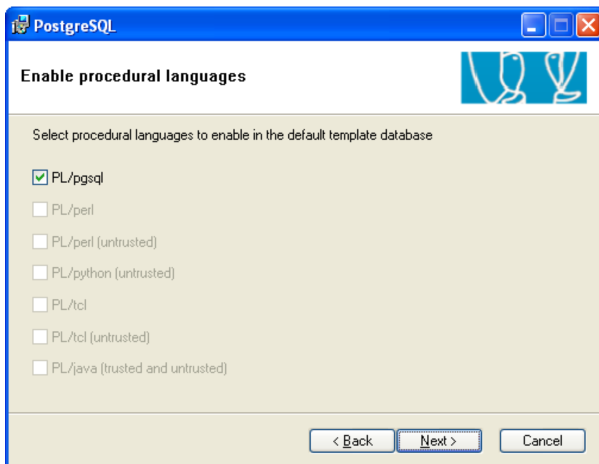
Ezen a képernyőn tudjuk beállítani az adatbázis jellemzőket. A kiszolgáló port, az alapértelmezett felhasználó, a „superuser" neve és jelszava és a nyelvi beállítások.

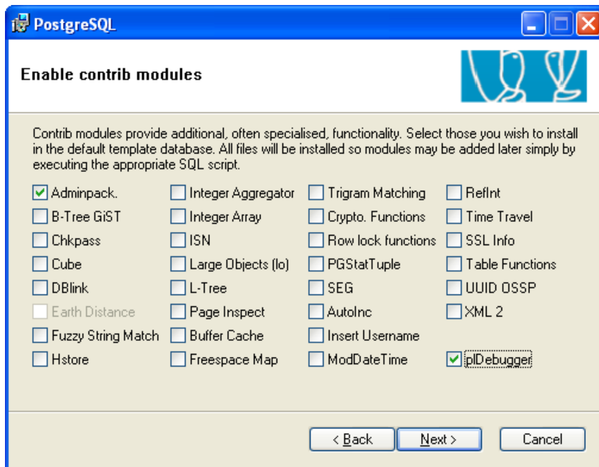




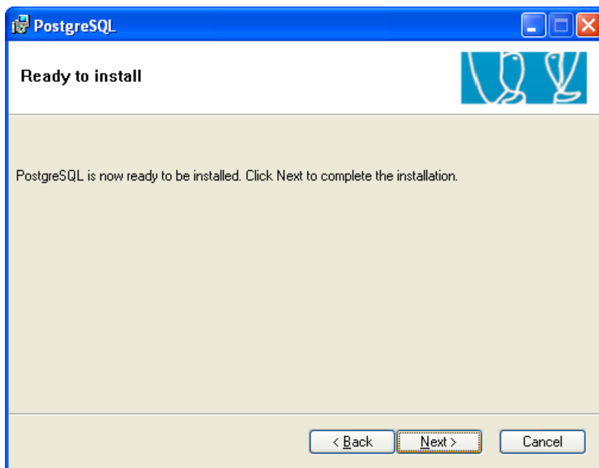


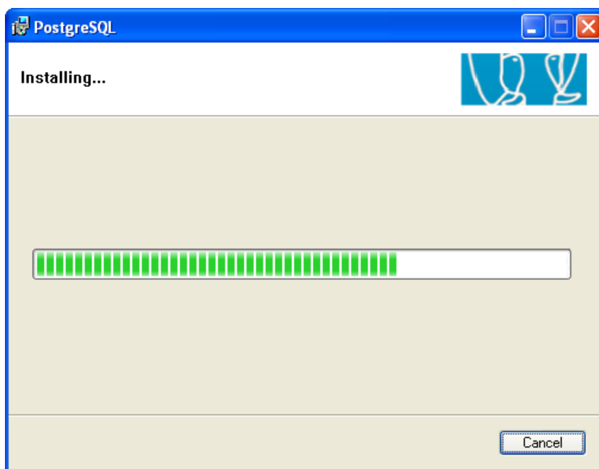
Az alapértelmezett procedurális nyelv a PL/pgsql, emellett számos más is bekapcsolható külön.

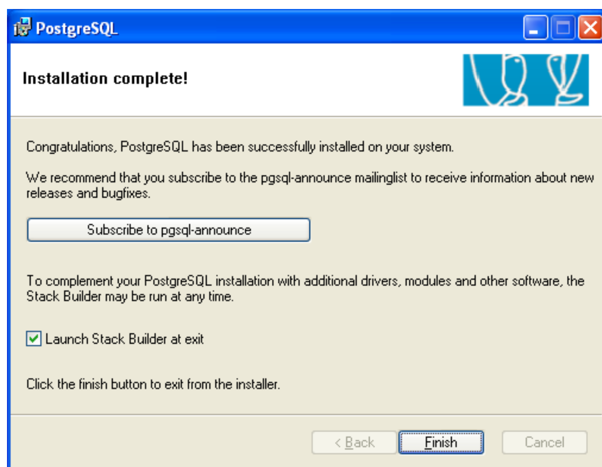




Ezek különböző modulok, az Adminpack és a plDebugger telepítése ajánlott.







2.3.3. Az alkalmazásszerver

Az alkalmazás szerverek feladata, hogy futtatják az alkalmazást és a hozzá tartozó üzleti logikát, s elrejtik a felhasználók elől az adatbázist és annak szerkezetét. Általuk lehetőség van több adatbázisrendszerre kapcsolódni anélkül, hogy a kliens bármit is lásson ebből.

Az alkalmazásszerver a háttérben végzi a dolgát. Összeköti a felhasználót az üzleti logikával, az adatokból olvasható formát képez a felhasználók számára. A felhasználók lehetőségeit szabja meg, jogkörkezelést jeleníti meg. Interfész a felhasználó és az alkalmazás között. A GlassFish teljes mértékben Java fejlesztés, logikus döntés volt az alkalmazása a rendszer összeállításához.

GlassFish

„A 2005-ös JavaOne (a Java felhasználói közösség) konferenciáján a Sun bejelentette a GlassFish projektet, az Alkalmazás Szerverének nyílt forráskódú kezdeményezését, és a Java EE (Enterprise Edition) Hivatkozás Implementációját. Ez volt az első kezdeményezése az összes nyílt forráskódú Java platformnak, de más hatásai is voltak.

A GlassFish projekt gyorsította a Java EE 5 adaptációját, hozzáadva egy új vállalati-minőségű alkalmazás szervert a nyílt forráskódú közösség által elérhető opciókhoz, és olyan változást hozott, mely folyamatban a Sun AppServere fejlettebbé, teszteltebbé, hozzáférhetőbbé és kiforrottabbá alakult, ezzel teremtve egy jobb terméket.

Másfél évvel a kezdeti lendület után, a GlassFish közösség elkészült az első végleges verzióval, és készülőben volt a második.

A GlassFish egy közösség és egy Alkalmazás Szerver. A közösség fő szolgáltatásai a Java EE 5-tel kompatibilis Alkalmazás Szerver, a GlassFish Alkalmazás Szerver és Hivatkozás implementáció – Reference Implementation (RI) a perzisztens réteghez (TopLink Essential).

A GlassFish közösség sok más hasznos összetevőket is szolgáltat, beleértve a Java EE 5-ben lévő JCP technológiák újrahasznosítható moduljait, úgymint a JAXB, JAX-WS, JAXP, StAX, JSP és JSF. GlassFish tartalmaz továbbá néhány népszerű NextWeb projektet: jMaki, Phobos és DynaFaces, a Continious Integration eszköze a Hudson, és tartalmaz hasznos infrastruktúrákat, mint például a NIO alapú keretrendszer szerver, a Grizzly.

A Sun 1995-ben adta ki a Java-t, a következő évben az első JavaOne konferencián prezentálták a Java Servlet API-t. 1999-ben a Java Server Pages-t (JSP) és az Enterprise JavaBeans-t (EJB) a Java Servlets-el egyesítették az első Java Enterprise platformmá: J2EE 1.2 – a verziószám a J2SE-hez való megfelelés alapján lett választva. Kiegészítő kiadások következtek 2001-ben (J2EE 1.3), 2003-ban (J2EE 1.4), majd 2006-ban névváltoztatással a Java EE 5.

Különböző típusú Alkalmazás Szervereket használ a Sun.

A legfrissebb kiadások:

- iPlanet 6.0 – eredetileg mint Netscape, J2EE 1.2 kompatibilis
- SunOne AppServer 7.0 – az első kiadás, mely tartalmazza a J2EE 1.3-t
- Sun Java System AppServer 8.0 – J2EE 1.4 kompatibilis
- Sun Java System AppServer 9.0 – Java EE 5. kompatibilis: GlassFish v1.
- Sun Java System AppServer 9.1 – Java EE 5 kompatibilis: GlassFish v2.

A Java EE környezetet széles körben alkalmazták mind felhasználók, mind vállalatok és több kliens, beleértve a Sun-t, akik saját Alkalmazás Szerverüket implementálták hozzá. A Sun elválasztotta a hivatkozás implementációját és a kereskedelmi alkalmazás szerverét, hogy különböző típusnév alatt kereskedjen velük.

Ez a két ágazat különböző tulajdonságokkal bír: a Hivatkozás Implementáció a fejlesztésre összpontosít, és oktatja a platformokat. A fejlesztése ingyenes volt, de nem bocsáthatták felhasználásra, amíg az államnak jövedelme származott belőle és a felhasználókra fókuszált.

A két ágazat az idő múlásával közeledett egymáshoz, kezdve a 7.0 kiadástól, ahol is elsőként tartalmazta a kereskedelmi termék a J2EE 1.3 SDK funkcionalitását (Sun One), folytatva a 8.x és 9.x kiadások alatt, Sun Java System néven.

Mire a J2EE 1.4 kiadás megérkezett, melyben az RI azonos volt az SJS AS (Sun Java System Application Server) 8.0 Platform Edition-el, addigra ingyenes lett a fejlesztése és felhasználása is egyaránt, és magába foglalt olyan kiadásokat, mint a J2EE 1.4 SDK.

Hivatkozás implementáció

A Java Community Process-en (JCP) keresztül készült specifikációk az Expert Group (EG) eredményei, melyet az EG Lead koordinál.

Az Expert Group szolgáltatásai:

- Platform Dokumentáció
- Technológia Kompatibilitási Csomag - Technology Compatibility Kit (TCK)
- Hivatkozási Implementáció - Reference Implementation (RI)

Az RI implementálja a specifikációkat, és továbbítja a TCK-t. Az RI minősége változtatható a fogalmi bizonyosságtól a termék minőség implementációjáig. A GlassFish egy termék minőségű, Java EE 5 szolgáltatású Alkalmazás Szerver.

Az igazat megvallva, a Java EE 5 RI egy GlassFish kódra épülő sajátos kiadás, és ez ki is van mondva.

Az SJS AS 8.x termékek 3 verzióban még elérhetőek maradtak:

- a Platform Edition-ben (PE), amely megegyezett az RI-vel
- a Standard Edition-ben (SE), mely magába foglalt egyes enterprise (vállalati)-minőségű vonásokat, úgymint a Clustering (fürt) és a Failover (fürt feladatátvevő),
- és az Enterprise Edition (EE), mely a nehezen elérhető piacot célozta meg (99.999%-os elérhetőség).

A 9.x kiadásoktól kezdve egyetlen verzióba egyesítették mindezen tulajdonságokat, mely tisztán a GlassFish közösség által volt fejlesztve nyílt forráskódú licenc alatt. A felhasználó szabadon konfigurálhatja a GlassFish-t, használva a developer, cluster és enterprise profilokat.

Nyílt forráskódú licenc

Az Open Source Initiative (OSI - nyílt forráskódú kezdeményezés) különböző típusú licenceket tartalmaz. A licenceknek gyakran létezik előzménye, melyből örökölnek néhányat a tulajdonságaik közül.

A GlassFish AppServer

A Sun 2005 júniusában a JavaOne konferencián jelentette be a GlassFish projektet. Alig egy évre rá a következő JavaOne konferencián 2006 májusában elérhetővé tették az első kiadást a nagyközönség számára, és véglegessé vált a Java EE 5 platform. Ez az AppServer elérhetővé vált a Sun letöltő oldalán Sun Java System 9.0 PE néven, a közösségi oldalakon GlassFish v1 néven, de ezek az installáció módjától eltekintve megegyeznek. A GlassFish v1 a Java EE 5 specifikációjának kivitelezésére fókuszált, és egyes vállalati jellegű tulajdonság nem szerepelt az első kiadásban. Ebből kifolyólag az AppServer a Sun által a PE elnevezést kapta. A GlassFish v2 tartalmazta mindezen tulajdonságokat, így a PE címkét elhagyta, mivel e tulajdonságok teljesen integrálva lettek: ugyanazon program a developer, enterprise vagy a cluster profilban is installálható, a felhasználó szándékától függően. A GlassFish v2 a Sun oldalán Sun Java System AS 9.1 néven található meg, 2007. szeptemberi kiadással.

A Java platformok

A Java EE egy umbrella specifikáció, amely a megfelelő Java SE specifikációra épül (pl.: Java EE 5 igényel Java SE 5-öt). A GlassFish közösségben is észrevehető ez, több olyan al-projektet tartalmaz, melyek a legtöbb Java EE specifikációnak a hivatalos RI-je. Több előnnyel jár, ha különálló al-projekteket használunk, beleértve más csoportok és projektek általi implementációk egyszerűbb újrahasználhatóságát.

Az EG (a JCP használatával) fejlesztése a Java standard API, számos EG vezető a GlassFish közösség tagja. A GlassFish közösség implementálja e specifikációkat, és megerősítést biztosít az ismertető folyamatokhoz és kérések előterjesztéséhez a jövőbeli kiadások érdekében. A Java EE 6 munkálatai a JSR 316-tal kezdődtek, mely umbrella projekt olyan specifikációkat csoportosít, mint az EJB 3.1, JPA 2.0 stb.. a GlassFish v3 implementációja a Java EE 6.

Nyílt forráskód és GlassFish

Az első széles körben adoptált szerver-oldali Java projekt egyike a Tomcat volt. A Tomcat fejlesztését az Apache kezdte egy csoport által, melybe Sun és a JServ fejlesztők is tartoztak. A Tomcat később a Java Servlets és Java Server Pages specifikációi korai verzióinak RI-je lett (az RI legújabb specifikációja a GlassFish).

A Tomcat elérhető volt nyílt forráskód alatt, és közreműködött a nyílt forráskódú szoftverek népszerűsítésében a vállalati szervezeteken belül.

Forrás injekció

A J2EE 1.4 viszonylag nagy mennyiségű sablon kódot igényel. A következő egy tipikus jellemző az EJB hivatkozásra:

```
Context initial = new InitialContext();
Context myEnv = (Context)initial..
Object objref = myEnv.lookup(„…”);
ConverterHome home = (ConverterHome)
PortableRemoteObject.narrow(...);
Converter currentConverter = home.create();
```

A Java EE 5 esetében elegendő a következő megegyezés és forrás injekció:

```
    @EJB Converter currencyConverter;
```

GlassFish és Java EE 5 adaptáció

A nyílt forráskódú, termék minőségű Java EE 5 implementáció jelenléte felgyorsította a Java EE 5 platform adaptációját, mivel létrehozta a kereslet – kínálat ellátását.

Több mint 3.5 millióan töltötték le a GlassFish-t, e fejlesztők és felhasználók fedték fel a Java EE 5 előnyeit, és növelték a terméktámogatást.

A terjesztők használják a GlassFish-t, beleértve a TmaxSoft, Oracle, JBoss, Geronimo (stb..) eszközöket is. A TmaxSoft egy koreai vezető Java EE licenc, legújabb kiadásuk a JEUS 6

Az Oracle még nem adott ki teljes Java EE 5 AppServert, de fő munkatársai a TopLink Essential-nek a GlassFish közösségen belül, és az Oracle tartalmazza ezt az implementációt a termékeiben.

Java EE 5 áttekintés

A Java EE 5 fő témája a kényelmes fejlesztés, melynek fő megvalósító eszközei a Java SE 5 annotációi, ezzel lehetővé téve a POJO (Plain Old Java Objects – közönséges Java objektumok) programozást. Az annotációk sokféleképpen használhatók, úgymint metódusok és osztályok, RI-k és portolható leírók tulajdonságainak leolvasására.

A fő Java EE 5 specifikációk:

- JAX-WS 2.0 & JAXB 2.0
- EJB 3.0 & Java Persistence API
- JSF 1.2 & JSP 2.1
- StAX

A Java EE 5 és GlassFish kapcsolata szimbiotikus volt: a Java EE 5 erőssége megnövelte a GlassFish értékét, és a GlassFish hozzáférhetősége megerősítette a Java EE 5 szerepét. E kapcsolat a Java EE 5 platform jövőbeli verziójában is remélhetőleg folytatódni fog egy még hatékonyabb visszacsatolási hurokkal (Feedback Loop) a specifikáció saját fejlődése során úgy, ahogy az implementációk korai elérhetősége lehetővé tette azok használatát.

GlassFish kiadások

A GlassFish-nek három különböző verziója létezik, ezek a fejlődésük külön szakaszaiban tartanak: v1, v2, és a v3. A fejlesztési szakaszok:

- Koncepció létrehozása – kulcsfontosságú tulajdonságok összegyűjtése, durva időkeret, egyszerű rutinok
- Aktív fejlesztés – Implementációk használható Mérföldkövekhez és végleges kiadáshoz vezettek
- Karbantartás – végleges kiadások hibáinak javítása, frissített kiadások készítése

A GlassFish v1 a karbantartás fázisában van, a GlassFish v2 mostanában készült el, így belépett a karbantartás fázisába. A GlassFish v3 jelenleg kerül át a koncepció létrehozásának fázisából az aktív fejlesztés fázisába.

GlassFish v1

A GlassFish v1 a karbantartás fázisában van. Végleges kiadása 2006 májusában volt a 2006-os JavaOne előtt. Nincs ütemezve több karbantartott kiadás, mivel a fejlesztés a v2-re összpontosít. A GlassFish kiváló v1 kiadását Java EE 5 RI „névre keresztelték". A GlassFish v1-et a Sun is kiadta Sun Java System AS 9.0 PE név alatt. A GlassFish v1 és a SJS AS 9.0 PE közötti egyetlen különbség a telepítőben rejlik. A Sun különböző kereskedelmi támogatásokat szolgáltat a GlassFish v1 (és az SJS AS 9.0 PE) végső verziójához, míg a közösség a legkiválóbb munkáját.

GlassFish v2

A GlassFish v2 2007. szeptemberi kiadás. Fő – és legfontosabb sajátossága eme kiadásnak a Clustering (csoportosítás, terhelés-egyensúly, adat replikáció). A GlassFish v2 az SJS AS 8.2 SE/EE kiadások minden vállalati minőségű tulajdonságát magában hordozza, és a Java Enterprise System összes termékeivel tesztelik.

A GlassFish v2 támogatja a profilfogalmat, és ugyanaz a végrehajtó konfigurálható developer, enterprise és cluster profilként. Az Enterprise profil konfigurálható HADB (High Avaliability Data Base) használatával. A GlassFish v2-t a Sun alatt Sun Java System AS 9.1 néven is kiadta, mely több csomagot is magába foglalt (mint a Java ES 5.1). A közösség és a kereskedelmi támogatás is szolgáltatva van.

GlassFish v3

A GlassFish v3 a 2007. októberi JavaOne alkalmával mutatták be, nem sok érdeklődést keltve. Az architektúrája az alapértelmezett, a kernelje rendkívül kicsi (100 Kbyte alatti, ezáltal megfelelő PC-k és mobilok számára), és a betöltési ideje 1 másodperc alatt van. A GlassFish v3 aktív fejlesztése nem olyan rég óta él. Bár a tervezés még mindig folyamatban van, a fejlesztés moduláris természete valószínűleg lehetővé tesz olyan magas szintű terméket, mint a Java EE 6.

Telepítési tapasztalatok

A GlassFish v2 egyetlen telepítővel rendelkezik, mely mindhárom profilban (developer, cluster vagy enterprise) installálható. A telepítés maga egy könnyű kétlépcsős folyamat: installáció, konfiguráció. A developer profil minősíthető cluster profillá. A csomag maga

alapjában véve kisebb mint eddig, köszönhetően a Pack200 tömörítési technológiának (mely alkalmazva van a Java SE 6 installációs csomagnál is).

A Sun Java System Alkalmazás Szerverében az installáció egy kissé kifinomultabb és szemléletesebb.

GlassFish a használatban

A nyílt forráskódú projektek közös jellemzője a gyorsított adaptáció, mivel a felhasználók dönthetik el a technológiák használatának esedékességét, mely a saját értékelésükön, a technológia készenlétén és saját kockázati szintjükön alapszik.

A GlassFish-nél ezt a gyorsított adaptációt a GlassFish v1, és a GlassFish v2 esetében is megfigyelhetjük.

A felhasználói tapasztalatok a visszacsatolási hurok utolsó lépcsőfokai, mely szükséges a GlassFish sikerességéhez. Erre az adaptációra egy példa a Peerlflix (peerflix.com), egy média-szolgáltató társaság, mely újraépítette infrastruktúráját a Microsoft-ról a GlassFish v1-re való váltással. Az új oldal megjelenítésére JSF-et Facelet-el és Apache-csal használnak, Kodo JDO-t a MySQL adatbázis perzisztenciájához, mindezek Solaris 10 operációs rendszeren futnak. Az oldallal kapcsolatos kezdeti tapasztalatok nagyon pozitívak. Egy másik kiváló példa a nagyra becsült (Ausztrália harmadik legnagyobb kereskedelmi oldala) wotif.com weboldal, mely szintén a GlassFish-t választotta a JBoss-szal szemben.

A GlassFish fejlesztése

A GlassFish-t nyílt keretek között, közösségi részvétellel és más közösségek közreműködésével fejlesztették.

A közösség és a fejlesztési folyamat nagy hatással volt az eredményre. Egy olyan termék született - összehasonlítva korábbi hagyományos fejlesztési folyamat által készített termékekkel -, mely a gyorsabb és pontosabb visszacsatolási huroknak köszönhetően jobb minőségű lett, és egy gyorsabb fejlesztési ciklus által jobban kielégítette a vásárlók igényeit. A hatások közül néhányra nem számítottak; például külső kommunikációs eszközök használata (web oldalak, blogok, levelező listák stb..) lehetővé tette olyan eszközök használatát, mint a kereső motor, blog olvasó, levelező lista archívum, mely a web felhasználók aktivitását is befolyásolta.

Mindezek eredménye, hogy az információcsere a heterogén GlassFish közösségben jobban történik, mint egy vállalaton belüli homogén csoport estében, még ha ez a csoport földrajzilag, vagy időzóna szempontjából különálló is.”**[3]**

Telepítés

A konkrét verzió az Eclipse update centerén lett letöltve és telepítve, de megtalálható külön is GlassFish Java EE néven a sun oldalán.

A GlassFish telepítése az Eclipse update centerén keresztül könnyedén elvégezhető. Egy új szerver létrehozásakor a listából kiválaszthatjuk azokat, melyek elérhetőek. Ha a GlassFish még nincs telepítve, azt a „További szerver adapterek letöltése” opciónál tudjuk pótolni. Az új szerver telepítése ablakban már láthatjuk a GlassFish alkalmazásszervert, sok egyéb szerver mellett. A telepítése a szokásos módon zajlik. Jogi nyilatkozat elfogadásával kezdődik. Itt sincs semmi akadály, hála a freeware-nek. A telepítés után már elérhetőek a legfrissebb GlassFish szerverek.

2.3.4. A megjelenítés

Az alkalmazás megjelenítését HTML kódok biztosítják. Az adott felhasználók űrlap segítségével tudnak adatokat az adatbázisba bevinni, azokat listázni és szerkeszteni. Ehhez JSF – JavaServer Faces – technológia az ajánlott. A JSF technológiát vékony kliensek esetében érdemes alkalmazni

Terminálszerver alkalmazása

„A költségek csökkentésére és a hatékony adatkezelésre jól alkalmazható a vékony kliens – erős szerver konstrukció. A feladat a következőképpen fogalmazható meg: földrajzilag elkülönülő munkaállomások ugyanazon a központi adatbázison és programcsomaggal dolgozzanak. A kulcsszó: táv-adatkapcsolat.

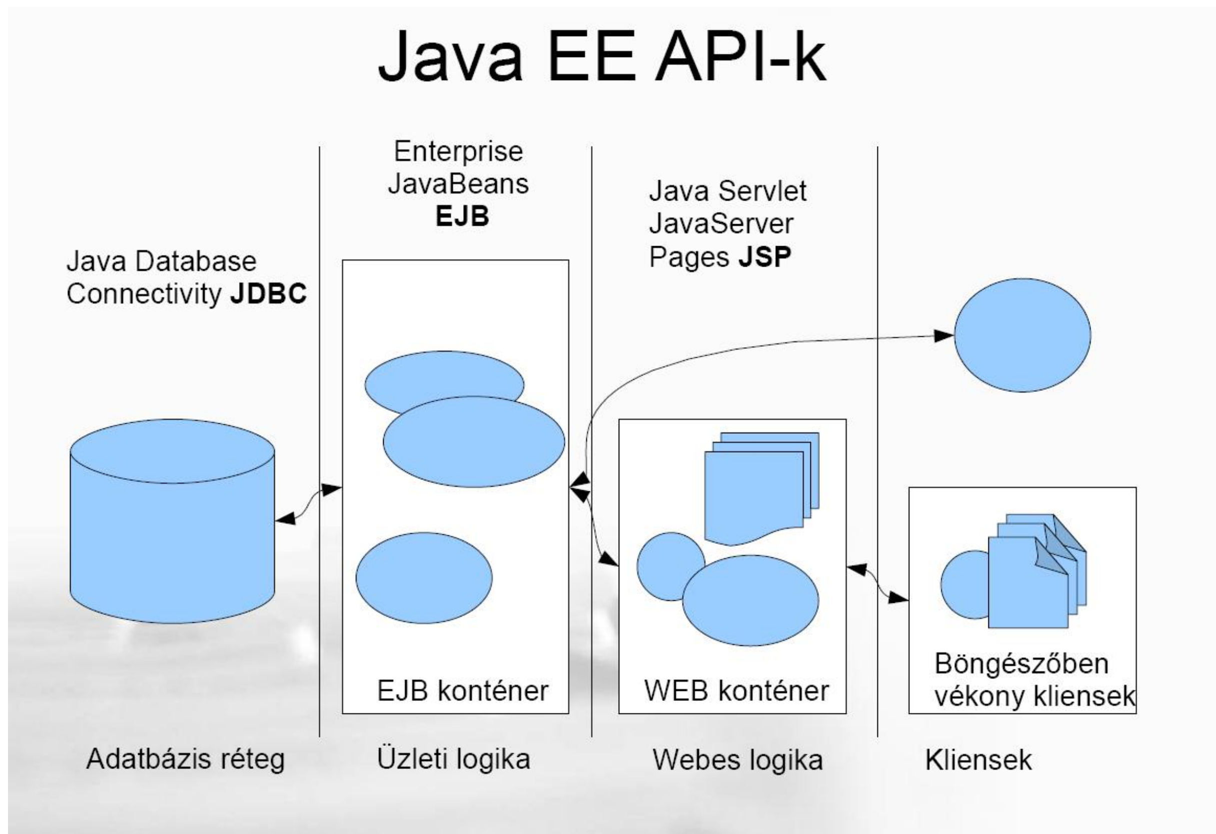
A távoli munkahelyek közötti összeköttetést és munkamegosztást a program szükség esetén kellőképpen támogatja, ami az alábbiak szerint néz ki:

A kliensek a terminálszerveren keresztül bonyolítják le az adatforgalmat a központi szerverrel, ahol az adatbázis található. Ily módon folytonos online kommunikáció valósul meg, ami jelentősen meggyorsítja az adatfeldolgozást, mivel a távoli munkahelyek is ugyanazt az adatbázist használják és ez által nem szükséges a napi adatcsere. A

munkaállomásokon csak a képernyő- és billentyűzet adatok bevitele történik, ezáltal kisebb teljesítményű számítógépek alkalmazására nyílik lehetőség.

A szerverek alkalmazásának másik előnye, hogy különböző operációs rendszereket használó számítógépek összeköttetésére is lehetőség nyílik, így megvalósítható a platformfüggetlenség.”**[4]**

A vékonykliens technológia a következő kapcsolat struktúrában is elképzelhető:



[5]

JavaServer Faces technológia áttekintése

„A JavaServer Faces, vagy JSF egy erőteljes, flexibilis, komponens-alapú technológia kimondottan egyszerű Java webes fejlesztésre. A JSF egy standard iparág a Java világ „meghatározó játékosainak” támogatásával. Mint olyan, kiváló eszközökkel is büszkélkedhet, nagy teljesítő képességű fejlesztési környezettel és a harmadik fél komponenseinek gazdag könyvtáraival.

A JavaServer Faces technológia a következőket fogalja magába:

- Egy rakat API-t a Felhasználói Interfész – User Interface (UI) komponenseinek megjelenítésére, kezeli az eseményeket és a beérkező validálásokat, definiálja az oldalak közti navigációt, többnyelvűséget és hozzáférhetőséget támogat.
- A JavaServer Pages (JSP) felhasználói tag könyvtárakat használ a JSF interfész kifejezéséhez a JSP oldalon belül.

Hogy rugalmas legyen, úgy tervezték a JSF technológiát, hogy befolyásolja a már létező, alapvető UI-t anélkül, hogy a fejlesztőket egy konkrét mark-up nyelvezethez, protokollhoz vagy kliens perifériához kösse. Az UI komponens osztályok, melyeket a JSF technológia tartalmaz, az összetevők funkcionalitását foglalja magába, nem pedig a kliens-specifikus megvalósítást, így lehetővé teszi az JSF UI komponensek fordítását a különböző kliens eszközök számára. Az UI komponens funkcionalitást egyéni fordítókkal - melyek a fordítási jellemzőket definiálják egy adott UI komponenshez - kombinálva a fejlesztők létrehozhatnak saját tag-eket egy konkrét kliens eszközhöz. Kényelmi eszközként a JSF technológia biztosít egy saját fordítót és egy JSP egyéni tag könyvtárat HTML kliensek rendereléséhez, így lehetővé téve a JEE alkalmazások fejlesztőinek, hogy ezen alkalmazások JSF technológiát használjanak. Mivel az elsődleges szempont a könnyű használhatóság volt, ezért a JSF architektúra világosan definiálja a határt az üzleti logika és a megjelenítés között, és egyúttal megkönnyíti a megjelenítési réteg és az implementációs kód összekapcsolását. Ez a kivitelezés lehetővé teszi egy web alkalmazás fejlesztői csapat minden egyes tagjának, hogy a saját részével foglalkozhasson a fejlesztési folyamatban, és ezen túl biztosít egy egyszerű programozási modellt, hogy a különböző részeket egybekapcsolják. Például egy Web fejlesztő, aki nem rendelkezik programozási tapasztalattal, arra használhatja a JSF UI komponens tag-eket, hogy az implementációs kódot hozzákapcsolja a Web oldalhoz anélkül, hogy bármilyen scriptet kelljen írnia. A JSF technológia a szerver oldali felhasználói interfész felépítésének az alapját hozta létre. A fejlesztő csoport hozzájárulásával a JSF API-kat úgy tervezték, hogy befolyásolható legyen olyan eszközök által, amik a Web alkalmazás fejlesztést még könnyebbé tehetik.”**[6]**

2.3.5. A perzisztens réteg

A Java alkalmazások körében, az újabb eszközök és az egyre magasabb szintű igények megjelenése egyre inkább szemléletváltáshoz vezetett. Az új keletű problémát az adatbázis, a

HTML és a Java kódok különbözősége okozza. A perzisztens réteg ezen okból szükséges. A három különböző nyelvnek értenie kell egymást.

A kiszolgáló szerver az a része a fejlesztésnek, melyet a fejlesztőnek kell elkészítenie. Ez a terület teszi egyedivé a rendszert, jó alkalmazkodó képességgel ruházza fel. A kiszolgáló és az adatbázis közötti kapcsolatot a perzisztens réteg látja el. Ehhez a JPA – Java Persistence API – interfész szükséges, hogy a HTML oldal, a Java osztályok és az adatbázis kommunikációja megvalósulhasson.

A fejlesztés során a kiszolgálót kell megírnunk

Négyrétegű modellt (üzleti logika, adatkezelés, adattárolás, megjelenítés) alkalmazva jól elkülöníthetőek a fejlesztési fázisok.

2.4. Fejlesztési Fázisok

2.4.1. Adatbázis táblák létrehozása

Az üzleti logika felépítését már nem sorolom az aktív fejlesztés közé, főként hogy ezt már korábban megtettem. A fejlesztés első lépésként az adatbázis tábláinak struktúráját kell meghatározni. Ezt szemléltetni lehet egyszerű táblázatokkal is.

A felhasználót leíró táblázat struktúrája:

Név	Típus, méret (opcionális)	Megszorítások
Azonosító	Numerikus, 10	Nem lehet üres, egyedinek kell lennie
Vezeték-név	Karakter, 50	Nem lehet üres
Kereszt-név	Karakter, 50	Nem lehet üres
Tel.	Karakter, 50	Nem lehet üres
e-mail	Karakter, 50	Nem lehet üres
Szül. Dátum	Dátum	Nem lehet üres
Státusz	Numerikus, 10	Nem lehet üres
Felettes Azonosító	Numerikus, 10	
Régió	Numerikus, 10	Nem lehet üres

Az adatbázis sokkal többet rejt az adatok tárolásánál. Azzal, hogy definiáljuk az egyes táblákban szereplő oszloptulajdonságokat, az adatbázisban tárolt Felhasználók tulajdonságait definiáljuk. De nem csak a Felhasználók tábláit hozzuk létre, ezen túl a kapcsolatokat is tároljuk. Pl. hogy egy Felhasználóhoz mely státusz tartozik. A feltételeket és kapcsolatokat megszorításokkal érhetjük el:

```
alter table employee add constraint fk_employee_status
foreign key (status) references
employee_status(status_id);
```

Ezzel a megszorítással kikötjük, hogy nem lehet felvenni úgy ügyfelet, hogy nincs hozzá kapcsolódó státusz.

További megszorítások:

A felhasználóneveket és jelszavakat tároló tábla kapcsolata a felhasználótáblával.

```
alter table auth_user add constraint fk_employee_id foreign
key (employee_id) references employee (employee_id);
```

A státuszokat az employee_status táblával reprezentáljuk:

```
create table employee_status (status_id numeric(10) not null
unique, name character varying (100) not null, description
character varying(100));
```

```
insert into employee_status values (1, 'Ügyvezető', 'A
régiókat irányítja');
insert into employee_status values (2, 'Hálózatvezető', 'A
hálózatot irányítja');
insert into employee_status values (3, 'Csoportvezető', 'A
csoportot irányítja');
insert into employee_status values (4, 'Ajánló partner', '');
```

A felhasználó tulajdonságait tartalmazó tábla. felvételének SQL utasítása a következő:

```
create table reference   (REFERENCE_ID NUMERIC(10) NOT NULL
UNIQUE ,
               CUSTOMER_ID NUMERIC(10) NOT NULL,
               LAST_NAME CHARACTER VARYING (15) ,
               FIRST_NAME CHARACTER VARYING (15),
               PHONE  CHARACTER VARYING (11),
               CITY CHARACTER VARYING (25),
               SOCIAL_STATUS CHARACTER VARYING (500)
               );
```

Ezen kívül a régióknak, a felhasználók státuszainak és még sok egyéb funkciónak hozok létre táblát.

A megszorítások egyfajta adatellenőrzést is végeznek. Hiszen a formátumot, és az adat meglétét is ellenőrzi.

Pl.: „REFERENCE_ID NUMERIC(10) NOT NULL UNIQUE” esetén a megszorítások:

- NOT NULL – jelentése: ezt a mezőt kötelező kitölteni.
- UNIQUE – jelentése: ennek a mezőnek egyedinek kell lennie, tehát nem szerepelhet két ugyanolyan azonosítóval az adatbázisban két referencia.

Ezek a megszorítások is meghatározzák az üzleti logika egy részét, elősegítik a program jobb megismerését, átlátását.

Ha olyan tanulmányozza az adatbázis struktúráját, aki nem ismeri a program funkcióit, felépítését vagy működését, így is sok mindenre következtethet abból.

2.4.2. Adatbázis táblák reprezentálása kód szinten

Minden adatbázis tábla számára létre kell hozni egy Java osztályt azok reprezentálásához. Ezek a DTO-k, adatszállító objektumok (Data Transfer Object). A DTO egy egységbe foglalja az üzleti részhez kapcsolódó adatokat. Ez által azok egyben kezelhetők. Egyébként külön-külön minden adatot lekérdezhetővé kellene tennünk, ami egy nagy üzleti megvalósítás és nagy adatforgalom (több felhasználós, sok kérés) esetén nagyban le tudja terhelni az erőforrásokat (minden kérés külön hálózati forgalmat igényel) és rengeteg függvény megírását szükségelteti.

Tehát egy adatbázistábla egy az egyben megfeleltethető egy Java osztálynak. Ezek a logikailag összetartozó adatbázis kódok kezelhetők a hozzájuk tartozó DAO osztály segítségével (nem feltétlenül igényel minden tábla DAO osztályt). Így pl. a felhasználó táblához tartozó DAO osztállyal tudunk felhasználót az adatbázisba felvenni, módosítani, törölni.

Lekérdezések esetén egy SQL (select) fut a háttérben. A connectionManager biztosítja az interfészt az alkalmazásszerver és az adatbázis között.

„DTO osztályra példa (StatusDTO):

```
package hu.company.dto;
public class StatusDTO{
        private Long statusId;
        private String statusName;
        private String description;
        public Long getStatusId(){
             return statusId;
        }
        public void setStatusId(Long statusId){
        this.statusId = statusId;
        }
        public String getStatusName(){
             return statusName;
        }
        public void setStatusName(String statusName){
             this.statusName = statusName;
```

```
        }
        public String getDescription(){
              return description;
        }
        public void setDescription(String description){
              this.description = description;
        }
}
```

2.4.3. Üzleti logikai rész leprogramozása

Az üzleti logika leprogramozást egy osztály végzi, az <EmployeeBO> (BO – Business Object). Itt már csak a feladat üzleti logikai részével kell foglalkozni. Különböző ellenőrzéseket lehet megírni (nem az alapvető adatellenőrzéseket, mert azok már az alkalmazásszerver részén végbe mennek), pl. nem engedünk elmenteni egy felhasználót, amíg minden szükséges adatot ki nem töltött (név, telefonszám stb..).

Továbbá a BO metódusaival definiáljuk, hogy milyen műveletek legyenek végrehajthatók egy felhasználó által. Ezeket a műveleteket az EmployeeDAO osztályon keresztül valósítja meg az adatbázissal. Azt nem tudja – nem szabad tudnia – hogy milyen adatbázis van alatta, és hogy hogyan kell vele kommunikálnia, az a DAO feladata, a BO csak az üzleti logikai feladatokat látja el.

Ezek a feladatok a következők:

```
private Logger logger =
GlobalUtils.getLogger(EmployeeBO.class);
```

A naplózáshoz szükséges logger-t hozza létre.

```
private Long employeeId = null;
```

Egy kezdeti értéket ad meg az azonosítónak, plusz biztonsági okokból is hasznos, mivel így valamely hibás működés vagy hibás ID esetén az adatbázisszerver fogja kidobni, hiszen NULL értéke nem lehet ennek a mezőnek.

```
EmployeeDAO empDAO = new EmployeeDAOImpl();
```

Egy új EmployeeDAO példányt hoz létre, melyet az EmployeeDAOImpl fog implementálni.

```
public UserObject checkUser(String userName, String
password) throws Exception;
```

A Felhasználó adatait ellenőrzi le. Ennek minden egyes action híváskor le kell futnia. Azért szükséges, mert ellenőrizni kell, hogy van-e érvényes felhasználó a rendszerben. Ezzel védjük ki, hogy egy megjegyzett URL esetén azt máskor beírva ne tudjon senki eljutni egy oldalra sem, csak aki be is van jelentkezve.

```
private Long getMaxEmployeeId() throws Exception;
```

Ez a legnagyobb azonosítót kéri el az adatbázisból, hiszen nem bízzuk az adatbázisra az azonosítók kiosztását, kód szinten végezzük mi. Jelentősége hogy könnyebb később belenyúlni a kódba.

```
public EmployeeDTO insertEmployee(EmployeeDTO employeeDTO)
throws Exception;
```

Új felhasználót tudunk felvenni (beszúrni).

```
public EmployeeDTO getEmployeeRecord(EmployeeDTO
employeeDTO) throws Exception;
```

A felhasználó adatait is el tudjuk kérni.

```
public List<StatusDTO> getEmployeeStatusList(Long from,
Long to) throws Exception;
```

A státuszát is le tudjuk kérni a felhasználónak. Az megmondja, hogy egy felhasználó milyen státuszú (Ügyvezető, Hálózatvezető, Csoportvezető, Ajánló partner). Egy új felhasználó felvételekor a lehetséges státuszok listáját hozza le a metódus.

A DAO, ahogy már korábban rátértem, az adatbázissal való kommunikációért felelős. Az EmployeeDAOImpl osztály a BaseDAO kiterjesztése, és az EmployeeDAO interfész metódusait implementálja, ez által valósítja meg az EmployeeBO osztályban definiált működést.

A BaseDAO terjeszti ki az összes olyan osztályt, ami az adatbázishoz nyúl (DAO osztályok). Így sok mindent, amiket minden egyes ilyen osztály esetében meg kellene írni, azt elég ebben, a többiben pedig csak meghívni.

Fontos, hogy a BaseDAO osztályban van lekódolva a beginStatement() metódus. A statement (PrepareStatement) használata szintén biztonsági okokból ajánlott. Az SQLInjection támadásokat védi ki, amikor is a felhasználónév helyére SQL parancs kerül. Ez – a PrepareStatement használata nélkül – lefutna az adatbázison, komoly problémákat okozhatva abban. Ezzel szemben minden beírt adat le lesz fordítva SQL nyelvre, azaz az ilyen jellegű

támadások nem jutnak el az adatbázisig (a beírt SQL parancs egy SQL parancsba ágyazva nem lesz értelmezhető).

A másik jelentősége, hogy az adatbázis számára egy un. preparált SQL parancsot küld el, amit az adatbázisrendszer lefordít a saját nyelvére, és készít belőle egy végrehajtási tervet (plan). Ha egy ezzel megegyező SQL érkezik az adatbázisrendszerbe (más paraméterekkel), akkor ezt a plan-t annak nem kell újra elkészítenie. Ez gyorsaság szempontjából nem elhanyagolható, nagyobb select-ek esetén van jelentősége.

A beginStatement() ebbe a folyamatba lép bele, a resultSet() pedig az SQL visszatérési értékét ellenőrzi. Ha pl. egy rekordot keresünk az adatbázisban (egy adott ügyfelet), de nem található, akkor a resultSet.next() false értékkel tér vissza. A resultSet.next() azt mondja meg, hogy van-e még érték a result-ban.

A RecordNotFoundException osztályt is érdemes itt megemlítni. Az adatbázisszerver az adott keresési feltételeknek eleget tevő rekordokat gyűjti össze és adja vissza egy resultSet–ben. Ha nem talált olyan rekordot, ami a feltételeknek eleget tenne, akkor egy üres resultSet- et ad vissza, így az if(resultSet.next()) vizsgálatnál ezt az egyezést vizsgáljuk.

Pl. ha azt vizsgáljuk, hogy egy adott felhasználó névvel illetve jelszóval létezik-e érvényes account. Ha nem létezik, akkor a resultSet.next() false értékkel tér vissza.

```
if (resultSet.next() == true){}
else {throw new RecordNotFoundException}
```

2.4.4. Felhasználói felület (GUI) és mögöttes kód kialakítása

A felhasználói felületet jelenti a JSF. Emberi formába teszi a felhasználók elé a kódot, interfészt biztosít a felhasználó és az adatok között. Ide tartozik a böngészőkben való megjelenítés, az űrlapok szerkesztése. A JSF a böngésző számára érthető HTML kódot generál.

A JSP kódot az alkalmazásszerver fordítja le Java kódra, amiből a felhasználónak a JSF HTML kódot gyárt.

Az alkalmazásszerver működése: JSP → JSF (Java kódra való fordítás) → HTML kódra fordítás.

A mögöttes kód kialakítása

Egy HTML oldalt (illetve a benne lévő Form-ot) szintén egy java osztállyal reprezentálunk. Amikor egy felhasználó a felületen kitölti a mezőket, azt az alkalmazásszerver Java osztállyá – Form osztállyá – fordítja át. Ezek lesznek a Form osztályok.

A BaseActionForm osztály egy absztrakt osztály, a ValidatorForm-ot terjeszti ki. Ezt az osztályt terjeszti ki az adott oldalhoz tartozó Form osztály, és a

```
public ActionErrors validate(ActionMapping mapping,
HttpServletRequest request)
```

metódust terheljük túl.

Az EditEmployeeForm osztály a felhasználók szerkesztési oldalának megfeleltetett osztály.

```
public void loadFromDTO(EmployeeDTO employeeDTO)
```

Ez a metódus egy Form-ot tölt fel egy DTO alapján.

```
public EmployeeDTO getEmployeeDTO(Logger logger)
```

Ez a metódus egy EmployeeDTO-t ad vissza az adott Form alapján.

Az EmployeeInfoForm a felhasználókról információt lekérő oldalnak megfelelő osztály

A LoginForm a bejelentkezési oldal, a MenuForm a menüből elérhető HTML oldalak (űrlapok) megfelelői, stb..

Validálások

Itt szükséges megemlíteni, hogy a

```
public void validate(ActionMapping mapping,
HttpServletRequest request, ActionErrors errors, Logger
logger){}
```

metódus segítségével lehet a bekért adatokat validálni. Itt nyílván nem üzleti logikai érvényesítésről van szó, hiszen ezek legtöbbje adatbázis szinten ellenőrizve van, mint pl. van-e már ilyen telefonszám, e-mail beregisztrálva stb.. Ezek a megszorítások által véghezvitt ellenőrzések. Itt olyan validálások szerepelnek, mint pl. ki van-e töltve minden kötelező adat, megfelelő-e a formátum (e-mail cím). Bármely validáció során fellépő hiba esetén a program automatikusan visszavisz a HTML oldalra.

Action osztályok

Ha a validálások során nem lépett fel semmilyen hiba, akkor az Action-höz jutunk tovább. Itt a program megkapja a már előállított Form-ot. Itt kell meghívni a háttér részt, ahol az üzleti folyamatok mennek végbe. Ekkor dől el, hogy melyik oldalra megyünk tovább. Hiba esetén az adott oldalra tér vissza a navigáció, egyébként a működésnek megfelelő oldalra. Az adott Form helyes lefutása után a hozzá tartozó Action hívódik meg.

Minden Action a BaseAction absztrakt osztályból származik, minden más Action osztály ezt terjeszti ki. Itt a

```
public abstract ActionForward execute(ActionMapping
mapping, HttpServletRequest request, ActionForm form,
ActionErrors errors, Logger logger) throws Exception;
```

metódust terheljük túl.

Az EditEmployeeAction, EmployeeInfoAction, LoginAction, MenuAction osztályok a megfelelő oldalak Action osztályai stb..

Pl. a MenuAction esetén minden menügomb egy másik oldalra visz. Így szükséges hozzá a megfelelő Action, hogy tudja az alkalmazásszerver, hogy másik oldalra megyünk, és végezze el a szükséges felhasználó ellenőrzést, adatokat kérjen el a háttérből stb.. Pl. egy Felhasználó szerkesztési oldal felvitele a következőkép alakul:

```
if ( MenuForm.ACTION_EDITEMPLOYEEDATA.
     equalsIgnoreCase(aform.getAction())){
     return mapping.findForward(FORWARD_EDITEMPLOYEE);
}
```

JSPPreparation osztály

Az oldalakon lehetnek statikus (mindig ugyan úgy megjelenő) összetevők. Mint pl. a ComboBox-ok. Az ezek kitöltéséhez szükséges a JSPPreparation osztály. Ennek az osztálynak a feladata, hogy ellenőrizze, milyen osztály következik, és hogy betöltse az ahhoz szükséges adatokat.

Hogy minden osztály előtt lefusson a JSPPreparation (ha volt hiba, ha nem), szükséges a BaseAction és a BaseActionForm osztály. Amikor az alkalmazásszerver létrehoz egy Form-ot, vagy meghív egy Actiont, akkor elindul az ős osztály, megtörténik a user ellenőrzés, a

JSPPreparation hívás, majd amikor elér a BaseActionForm validate, vagy a BaseAction execute metódusaihoz, akkor már az általunk írt kód fog lefutni.

Felhasználó validálás

Érvényes bejelentkezett felhasználó ellenőrzésére minden egyes művelet előtt szükség van. Ezt a munkamenetből (session) az alkalmazásszerver egy adott kulcsú objektum (UserObject) kiolvasásával teszi meg.

Ha ez NULL, vagy hibás, akkor a rendszer automatikusan megszakítja a folyamatot, és átirányítja a felhasznált a kezdő oldalra.

A session addig él, amíg a felhasználó használja az adott oldalt, amíg a böngészőablak nyitva van. Általában vannak – beépített – időkorlátok, mely letelése után az alkalmazásszerver automatikusan eldobja ezt a session-t, ha nem történik kommunikáció.

Egy felhasználó belépésekor mindig a session-be tesszük a UserObject-et. Ez a scope annyi ideig él, amíg az érvényes felhasználó él.

Minden egyes scope-hoz (application, request, session) létrejön egy MAP, a session és a request esetében minden egyes felhasználóhoz külön. A request-nél egy metódus futásakor jön létre új MAP, és minden egyes kérés után elvész, a munkamenetnél addig él, amíg a felhasználó ki nem lép, vagy le nem telik az időkorlát, az application-nél pedig addig, amíg az alkalmazásszerver fut.

Felhasználó beléptetés (Authentikáció)

A felhasználó belépésekor a programnak ellenőriznie kell, hogy érvényes felhasználónevet illetve jelszót adott-e meg az illető. Érvényes user/pass esetén a rendszer létrehoz egy UserObject osztályt, és beleteszi az adott felhasználóhoz keletkező session-be.

Framework csomag

Ebben a csomagban olyan általános és gyakran használt eszközök vannak, amelyek az alkalmazás alapjait építik fel.

A ConnectionManager osztály a kapcsolatok kiépítéséért felelős. Egy kapcsolódást a ConnectionPool-tól kérünk. Ha már a kapcsolatra nincs szükség, akkor azt nem iktatjuk ki, hanem eltesszük a ConnectionPool-ba, és ha szükség van ismét egy connection-re, akkor nem

kell újra kiépíteni az adatbázissal azt, elég ehhez nyúlni. Ez által gyorsabb lesz az adatbázis kommunikáció.

A GlobalUtils a GUI részhez nyújt segítséget. Pl. a dátum érvényesítés miden egyes esetbeli újbóli leprogramozását ki tudjuk kerülni úgy, hogy a GU-ban reprezentáljuk, és később, amikor szükséges, innen hívjuk meg. Ebben van a logger elérése, dátum validálás, user ellenőrzés.

A JSPPreparation osztályról már volt szó. Ez egy előkészítő osztály a HTML oldalak számára. Az oldal megjelenítéséhez elengedhetetlen, szükséges adatokat itt gyűjtjük össze, a JSPPreparation a BO-val, a BO a DAOImpl-nel, az pedig végül az adatbázissal „kommunikál". Vagyis az adatbázisból kér el adatokat köztes interfészen keresztül.

A Log4JServlet a naplózáshoz szükséges osztály. Az alkalmazásszervernek be van állítva, hogy elindulásakor indítsa ezt a servlet-et, ami pedig beállítja számunkra a logger-t a naplózáshoz.

A UserObject osztály kerül be egy adott felhasználó számára létrejövő session-be. Az EmployeeBO esetében leírt checkUser-ben ellenőrizzük, hogy létezik-e az objektum. Ha igen, akkor nem NULL a visszatérési értéke.

```
UserObject userObject =
(UserObject)request.getSession().getAttribute(USESSIONID);
    if (userObject == null || userObject.getUserId() == null)
        return false;
    return true;
```

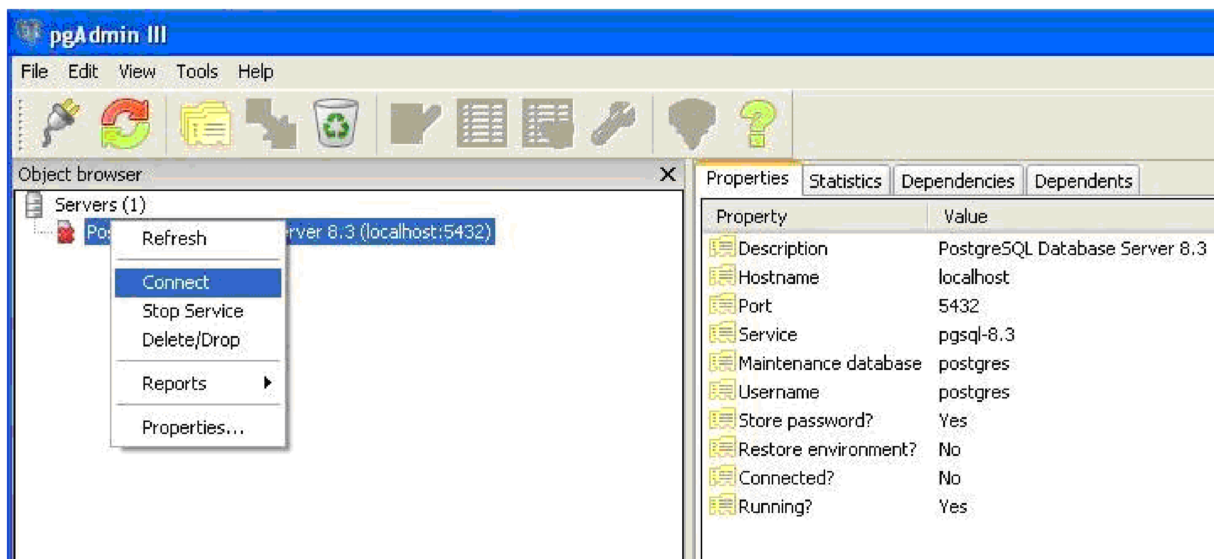
Ott jelenik meg a session, amitől elkérjük és ellenőrizzük.

Webapp

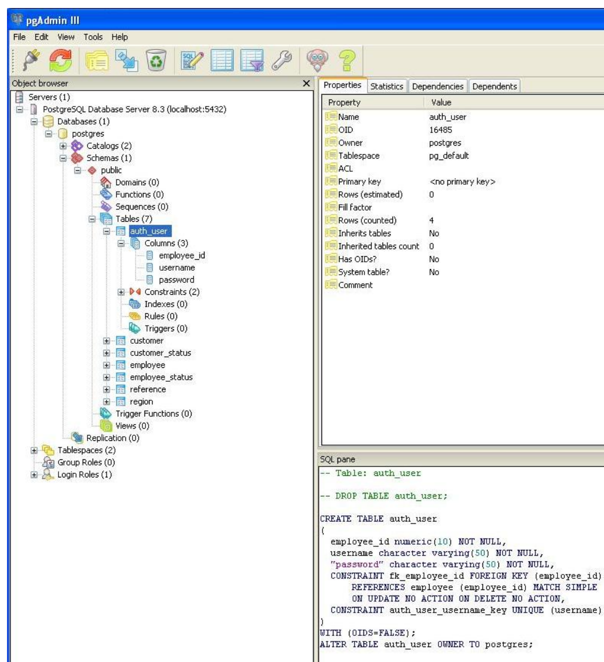
Ez is felhasználó ellenőrzést végez. Ha nem érvényes felhasználó hívta meg az adott oldalt, akkor kilépteti a rendszert, és átviszi a bejelentkező oldalra. Segítségével nem szükséges minden HTML elemet megírni, tudunk Java kódokkal TAG-eket létrehozni, ami sokkal átláthatóbbá teszi a programot.

2.5. A rendszer feláll

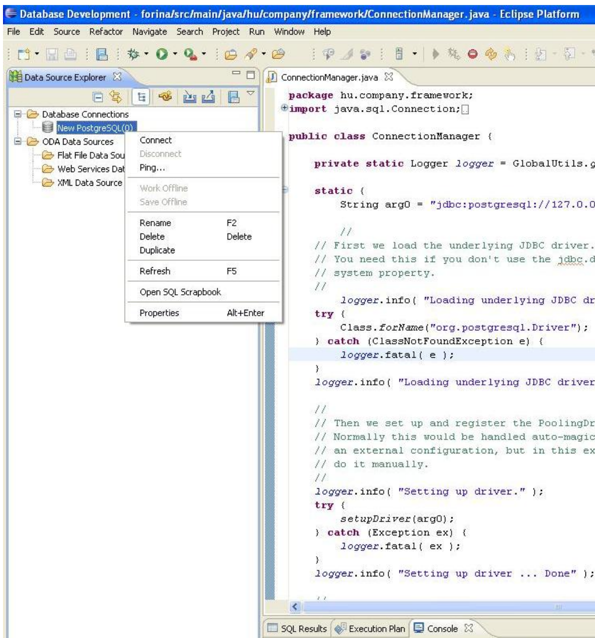
Első lépésként az Eclipse-ről gondoskodtam, ez emésztette fel a legtöbb időt. Felállása alatt futtattam a PostgreSQL-t majd csatlakoztattam is az adatbázisszervert a localhost-ra.

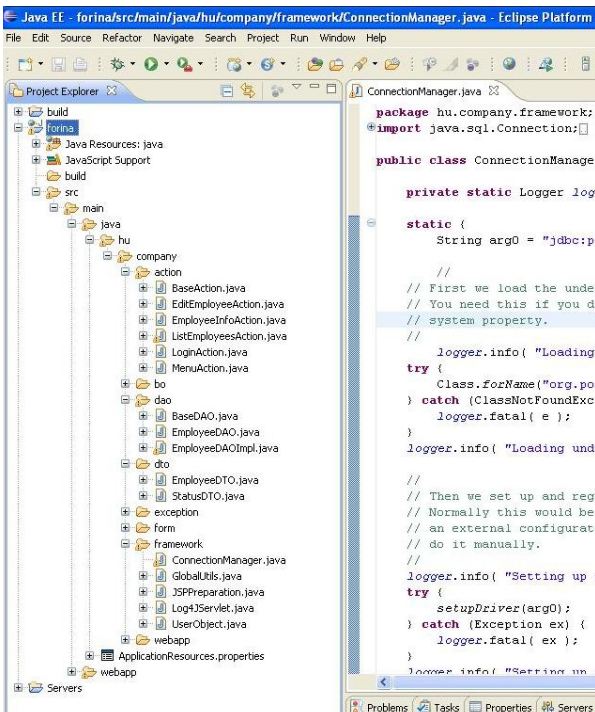


A meglévő tábláinkat láthatjuk is, ahol tetszőleges számú SQL kérést hajthatunk végre.

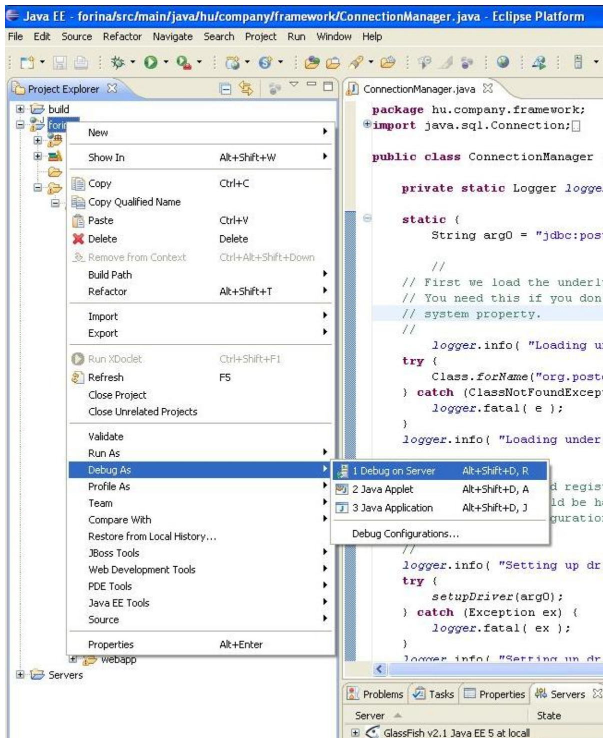


Ezt követően az Eclipse már általában használhatóvá vált. Elsőként itt is az adatbázissal kezdtem, csatlakoztam a szerverhez. Ez után már az Eclipse-ből is lehet látni az adatbázistáblákat, azok felépítését.

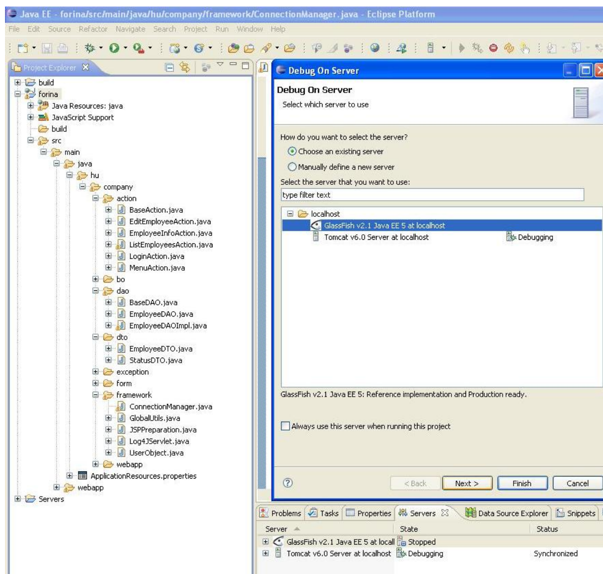




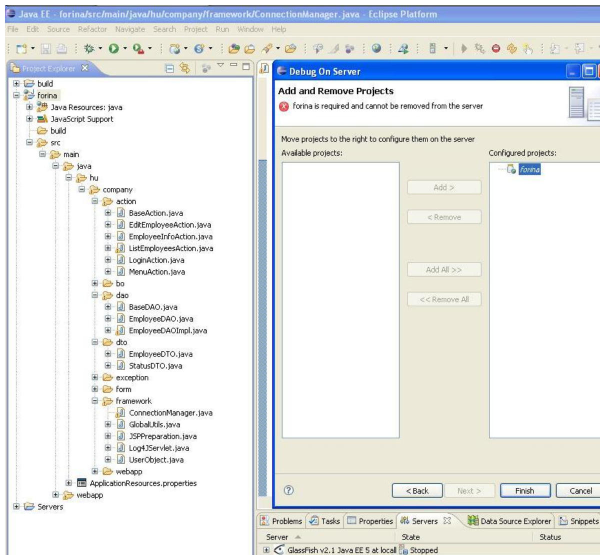
Legvégül az alkalmazás futtatása a feladat. A project futtatásával egyidejűleg futtatom az alkalmazásszervert is.



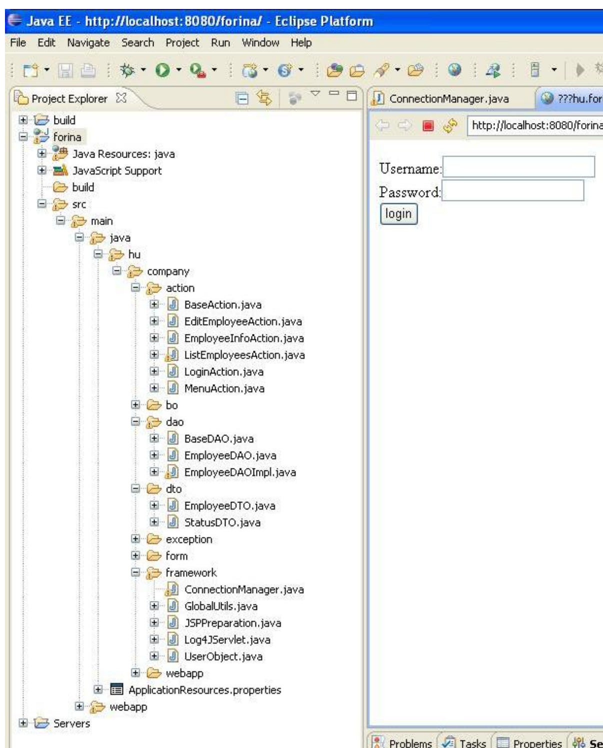
Ha több alkalmazásszerver is létre van hozva, akkor ki kell választanunk a listából a megfelelőt.



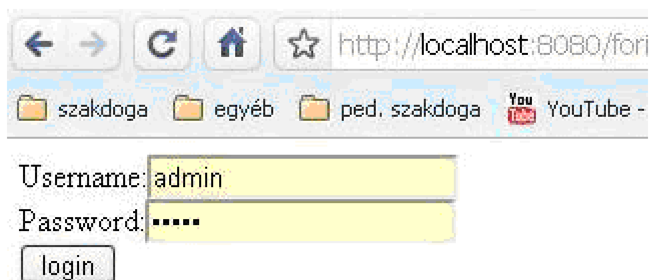
Ha még nem adtuk hozzá a projektünkhöz, még megtehetjük.



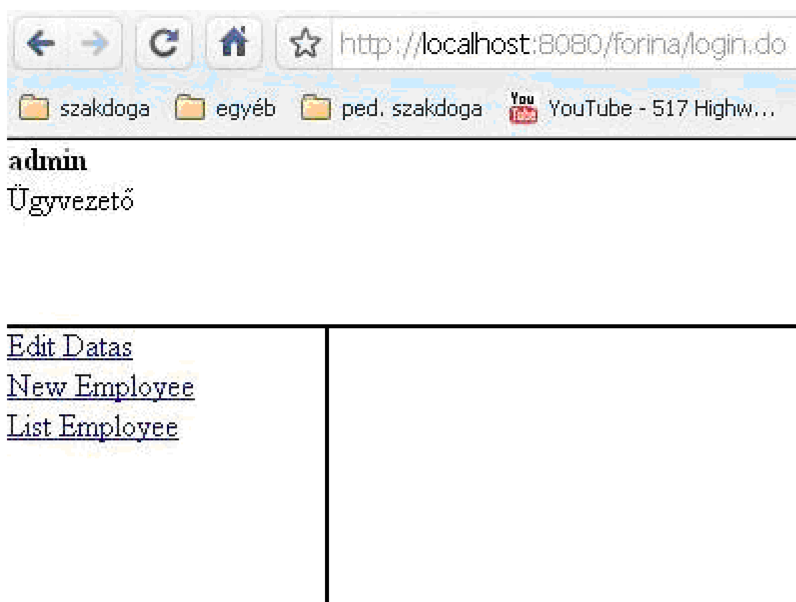
Ezt követően már fut is az alkalmazás. Én böngészőből szeretem elérni, így a további lépéseket onnan folytatom.



Egy felhasználó felvétele egy már meglévő felhasználó segítségével, vagy a rendszergazda közreműködésével lehetséges. Az admin profilt nyílván kód szinten kell létrehozni.



A bejelentkezés után a „főoldalra” jutunk, ahol elvégezhetjük a lehetséges műveleteket.



Ezek pedig

- A személyes adatok módosítása
- Új ügyfél felvétele
- Meglévő ügyfelek listázása

admin
Ügyvezető

Edit Datas
New Employee
List Employee

First name	Nagy
Last name	Sandor
Phone	20/123-2313
Email	sandornagy@company.co
Birth date	2002-01-01
City	Debrecen
Status	Ügyvezető
Boss	Nagy Sandor
Region	Debreceni Regio

Save

A korábban definiált és létrehozott felhasználó tábla megfelelő attribútumait tudjuk értékes adatokkal feltölteni.

admin
Ügyvezető

Edit Datas
New Employee
List Employee

First name	
Last name	
Phone	
Email	
Birth date	
City	
Status	Hálózatvezető
Boss	
Region	

Ügyvezető
Hálózatvezető
Csoportveető
Ajánló partner

Save

Új felhasználó felvételénél listából adhatjuk meg, milyen státuszú ügyfelet szeretnénk felvenni. Lehetőség van ezt az opciót korlátozni, a hierarchiának megfelelően kisebb szintet képviselő felhasználó felvételére.

admin
Ügyvezető

Edit Datas
New Employee
List Employee

First name	Attila
Last name	Fehér
Phone	70/234-1236
Email	attilafeher@company.com
Birth date	1974-09-26
City	Debrecen
Status	Hálózatvezető
Boss	Nagy Sandor
Region	Debreceni Regio

Save

A felettes és a régió is tárolva lesz a rendszerben, ezzel hozzuk létre a hierarchia alapjait. Listázáskor láthatjuk a felvett – alánk tartozó – felhasználókat.

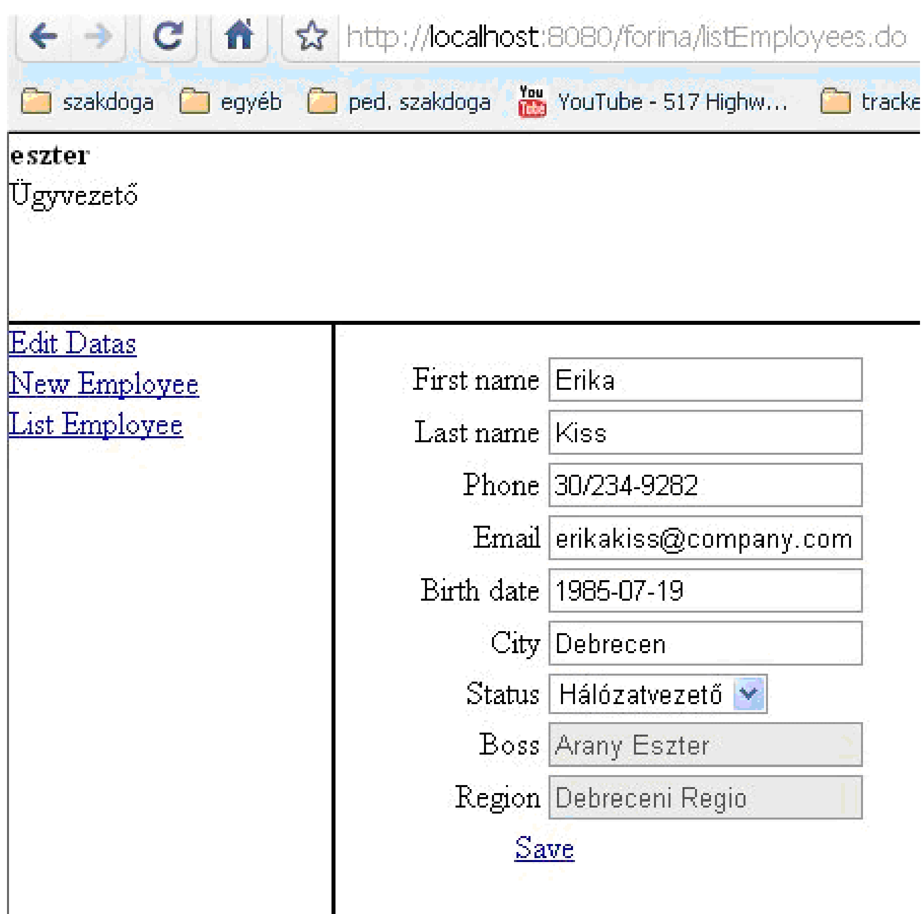
admin
Ügyvezető

Edit Datas
New Employee
List Employee

First name	Last name	Phone	Email	Status name		
halozat first name	halozat lastname	2324	fdsfs@efe.hu	Ügyvezető	List employee's	Edit employee
Arany	Eszter	30/234-2341	arany@arany.hu	Ügyvezető	List employee's	Edit employee
Szabo	Istvan	30/744-3837	istvanszabo@company.com	Hálózatvezető	List employee's	Edit employee
Nagy	Sandor	20/123-2313	sandornagy@company.com	Ügyvezető	List employee's	Edit employee
Attila	Feher	70/234-9189	attilafeher@company.com	Ügyvezető	List employee's	Edit employee

Innen lehetőségünk van minden egyes alkalmazottunk saját alkalmazottait listázni, azok személyes adatait módosítani.

Eszter nevű ügyvezetővel való bejelentkezés során tudunk számára is alkalmazottat létrehozni, annak adatait később módosítani.



A felépített rendszer funkcióit tetszőleges részletességig bővíthetjük, személyre szabott alkalmazássá alakíthatjuk. Az alapkövek és a szerkezet felépítése sikerült.

3. Összegzés

A fejlesztés szakaszai bár jól elkülönülnek egymástól, mégis párhuzamosan kell, hogy folyjanak, legalább a vázlat felett vagy a fejlesztő fejében. Gyakran hallom informatikus barátaimtól, sőt a legtöbb egyetemi tanártól is, hogy a programkészítés papírral és ceruzával kezdődik. Ez persze nem kódvázlathoz kell, hanem a program szerkezeti felépítéséhez, struktúrájához. Ezt egy alkalmazásfejlesztés során még inkább meg kell fogadni, hisz a részletekben rejlik a lényeg, és épp ebben lehet nagyon hamar elveszni.

Az alkalmazásfejlesztés – bár nem egy komplett rendszer bemutatása a célom – az én esetemben is rajzzal és üzleti logikát felépítő ábrákkal kezdődött. Az alapköveket lehelyező gondolatok és elképzelések után a legapróbb részletekig ki kellett gondolnom, hogy mit akarok megvalósítani.

Ezt követte a megvalósításhoz szükséges technológiák kigondolása, a szükségesnek vélt képességek legmegfelelőbb módjának meghatározása. A kapcsolatot a szerver és az adatbázis között JDBC, EJB technológiával is ki lehetett volna építeni, de én ragaszkodtam a nagyobb rendszer kezeléséhez szükséges összetettebb szelektálások és az egységes kertrendszert biztosító JPA megoldáshoz. Az adatbázis kezelő rendszer lehetett volna MySQL is, tökéletesen teljesíthette volna a kitűzött feladatokat, de a PostgreSQL számos nagyobb volumenű fejlesztéshez szükséges előnyökkel rendelkezik. A PostgreSQL az egyik leggyakrabban használt adatbázisszerver. A Java alkalmazások adatbázis támogatását kiválóan teljesítette. Az adatbázisban tárolt adatok közötti komplettebb kereséseket és szelektálásokat a PostgreSQL felhasználóbarát felülete és könnyű kezelhetősége nagyban megkönnyítette. A PostgreSQL megbízhatóan működött, az elvárásoknak megfelelően.

Alkalmazásszerver esetében szintén egyszerűbb választás lett volna a Tomcat, bizonyos teszt-esetekben ellenőriztem a Tomcat-tel való használatbeli különbségeket. De a GlassFish új, kipróbáltam. A szerver futása nem volt megbízható, bizonyos időközökben nem volt elérhető, bár a szerver futott.

Az érintett fejezetek jól átfogták a szükséges területeket. Átlátható volt a struktúra alapján az alkalmazás fejlesztésének menete, látható volt a rendszer szerkezetének felépülése, előre haladva körvonalazódott annak végső formája. Mindent összevetve a Java alkalmazásfejlesztés komoly idő – és energiaráfordítást igénylő feladat. Egyre többen állnak neki kisebb projectek készítéséhez, ami az iparág egyre nagyobb elterjedéséhez vezet.

4. Irodalomjegyzék

[1] Object Technology International, Inc. „Eclipse Platform Technical Overview”

http://www.eclipse.org/whitepapers/eclipse-overview.pdf

[2] Bruce Momjian, „PostgreSQL GyIK”

http://www.postgresql.org/docs/faqs.FAQ_hungarian.html

[3] Sun Microsystems, Inc. „The GlassFish Community Delivering a Java EE Application Server”

http://glassfish.dev.java.net/faq/v2/GlassFishOverview.pdf

[4] Széplaki Gábor, „Termék nyomon követési rendszerek az élelmiszerláncban”

http://odin.agr.unideb.hu/su2006/studentwork/Szeplaki%20Gabor.pdf

[5] Bátfai Norbert, „Operációs rendszerek 2”

http://www.inf.unideb.hu/~nbatfai/os/DEIK_MIPPOS2_BN_8.pdf

[6] Sun Microsystems, Inc. „JavaServer Faces Technology Overview"

http://java.sun.com/javaee/javaserverfaces/overview.html