SZAKDOLGOZAT

Vígh Gábor

DEBRECEN

2009.

Debreceni Egyetem

Informatika Kar

Témavezető:

Dr. Kuki Attila

adjunktus

Készítette:

Vígh Gábor

Programtervező informatikus (Bsc)

Debrecen

2009.

Webes alkalmazásfejlesztés szabadon választott témában

„Hotel Services”

Tartalomjegyzék

1.Bevezetés

Manapság szinte minden elérhető az interneten, pizzát lehet rendelni, pénzt lehet utalni, repülő és mozijegyeket lehet venni a lehetőségek száma végtelen. Az internet nagy és gyors fejlődésének köszönhetően egyre fontosabbak és egyre nagyobb népszerűségnek örvendenek a webes alkalmazások. Mik is ezek valójában? A szoftverfejlesztés világában Web Alkalmazás egy olyan program amelyet az interneten keresztül érünk el. A webes alkalmazások népszerűsége annak köszönhető hogy a kliens oldaláról csak egy böngészőre van szükség a működéséhez. Az alkalmazás könnyen módosítható és karbantartható anélkül hogy a kliens gép szoftverén bármit is változtatnánk. Manapság a legnépszerűbb webes alkalmazások a Webmail, Webáruház, online aukció, fórum, de ezeken kívül még sok más létezik.

1.1 Témaválasztás okai

A Debreceni Egyetemen szerzett tanulmányaim során megismerkedtem nyelvekkel Java, ANSI C , PL/SQL, SQL ,CLIPS és korábbi tanulmányaimból a Pascal. Ezek közül ami számomra kiemelkedett és a legtöbbet használtam a Java Objektumorientált magas szintű programozási nyelv volt. Azt várom, a szakdolgozat megírásától, hogy rálátásom lesz a Java alapú webes alkalmazásfejlesztésre, amely a későbbiekben sokat segíthet más hasonló technológiák elsajátításában. A programom egy hotelt működtető rendszer. Elsődleges célom, hogy egy hatékony, és jól működő szoftvert készítsek, amely segítségével megkönnyíthetem egy Hotel működését. Úgy gondolom hogy egy jó programnak egyszerűen, és könnyen kezelhetőnek kell lennie, és ez egy nagyon fontos szempont volt a szoftverem fejlesztésénél.

1.2 Használt technológiák és programok

Először is a fejlesztés alapját képző modellt kellett kiválasztanom. Mivel ez egy kis szoftver és saját magam állítottam fel a követelményeket nem biztos hogy teljesek a követelmények, ezért úgy gondolom, hogy a fejlesztésre legmegfelelőbb modell az Evolúciós modell. Egy jó szoftvernél elengedhetetlen, hogy megfelelően tárjuk fel a követelményeket, és sepcifikáljuk azokat. Követelmény feltárásnál nézőpont orientált feltárást és interjú-t alkalmaztam. A félreértések elkerülése érdekében ezeket dokumentálni kell. Elkészítettem a dokumentációkat Áttekintés dokumentum , fogalomszótár, szakterületi kapcsolatok és folyamatok, UML diagrammok, Deployment diagramm Use Case diagramm, Activity diagram**.** A diagrammokat StarUML-ben készítettem el. Megterveztem az adatbázist MySQL WORKBECH-ben ez a program a MySQL egyik tervező programja amelyben igen könnyedén előtudtam állítani a táblák szerkezetét és kapcsolatait. A szoftverhez MySQL adatbázist használok. Az adatbázis kapcsolathoz JDBC(Java Database Connectivity)-t használtam. A webes alkalmazásomhoz a NetBeans által felajánlott GlassFish nyílt forrású vállalati alkalmazásszervert használtam. Az alkalmazásomat az MVC(Model View Control) alapján építettem fel. Jól el különülnek az egyes részek. A View-hoz azaz a megjelenítéshez JSP(Java Server Pages)-t használtam a vezérlést, azaz a Controll-t Servletek segítségével oldottam meg, és végül a Model-t a DAO-DTO tervezési minták alapján építettem fel. Az alkalmazásomat Mozilla Firefox-al teszteltem.

1.3 A szakdolgozat felépítése

A szakdolgozatom első felében bemutatom a rendszerfejlesztéssel kapcsolatos fogalmakat lépéseket majd a továbbiakban leírom, hogy mely technológiákat használom a programom megírásához. A technológiákat röviden elmagyarázom és néhol példákkal illusztrálom. A második felében magát a programomat mutatom be diagrammok és képekkel segítségével működés közben.

2. A rendszerfejlesztés

Egy Rendszerfejlesztés életciklusára vonatkozó lépések a következők:

- Vízió
- Követelmények feltárása
- Elemzés
- Architekturális tervezés
- Tervezés
- Implementálás
- Tesztelés
- Üzembe helyezés
- Üzemeltetés
- Karbantartás
- Evolúció
- Üzemen kívül helyezés

Igyekeztem ezeken a lépéseken végigmenni az alkalmazásom fejlesztése során.

A Vízió. Volt egy elképzelésem, hogy körülbelül hogy nézzen ki az alkalmazásom, hogyan épüljön fel az osztályszerkezet, milyen legyen a felhasználói felület, hogy nézzen ki az adatbázis, csak ezek nagyon vázlatosak voltak. De legelőször egy rendszerfejlesztési modellt kellett választanom.

2.1 Szóba jöhető rendszerfejlesztési modellek

Vízesésmodell:

Ez a modell nem a legmegfelelőbb mivel a követelményeket nem biztos hogy sikerül teljesen feltárnom, tapasztalathiány miatt és későbbiekben ez komoly módosítási problémákat jelenthet számomra, és emiatt kezdhetném előről a tervezést. A másik ok ami a modell ellen szól az az időhiány mivel igen komoly időt kellene fektetni a tervezésbe és időszűkében ez nem megoldható.

Újrafelhasználhatósági modell:

Mivel ez a modell a komponens újrafelhasználhatóságot támogatja, és mivel nekem nincsenek komponenseim ezért ezt a modellt rögtön kizártam.

Inkrementális Modell:

Mivel nem tudom, hogy mik lesznek az inkremenseim ezért ezt a modellt is elvetettem.

Spirális modell:

Mivel nem kell a kockázatkezeléssel foglalkoznom ezért ezt a modellt is kizártam.

Evolúciós modell:

Végül az Evolúciós modell mellett döntöttem. Hogy miért? Nem kell hogy meglegyenek a követelmények teljesen elég ha csak vázlatosan vannak meg. Létrehozok egy kezdeti rendszert amit az újabb követelmények felbukkanásával folyamatosan bővítek és fejlesztek. Nincs olyan hogy a rendszer teljesen ne működne esetleg csak egyes funkciók nem működnek de a folyamatos fejlesztéssel eljutok a végleges verzióig.

2.2 Követelmények feltárása

Nézőpont orientált feltárást és Interjú-t alkalmaztam. Megkérdeztem ismerőseim hogy hogyan képzelnének el egy hotel működtetését végző rendszert majd leírtam őket.

2.3 Elemzés

Ebben a lépésben készítettem el a forgatókönyvet az áttekintés dokumentumot és a use case diagrammot melyeket a későbbiekben láthatunk majd.

2.4 Architekturális tervezés

Ebben a lépésben döntöttem el hogy hogyan épül föl a rendszerem, milyen adatbázist fogok használni, milyen alkalmazásszervert, milyen böngészőt, és elkészítettem a Deployment diagrammot.

2.5 Tervezés

Ebben a részben határoztam el, hogy milyen technológiákat fogunk alkalmazni DAO, DTO tervezési minta, MVC modell, megterveztem az adatbázisomat, és a felhasználói felületet.

3. Alkalmazott eszközök és technológiák leírása

3.1 JAVA

A Java egy objektumorientált programozási nyelv amelyet a Sun Microsystems fejleszt a 90-es évek elejétől kezdve napjainkig. A Java alkalmazásokat jellemzően bytecode formátumra alakítják, de közvetlenül natív (gépi) kód is készíthető Java forráskódból. A bytecode futtatása a Java virtuális géppel történik, ami vagy interpretálja a bytecode-ot vagy natív gépikódot készít belőle és azt futtatja az adott operációs rendszeren. A nyelv első tulajdonsága, az objektum-orientáltság ("OO"), a programozási stílusra és a nyelvstruktúrájára utal. Az OO fontos szempontja, hogy a

szoftvert "dolgok" (objektumok) alapján csoportosítja, nem az elvégzett feladatok a fő szempont. Ennek alapja, hogy az előbbi sokkal kevesebbet változik, mint az utóbbi, így az objektumok (az adatokat tartalmazó entitások) jobb alapot biztosítanak egy szoftverrendszer megtervezéséhez. A cél az volt, hogy nagy fejlesztési projekteket könnyebben lehessen kezelni, így csökken az elhibázott projektek száma. A második tulajdonság, a platformfüggetlenség azt jelenti, hogy a Javában íródott programok hasonlóan fognak futni különböző hardvereken. Ezt úgy lehet megvalósítani, hogy a Java fordítóprogram csak egy úgynevezett *Java gépi kódra* fordítja le a forráskódot, ami aztán futtatva lesz a virtuális gépben, amely lefordítja az illető hardver gépi kódjára. Továbbá, szabványos könyvtárcsomagok léteznek, amelyek elérhetővé teszik az illető hardver sajátságosságait (grafika, szálak és hálózat) egységes módon.

3.2 NETBEANS 6.1

„A NetBeans egy integrált fejlesztői környezet amely a Java nyelven alapul. A program grafikus fejlesztőfelületet kínál a különböző alkalmazások, Appletek vagy akár JavaBeanek elkészítéséhez, amelynek segítségével könnyebben, gyorsabban tudjuk fejleszteni saját programjainkat.”

3.3 MySQL

A MySQL egy többfelhasználós, többszálú, SQL-alapú relációs-adatbázis-kezelő szerver. A szoftver fejlesztője a svéd MySQLAB cég, amely kettős licenceléssel teszi elérhetővé a MySQL-t; választható módon vagy a GPL, vagy egy kereskedelmi licenc érvényes a felhasználásra. 2008 januárjában a Sun felvásárolta 800 millió dollárért a céget. A MySQL az egyik legelterjedtebb adatbázis-kezelő, aminek egyik oka lehet, hogy a teljesen nyílt forráskódú LAMP (Linux–Apache–**MySQL**–PHP) összeállítás részeként költséghatékony és egyszerűen beállítható megoldást ad dinamikus webhelyek szolgáltatására.

3.4 MYSQL WORKBENCH

A Sun Microsystems által készített, vizuális adatbázis-tervező a MySQL Workbench Az eszköz lehetővé teszi az adatbázis adminisztrátor, fejlesztő számára, hogy az vizuálisan tervezhesse meg, hozhassa létre és kezelje az összes adatbázis típust.

3.5 JDBC(Java Database Connectivity)

A **Java Database Connectivity**, röviden a **JDBC** egy API a Java programozási nyelvhez, amely az adatbázishozzáférést támogatja. A JDBC definiálja az adatbázisok lekérdezéséhez és módosításához szükséges osztályokat és metódusokat. A relációs adatmodellhez igazodik. A *Standard Edition* és az *Enterprise Edition* egyaránt tartalmazza a JDBC-t a specifikáció

A JDBC már az 1.1 verziótól kezdve a *Standard Edition* része.

Az adatbázis kapcsolatot a `java.sql` csomag `Connection` osztálya reprezentálja. Ezekkel SQL ___ kifejezéseket lehet készíteni és futtatni. Az SQL kifejezéseket a `Statement` illetve a `PreparedStatement` osztályok reprezentálják. A kifejezések lehetnek lekérdező SELECT kifejezések vagy módosító CREATE, INSERT, UPDATE és DELETE kifejezések, de lehetőség van tárolt eljárások futtatására is a `java.sql.CallableStatement` osztállyal:

Az első lépés az adatbázis kapcsolat létrehozása egy `Connection` példány formájában a `DriverManager.getConnection()` metódus segítségével:

Általánosan:

```
Connection conn = DriverManager.getConnection(
      "jdbc:valamijdbcforgalmazo:további adatok a jdbc
                                 forgalmazótól függően",
      "felhasznalonev",
      "jelszo" );
```

PL:

```
Connection con =(Connection)DriverManager.getConnection(
 "jdbc:mysql://localhost:3306/hotelservices",
 "root",
 "admin");
```

Mostantól kezdve kacsolódva vagyunk az adatbázishoz és műveleteket hajthatunk rajta végre.

A PreparedStatement interfész

A PreparedStatement interfész a Statement interfészt terjeszti ki újabb tulajdonságokkal melyek lehetővé teszik az előfordított utasítások használatát adatbázisunkban. Míg a Statement interfész alkalmazása esetén minden egyes újabb utasítás végrehajtásának alkalmával át kell adni az adatbáziskezelőnek az utasítás teljes szövegét, illetve az adatbáziskezelőnek fel kell dolgoznia azt, addig a PreparedStatement interfész alkalmazása esetén elegendő ezt egy alkalommal megtenni, majd a későbbiekben - az újbóli használat alkalmával - már csak a paramétereket kell átadni az adatbázisban lévő előfordított utasításnak. Ennek a technológiának az előnyét nyilván akkor tudjuk leginkább kamatoztatni, ha ugyanazt az utasítást ciklikusan többször hajtjuk végre egymás után más-más paraméterekkel, mivel az utasításunk előfordított (csőre töltött) állapotban figyel az adatbáziskezelőben, melynek csak a paramétereit kell megadni és már futtatható is.

PreparedStatement létrehozása

Ezért az SQL utasítás szövegét már a PreparedStatement objektumot létrehozó metódus paramétreként meg kell adni, melyben a paraméterek helyét '?' karakterrel kell kijelölnünk. Ez a paraméter átadódik a PreparedStatement interfészt implementáló osztály konstruktorának, mely továbbítja azt az adatbázis-kezelő felé, amely létrehozza az előkészített utasítást. PreparedStatement objektumhoz a Connection objektum prepareStatement metódus meghívásával juthatunk, melynek – mint ahogy a Statement interfész bemutatásánál láthattuk – szintén többféle típusa

van melyekkel speciális, a ResultSet objektummal kapcsolatos tulajdonságokat állíthatunk be.

PreparedStatement futtatása

A PreparedStatement végrehajtása előtt minden kijelölt paramétert inicializálni kell, melyeket a set<típus> kezdetű metódusokkal tehetünk meg úgy, hogy a metódus első paramétreként egy egész számmal hivatkozunk az SQL szövegében kijelölt paraméterre, a metódus második paraméterében pedig megadjuk magát az értéket. A balesetek elkerülése végett, üdvös dolog ha ezt a tevékenységet megelőzi egy clearParameters metódus meghívása, amely törli az összes, korábban beállított értéket. Mindezek után végrehajthatjuk SQL utasításunkat a PreparedStatement objektum execute, executeQuery vagy executeUpdate metódusának megívásával melyeknek különbözőségeiről fentebb írtam.

Egy konkrét példa a PreparedStatement interfész alkalmazására:

```
PreparedStatement stmt = (PreparedStatement)
con.prepareStatement("SELECT * FROM vendeg
                     WHERE FelhasznaloNev = ?
                     AND Jelszo = ?");
stmt.setString(1, nev);
stmt.setString(2, jelszo);
```

Az SQL utasítás eredményét visszaadja egy ResultSet-be és a halmaz elemein egyenként végigmenve feldolgozhatjuk azokat.

```
ResultSet rs = stmt.executeQuery();

while (rs.next()) {
              ;//feldolgozás
}
```

A JDBC objektumok lezárása nagyon fontos, mert az adatbáziskapcsolatok, kifejezések és eredményhalmazok erőforrásokat, például socket-eket és *file descriptor*okat, foglalnak le az operációs rendszerben. Távoli szerver esetében a szerveren kurzorokat is lefoglalhatnak. A nyitva felejtett objektumok váratlan és zavarbaejtő hibákhoz vezethetnek. A JDBC objektumok használatakor ajánlatos követni az alábbi `try-finally` mintát:

```
Statement stmt = conn.createStatement();
try {
    ResultSet rs = stmt.executeQuery( "SELECT * FROM
MyTable" );
    try {
        while ( rs.next() ) {
            // Sorfeldolgozás.
        }
    } finally {
        rs.close();
    }
} finally {
    stmt.close();
}
```

3.6 Servlet

Java servlet-ek segítségével web-es alkalamzások írhatóak Java nyelven. Ezen technológia egyik nagy előnye, hogy nincs minden kérés kiszolgálásakor egy processzindítási többletidő és az azzal járó teljesítménycsökkenés, mint a CGI programok esetében. Másik nagy előnye a szabványos Servlet API által kínált eszköztár, amelyen keresztül a servlet-ek az őket futtató servlet container-rel kommunikálhatnak.

Egy servlet tehát valójában egy olyan speciális Java program, amely szorosan együttműködik egy webszerverrel, így lehetővé teszi a szerveroldalon HTML oldalak

dinamikus létrehozását és paraméterezését. A servletek csakis szerverfunkciókat képesek ellátni.

A CGI megoldásokkal szemben a servletek a következő előnyökkel rendelkeznek

- A CGI szkriptek rendszerint plattformfüggők, ezzel szemben a servletek plattformfüggetlen megoldást biztosítanak,
- sokkal gyorsabb a kiszolgálás, mert a webszerveren állandóan fut egy virtuális gép, így nem kell minden egyes kérés kiszolgálása érdekében egy külső programot indítania a webszervernek,
- a kéréskiszolgálások szinkronizációja leegyszerűsödik,
- ha egy webszerver nagyon le van terhelve, akkor a servlet átirányíthatja a kérést egy másik servletnek.

A servlet API-n keresztül történik a webszerver és a servlet közti kommunikáció. De ez a servlet API nem része a standard JDK-nak, hanem azt külön kell letölteni, ha valaki használni akarja. Egész pontosan a servlet konténerekkel szokott jönni.

Egy servlet életciklusa az alábbi fázisokból áll:

1. A container példányosítja (létrehozza) a servlet objektumot.
2. A container meghívja a servlet példány init() metódusát. Ez a metódus inicializálja a servletet és mindenképp le kell futnia mielőtt a servlet HTTP kéréseket tudna fogadni. Az init() metódus csak egyszer fut le a servlet élete során.
3. Az inicializációt követően a servlet képes a klienseket kiszolgálni. A container minden HTTP kérésre meghívja a servlet service() metódusát. Minden kérés külön szálban hajtódik végre.
4. A servlet életének záróakkordja az, amikor a container meghívja neki a destroy() metódusát. Az init() metódushoz hasonlóan a destroy() is csak egyszer hajtódik végre a szerver életében.

3.7 JavaServer Pages

A **JavaServer Pages** (röviden **JSP**) egy technológia, melynek segítségével a szoftverfejlesztő dinamikusan tud generálni HTML, XML vagy egyéb dokumentumokat HTTP kérésekre reagálva. A JSP tekinthető a szervlet réteg feletti absztrakciós szintnek. A JSP oldalból java servlet forráskód generálódik. A JSP 2006 májusa óta a J2EE specifikáció része.

Egy JSP oldal a következő nyelvi elemeket tartalmazhatja:

- statikus adat, például HTML kód
- direktívák
- szkriptelemek és változók
- akciók
- elemkönyvtárakban definiált *tag*-ek

3.8 Három rétegű alkalmazás az MVC (Model View Controll)

Az MVC mint látható 3 szóból tevődik össze és mindegyiknek önálló jelentése van (Model View Controll). Az MVC egy architektúrális minta amely 1979-ben született és azóta sok területen sikeresen alkalmazzák. Sokszor előfordul az hogy változtatni szeretnénk valamin például a program kinézetén vagy az adatmodellt szeretném megváltoztatni egy szoftver esetén. Az adathozzáférés és az ún. üzleti logika elválik az adat prezentációjától, melyet egy köztes komponens bevezetésével érünk el, ezt hívjuk Controller-nek.

Model :

Ez a rész felelős az üzleti logikáért. A réteg feladata, hogy leírja az adatszerkezeteket (melyeket használni fogunk az alkalmazásban) és definiálni azokat a szabályokat, melyek alapján elérhetjük azokat. Abban az esetben ha az alkalmazásunk egy

adatbázisból veszi az adatokat (általában ez így szokás) és ennek az adatbázisnak változtatjuk a szerkezetét, vagy netán lecseréljük egy másik adatbázisra (pl. MySQL helyett Oracle), akkor jobb esetben csak ezt a réteget kell módosítanunk, de ami a leges legfontosabb, hogy nem kell emiatt módosítanunk a user interface-t.

View :

A View a megjelenítéssel foglakozó réteg, és ebben rejlik az MVC-nek az igazi ereje mert a megjelenítés (View) teljesen elszeparálódik az adatkezeléstől. A view dolga hogy megjelenítse a Modell teljes tartalmát. A legfontosabb alapszabály: az adatokat csak a Model osztályainak példányain keresztül érhetjük el és módosíthatjuk! Nagyon fontos hogy a View-ba ne keveredjen bele az üzleti logika és ha ezt betartjuk akkor egy nagyon könnyen módosítható rendszert kapunk hiszen ha egy másik megjelenítést szerenénk akkor nem kell kidobnunk az egész alkalmazást hanem elegendő csak ezt a réteget lecserélni egy jobbra modernebbre.

Controll :

Rendben van egy Modell és egy View rétegünk de ezez önmagukban csak úgy vannak valahogyan kommunikálniuk is kellene ezért felelős a Controll réteg. Bizonyos eseményeket pl: kattintás átfordítja egy Modell által végrehajtandó eseményre és az akció bekövetkezte utáni változtatásokat megjeleníti a View réteg.
A Controll réteg nem mindig van külön megvalósítva előfordulnak olyan esetek Pl: vasatag kliens esetén amikor a Controll réteg összeolvad a View-vel.

3.9 DAO, DTO tervezési minták

Minden rendszer alapja az adat, amivel a rendszer dolgozik. Az adatokat egy üzleti rendszernél üzleti entitásokba fogjuk össze, ahol a tényleges adat az entitás egy-egy tulajdonsága. Ezeket az entitásokat adatbázis entitásokká alakítjuk és így mentjük őket az adatbázisba, hogy később, ha szükség van rájuk, elő tudjuk venni őket.

A **DAO** (Data Access Object) osztályok, amelyeknek feladata az Entity-k kezelése. A lényege, hogy elrejti a feljebb található rétegek elől az adatbázis felépítését, hiszen azok a DAO egy-egy funkcióját hívják meg, így lépve kapcsolatba az adatbázissal, arról nincs információjuk, hogy a nekik szükséges információ honnan, milyen módon kerül hozzájuk. Ha a program rétegei nem mosódnak egybe, akkor a JDBC utasítások csak itt a DAO-ban szabad, hogy megjelenjenek.

A DTO (Data Transfer Object) lényege, hogy egy adott üzleti entitás adatait foglalja egy egységbe, mert így egyrészt az entitás adatai egyben kezelhetők, másrészt jelentős erőforrást takaríthatunk meg. Ha nem használnánk DTO-t, akkor minden adatot külön-külön lekérdezhetővé kellene tenni, ami rengeteg függvény megírását jelenti, ráadásul mivel ezeket a kliens (GUI) rétegbe is le kell juttatni, ezért minden lekérés külön hálózati forgalmat generál.

Itt jön a lényeg – egy kisebb rendszernél még meg lehet csinálni, hogy az üzleti entitásaink és az adatbázis entitásaink ugyanazok legyenek, de egy nagy rendszernél ezt nem szabad elkövetni – márpedig ahogy a neve is mutatja Java Enterprise Edition-nel dolgozunk, vagyis nem kisebb rendszert fejlesztünk. Azért nem szabad ezt elkövetni, mert egyrészt így nagyon sok lesz a redundáns adat, másrészt az üzleti entitások felépítése nem megfelelő, hogy az adatbázis szerver teljesítményét megfelelően kihasználjuk.

4. Áttekintés dokumentum – HotelServices

I. Általános leírás

A HotelServices egy Hotelek, Szállodák által használt számítógépes rendszer, mely egy hotel számára könnyíti meg a működést webes felületen keresztül lehet hirdetni rendezvényeket, takarítást kérni egy adott szobába, a szálloda által kiírt menüt lehet igényelni.

A rendszert ötféle személy használhatja.

–A rendszer webes felületén keresztül a vendégnek lehetősége van szobát kivenni a hotelben, ételt rendelni, takarítást kérni és a hotelben rendezett rendezvényeken részt venni.
–A Szakács tervezi meg a heti menüt és kiírhatja amit a rendszerben látnak a vendégek vagy törölheti a menüt.
–A takarítók látják a webes felületen keresztül hogy mely szobák várnak takarításra és látják hogy hova kell menniük takarítani.
–A Recepciós a szállodába érkező vendégeket tudja regisztrálni szobát kiadni nekik és takarítást kiírni egy adott szobához.
–A Szállodavezető képes regisztrálni az alkalmazottakat(Szakács, Takarító, Recepciós) és elbocsájtani is azokat, tud rendezvényeket hirdetni és törölni.

A rendszer összes felhasználóját a rendszer használatához regisztrálni kell.

II. Általános követelmények

1. A kivitelező köteles minden üzleti logikához és felületi elemhez tartozó döntésükről kikérni cégünk egyetértését, hozzájárulását. A fejlesztés minden lényeges pontja, a fejlesztési eredmények, és a fejlesztés teljes dokumentációja elérhető kell hogy legyen a megrendelő számára. A fejlesztés végeztével a forráskód és a teljes dokumentáció tulajdonjoga a megrendelőt illeti meg.
2. A szoftverfejlesztés minden lépéséről származó dokumentumok a megrendelő számára hozzáférhetők kell, hogy legyenek. A dokumentálás "PDF" formátumban kell történnie.
3. Alkalmazottaink egész napos munkájukat gépnél töltik, ezért lényeges a felületek szép kinézete, és a szemet nem zavaró színek, zavaró funkcióelrendezések kerülése.
4. A rendszer minden művelet eredményéről tájékoztatja a szoftver használóját. Hiba esetén értesítést ad a probléma okáról a felhasználó minőségétől függően.
5. A rendszer minden kommunikációja biztonságos csatornán kell, hogy történjen.

6.Barátságos megjelenítés, színek.

III.Rendszerkövetelmények

Operációs rendszer: WindowsXP.

Szoftverfejlesztési Technológia: Java Eneterprise Editidion

Célhardver: A szoftver igényeihez fognak igazodni. Szoftver tervezésekor nem kell figyelembe venni.

Várható felhasználók száma: 1000 fő

5. Fogalomszótár – HotelServices

vendég- a hotel szolgáltatásait igénybe vevő személy(ek).

szakács – a hotelben dolgozó személy aki a napi menü kiírására jogosult.

Menü – 3 fogásos étel ami áll egy előételből, főételből és egy desszertből.

szállodavezető – az a személy aki vezeti a szállodát és rendezvények kiírására jogosult és alkalmazottakat vehet fel ill. regisztrálhat.

rendezvény – a szállodában meghirdetett konkrét időponttal egy adott esemény.

alkalmazott- a szálloda dolgozói (Szakács, Recepciós, Takarító).

takarító – az a személy aki a szálloda tisztaságáért felelős.

recepciós – a portán felügyelő személy aki az új vendégeket fogadja és információval látja el őket. Vehet fel a szállodába vendégeket.

szoba – olyan helység ahol a vendégek megszállhatnak.

férőhely – megmondja hogy hány személy veheti igénybe a szobát.

napok száma – mennyi időre veszi ki a vendég a szobát.

6. Szakterületi kapcsolatok és folyamatok

Szobafoglalás – az a folyamat amely során a vendég lefoglal egy szobát bizonyos időre és a innentől kezdve igénybe veheti a hotel szolgáltatásait.

Étel igénylés – az a folyamat mely során a vendég az ízlésének megfelelő ételt rendeli meg és fogyasztja majd el.

Takarítás igénylése - az a folyamat, amely során a vendég a szobájába takarítást kérhet melyet majd egy takarító fog elvégezni.

Program igénylés - az a folyamat, amely során a vendég a szállodában meghirdetett programokra jelentkezik vagy már jelentkezetteket lemond.

Takarítás - az a folyamat, amely során a takarító kiválasztja a takarításra váró szobát és elvégzi a kívánt tevékenységet.

Program kiírása - az a folyamat, amely során a szállodavezető meghirdet egy eseményt amelyre lehet ezek után jelentkezni.

Program törlése - az a folyamat, amely során a szállodavezető töröl egy eseményt amelyet korábban meghirdetett.

Alkalmazott felvétele - az a folyamat, amely során a szállodavezető regisztrál egy alkalmazottat és ezután a saját felhasználónevével beléphet a rendszerbe.

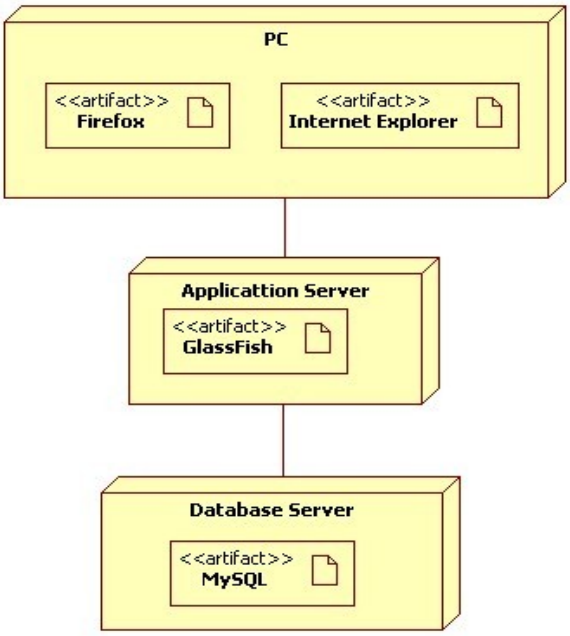
Alkalmazott elbocsájtása - az a folyamat, amely során a szállodavezető töröl egy alkalmazottat és ezután az alkalmazott már nem tud belépni a rendszerbe.

Napi menü kiírása - az a folyamat, amely során a szakács meghirdeti a napi menüt melyet ezután a vendégek igényelhetnek.

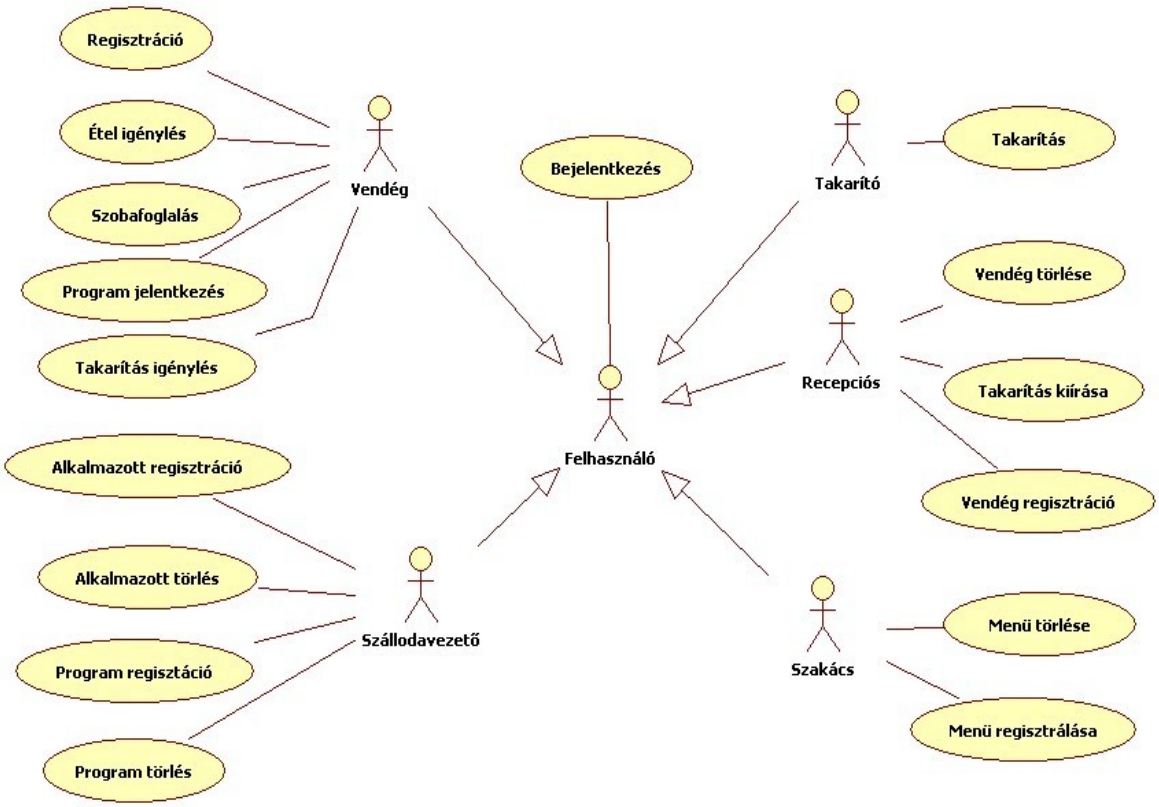
Napi menü törlése - az a folyamat, amely során a szakács törölheti az eddig meghirdetett menüket.

7. UML Diagrammok

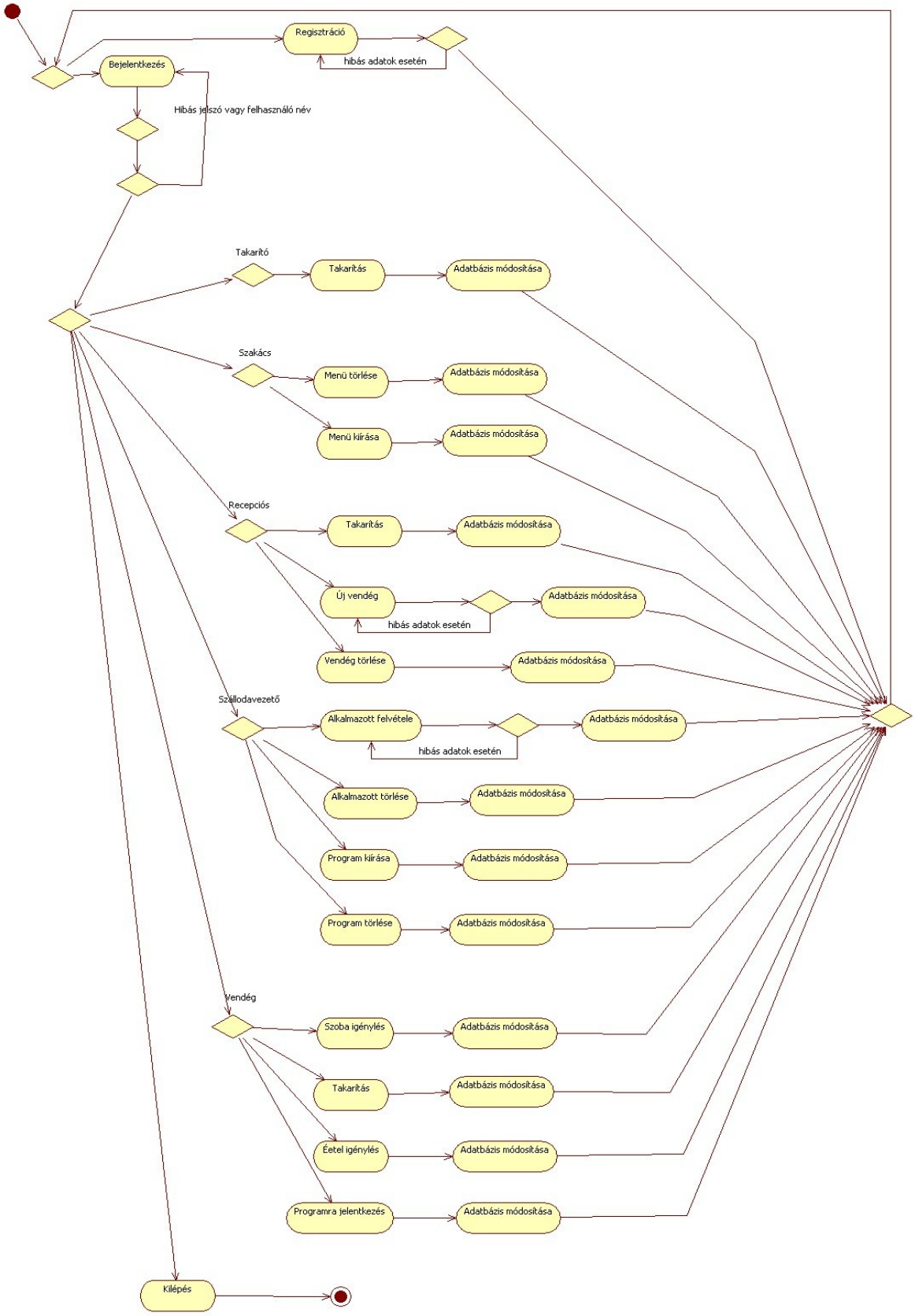
7.1 Deployment Diagram:



7.2 Use Case Diagramm:



7.3 Activity Diagramm:



8. Adatbázis fizikai szerkezete magyarázattal

Az adatbázisom 12 táblából épül fel ebben a részben részletezem hogy mely táblámnak mi a funkciója és hogyan néz ki a fizikai szerkezete.

A tábláim :

- Recepcios
- Vendeg
- Takarito
- Szakacs
- Szallodavezeto
- Etel
- Etel_Vendeg
- Szoba
- Szoba_Vendeg
- Takaritas
- Programok
- Programok_Vendeg

8.1 A táblák leírása:

Recepciós

Oszlop Név	Típus	Not Null	Default érték
idRecepcios	INT	Igen	-
FelhasznaloNev	INT	Nem	-
Jelszo	VARCHAR(45)	Nem	-
Nev	VARCHAR(45)	Nem	-
Iranyitoszam	INT	Nem	-
Cim	VARCHAR(45)	Nem	-
Telefonszam	VARCHAR(45)	Nem	-

Vendég

Oszlop Név	Típus	Not Null	Default érték
idRecepcios	INT	Igen	-
FelhasznaloNev	INT	Nem	-
Jelszo	VARCHAR(45)	Nem	-
Nev	VARCHAR(45)	Nem	-
Iranyitoszam	INT	Nem	-
Cim	VARCHAR(45)	Nem	-
Telefonszam	VARCHAR(45)	Nem	-

Takarító

Oszlop Név	Típus	Not Null	Default érték
idRecepcios	INT	Igen	-
FelhasznaloNev	INT	Nem	-
Jelszo	VARCHAR(45)	Nem	-
Nev	VARCHAR(45)	Nem	-
Iranyitoszam	INT	Nem	-
Cim	VARCHAR(45)	Nem	-
Telefonszam	VARCHAR(45)	Nem	-

Szakács			
Oszlop Név	**Típus**	**Not Null**	**Default érték**
idRecepcios	INT	Igen	-
FelhasznaloNev	INT	Nem	-
Jelszo	VARCHAR(45)	Nem	-
Nev	VARCHAR(45)	Nem	-
Iranyitoszam	INT	Nem	-
Cim	VARCHAR(45)	Nem	-
Telefonszam	VARCHAR(45)	Nem	-

Szállodavezető			
Oszlop Név	**Típus**	**Not Null**	**Default érték**
idRecepcios	INT	Igen	-
FelhasznaloNev	INT	Nem	-
Jelszo	VARCHAR(45)	Nem	-
Nev	VARCHAR(45)	Nem	-
Iranyitoszam	INT	Nem	-
Cim	VARCHAR(45)	Nem	-
Telefonszam	VARCHAR(45)	Nem	-

Magyarázat:

Ez az öt tábla szerkezetileg megegyezik, viszont logikailag teljesen más csoportba tartoznak ezért jobbnak láttam a különböző szerepkörű felhasználókat külön táblába gyűjteni.

`idRecepcios` : ezt az értéket én generálom le úgy hogy biztosan ne legyen még egy ilyen érték a táblában ez az elsődleges kulcs.

`FelhasznaloNev` : A felhasználó nevét tárolom el amivel be tud majd jelentkezni a rendszerbe.

`Jelszo` : Ahhoz hogy egy adott felhasználó be tudjon lépni a rendszerbe szüksége van egy azonosítóra és egy jelszóra itt tárolom a jelszót.

`Nev` : A felhasználónak az igazi neve család és keresztnév.

`Iranyitoszam` : Az állandó lakhelyének irányítószáma.

`Cím` : Az állandó lakhelyének a címe.

`Telefonszam` : A felhasználó telefonszáma.

Etel

Oszlop Név	Típus	Not Null	Default érték
idEtel	INT	Igen	-
Nap	VARCHAR(45)	Nem	-
Menu	VARCHAR(45)	Nem	-
Eloetel	VARCHAR(45)	Nem	-
Foetel	VARCHAR(45)	Nem	-
Desszert	VARCHAR(45)	Nem	-
Szakacs_idSzakacs	INT	Igen	-

Magyarázat:

`idEtel` : ezt az értéket én generálom le úgy hogy biztosan ne legyen még egy ilyen érték a táblában az étel azonosítója.

`Nap` : mely napra szeretném ezt a menüt készíteni.

`Menu` : Többféle menüt lehet összeállítani egy napra és ez azonosítja pl: „A”, „B” vagy „Vegetáriánus” ...

`Eloetel, Foetel, Desszert` : Maguk az ételek.

`Iranyitoszam` : Az állandó lakhelyének irányítószáma.

`Szakacs_idSzakacs` : A menüt a Szakács írja ki, ez egy külső kulcs amivel össze van kapcsolva a Szakács táblával hogy melyik szakács írta ki a menüt.

Etel_Vendeg

Oszlop Név	Típus	Not Null	Default érték
IdEtel_Vendeg	INT	Igen	-
Fo	VARCHAR(45)	Nem	-
Etel_idEtel	INT	Igen	-
Vendeg_idVendeg	INT	Igen	-

Magyarázat:

Ez egy kapcsolótábla mely összekapcsolja hogy mely vendég hány főre rendelt ételt és milyen ételt.

`IdEtel_Vendeg` : ezt az értéket én generálom le úgy hogy biztosan ne legyen még egy ilyen érték a táblában elsődleges kulcs.

`Fo` : Hány személyre rendelik meg a menüt.

`Etel_idEtel` : A menü azonosítója hogy mely menüt rendeljük meg.

`Vendeg_idVendeg` : külső kulcs, mely vendég rendelte meg a menüt.

Szoba			
Oszlop Név	**Típus**	**Not Null**	**Default érték**
idSzoba	INT	Igen	-
Ferohely	INT	Nem	-
Ar_Ejszaka	INT	Nem	-

Magyarázat:

Ez a tábla tartalmazza a szobákat.

`idSzoba` : ezt az értéket én generálom le úgy hogy biztosan ne legyen még egy ilyen érték a táblában ez az elsődleges kulcs.

`Ferohely` : Hány személyes a szoba.

`Ar_Ejszaka` : mennyibe kerül a szoba egy éjszakára.

Szoba_Vendeg			
Oszlop Név	**Típus**	**Not Null**	**Default érték**
idSzoba_Vendeg	INT	Igen	-
Napok	INT	Nem	-
Szoba_idSzoba	INT	Igen	-
Vendeg_idVendeg	INT	Igen	-

Magyarázat:

Ez a tábla egy kapcsolótábla és azt definiálja hogy mely vendég mely szobát vette ki.

`idSzoba_Vendeg` : ezt az értéket én generálom le úgy hogy biztosan ne legyen még egy ilyen érték a táblában ez az elsődleges kulcs.

`Napok` : Hány napra bérli ki a vendég a szobát.

`Szoba_idSzoba` : ez egy külső kulcs és azt mondja meg hogy mely szobára vonatkoznak az értékek.

`Vendeg_idVendeg` : ez egy külső kulcs és azt mondja meg hogy mely vendég vette ki a szobát.

Takaritas

Oszlop Név	Típus	Not Null	Default érték
idTakaritas	INT	Igen	-
Szobaszam	INT	Nem	-
Leiras	VARCHAR(500)	Nem	-
Vendeg_idVendeg	INT	Igen	-

Magyarázat:

Ez a tábla tartalmazza a takarításra váró szobák listáját.

`idTakaritas` : ezt az értéket én generálom le úgy hogy biztosan ne legyen még egy ilyen érték a táblában ez az elsődleges kulcs.

`Szobaszam` : mely szobát kell takarítani.

`Leiras` : itt leírhatjuk a szükséges információkat a takarítók számára

`Vendeg_idVendeg` : ez egy külső kulcs és azt mondja meg hogy mely vendég kért takarítást.

Programok			
Oszlop Név	**Típus**	**Not Null**	**Default érték**
idProgramok	INT	Igen	-
Programnev	VARCHAR(45)	Nem	-
Datum	DATE	Nem	-
Ido	TIME	Nem	-
Szallodavezeto_idSzallodavezeto	INT	Igen	-

Magyarázat:

Ez a tábla tartalmazza a rendezvényeket melyeket a szállodavezető meghírdet.

`idProgramok` : ezt az értéket én generálom le úgy hogy biztosan ne legyen még egy ilyen érték a táblában ez az elsődleges kulcs.

`Programnev` : a rendezvény neve.

`Datum` : a rendezvény dátuma.

`Ido` : a rendezvény időpontja.

`Szallodavezeto_idSzallodavezeto` : külső kulcs és azt mondja meg hogy mely szállodavezető írta ki a programot.

Programok_Vendeg			
Oszlop Név	**Típus**	**Not Null**	**Default érték**
`idProgramok_Vendeg`	`INT`	`Igen`	-
`Fo`	`INT`	`Nem`	-
`Vendeg_idVendeg`	`INT`	`Igen`	-
`Programok_idProgramok`	`INT`	`Igen`	-

Magyarázat:

Ez a tábla egy kapcsolótábla és azt tartalmazza, hogy mely vendég hány fővel mely programon kíván részt venni.

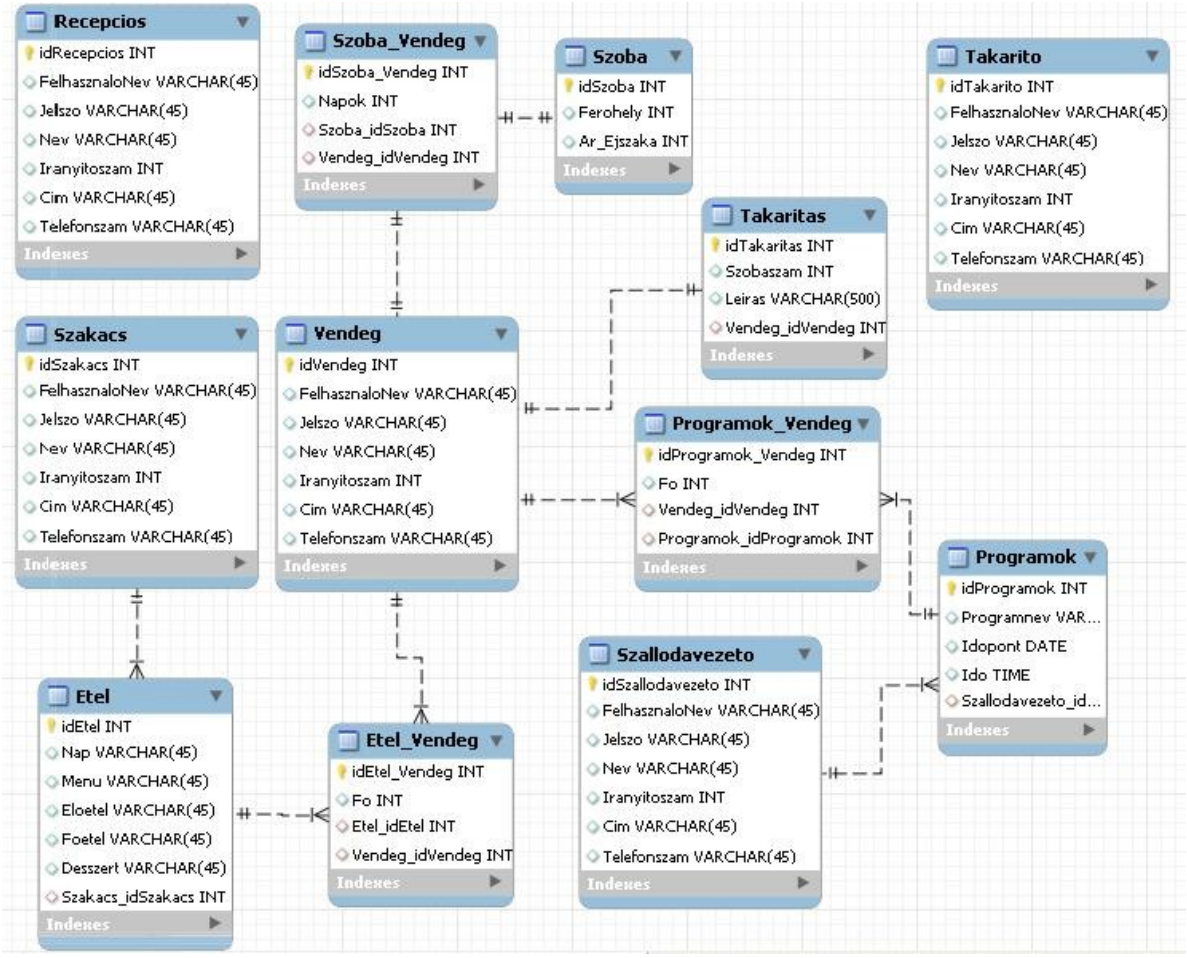
`idProgramok_Vendeg` : ezt az értéket én generálom le úgy hogy biztosan ne legyen még egy ilyen érték a táblában ez az elsődleges kulcs.

`Fo` : hány fővel jelentkezik a vendég a rendezvényre.

`Vendeg_idVendeg`: a vendég azonosítója mely vendég vesz részt a rendezvényen.

`Programok_idProgramok` : külső kulcs a program azonosítója.

8.2 A táblák kapcsolata:



9. Implementáció (Részlet)

Bejelentkezés JSP

```
<%--
    Document   : bejelentkezes
    Created on : 2009.03.14., 12:29:15
    Author     : Vígh Gábor
--%>

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
        <title>JSP Page</title>
        <style type="text/css">

.style1 {
         font-family: Georgia, "Times New Roman", Times, serif;
         font-weight: bold;
         font-size: 30pt;
         color: #ECE9D8;
         font-style: italic;
}
.style7 {font-size: 14pt; font-weight: bold; color: #000000; font-style:
italic; }
        .style8 {font-size: 12pt}
</style>
        <SCRIPT LANGUAGE="JavaScript">
                function bejelentkezes(){
                       javascript:window.location.href='bejelentkezes.jsp';
                }
                function regisztracio(){
```

```
                javascript:window.location.href='regisztracio.jsp';
            }
            function rendszerrol(){
                javascript:window.location.href='rendszerrol.jsp';
            }
            function kezdolap(){
                javascript:window.location.href='index.jsp';
            }
        </SCRIPT>
</head>
    <body  bgcolor="#FFFFCC" style="background-image: url('back.jpg');
    background-position: top center; background-repeat: no-repeat" >
    <form action="MainServlet" method="POST">
        <h2 align="center" class="style1">Bejelentkezés!</h2>
        <p align="center"> </p>
        <table width="820" height="27" border="1" align="center"
        bgcolor="#DDD8C1">
            <tr>
              <td onmouseover="this.style.color='red'"
onmouseout="this.style.color='black'"><div align="center" onclick
="kezdolap()"><strong>Kezdőlap</strong></div></td>
              <td onmouseover="this.style.color='red'"
onmouseout="this.style.color='black'"><div align="center" onclick
="bejelentkezes()"><strong>Bejelentkezes</strong></div></td>
              <td onmouseover="this.style.color='red'"
onmouseout="this.style.color='black'"><div align="center" onclick
="regisztracio()"><strong>Regisztráció</strong></div></td>
              <td onmouseover="this.style.color='red'"
onmouseout="this.style.color='black'"><div align="center" onclick
="rendszerrol()"><strong>Rendszerről</strong></div></td>
            </tr>
         </table>

        <p> </p>
        <table width="223" border="1" align="center">
          <tr>
            <td><img src="hotel4.jpg" width="159" height="88"></td>
          </tr>
          <tr>
```

```
            <td class="style7"><div align="right"
class="style3">Felhasználónév:</div></td>
            <td><input type="text" name="felhasznalonev" value=""
size="15"/></td>
          </tr>
          <tr>
            <td class="style7"><div align="right"
class="style3">Jelszó:</div></td>
            <td><input type="password" name="jelszo" value="" size="15"/></
td>
          </tr>
    </table>
        <p> </p>
        <div align="center">
          <input type="submit" value="Bejelentkezes" name="bejelentkezik"
onmouseover="this.style.color='red'" onmouseout="this.style.color='black'"/
>
        </div>
    </form>
    </body>
</html>
```

Ez a kódrészlet azt csinálja, hogy egy stílust definiálok és nem kell mindig külön külön beállítani.

```
.style1 {
          font-family: Georgia, "Times New Roman", Times, serif;
          font-weight: bold;
          font-size: 30pt;
          color: #ECE9D8;
          font-style: italic;
}
```

Használok minimális Java scriptet. Ezek függvények melyek átirányitják egy másik jsp re a felhasználót. Azért így oldottam meg mert szerettem volna ha amikor a szövegre húzom az egeret akkor más színben tünjön fel.

```
<SCRIPT LANGUAGE="JavaScript">
                function bejelentkezes(){
                        javascript:window.location.href='bejelentkezes.jsp';
                }
                function regisztracio(){
                        javascript:window.location.href='regisztracio.jsp';
                }
                function rendszerrol(){
                        javascript:window.location.href='rendszerrol.jsp';
                }
                function kezdolap(){
                        javascript:window.location.href='index.jsp';
                }
</SCRIPT>


<td onmouseover="this.style.color='red'"
onmouseout="this.style.color='black'"><div align="center" onclick
="kezdolap()"><strong>Kezdőlap</strong></div></td>
```

Az onmouseover és onmouseout opciók az onclick eseménnyel csak így működtek.

```
<tr>
   <td class="style7"><div align="right" class="style3">Jelszó:</div></td>
   <td><input type="password" name="jelszo" value="" size="15"/></td>
</tr>
```

A <tr></tr> teg között a táblázat egy sorát adjuk meg. A <td></td> teg között a táblázat egy oszlopát adjuk meg. Az align-al a pozíciót állíthatjuk. A beviteli mező tipusa lehet text vagy password text esetén szöveget jelenít meg password esetén pedig csillagokat.

A bejelentkezés oldal.

A bejelentkezés gomb által kiváltott eseményt a MainServlet kezeli le.

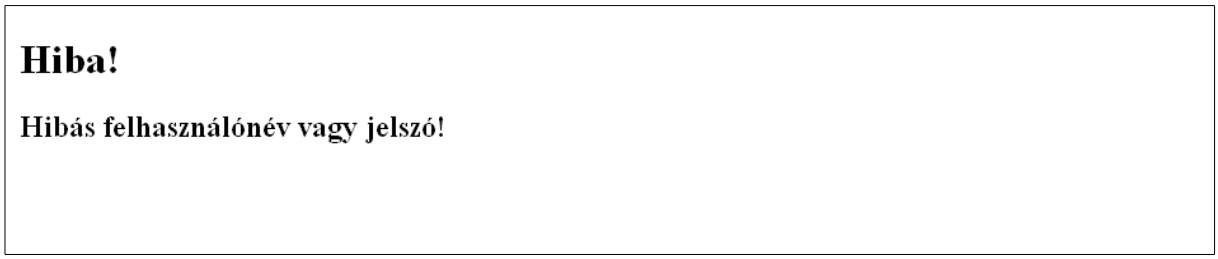
MainServlet

```
try {
            if (request.getParameter("bejelentkezik") != null &&
request.getParameter("bejelentkezik").equals("Bejelentkezes")) {
                if (request.getParameter("felhasznalonev") != null &&
request.getParameter("jelszo") != null) {
                    int i = new
Bejelentkezik().felhasznalo(request.getParameter("felhasznalonev"),
request.getParameter("jelszo"));
                    switch (i) {
                        case 1: {
                            response.sendRedirect("vendeg_index.jsp");
                            String fNev =
                            request.getParameter("felhasznalonev");
                            request.getSession().setAttribute("fnev",
fNev);
```

```
                        break;
                    }
                    case 2: {
                        response.sendRedirect("szakacs_index.jsp");
                        String fNev =
                        request.getParameter("felhasznalonev");
                        request.getSession().setAttribute("fnev",
fNev);
                        break;
                    }
                    case 3: {
                        response.sendRedirect("szallodavezeto_index.jsp
");
                        String fNev =
                        request.getParameter("felhasznalonev");
                        request.getSession().setAttribute("fnev",
fNev);
                        break;
                    }
                    case 4: {
                        response.sendRedirect("recepcios_index.jsp");
                        String fNev =
                        request.getParameter("felhasznalonev");
                        request.getSession().setAttribute("fnev",
fNev);
                        break;
                    }
                    case 5: {
                        response.sendRedirect("takarito_index.jsp");
                        String fNev =
                        request.getParameter("felhasznalonev");
                        request.getSession().setAttribute("fnev",
fNev);
                        break;
                    }
                    default: {
                        response.sendRedirect("hiba01.jsp");
                        break;
                    }
```

```
                    }
                }
            }
        } finally {
            out.close();
        }
```

Leellenőrzöm hogy a beviteli mezők kivannak-e töltve ha nincsenek hiba, egyébként megnézem, átadom a mezők értékeit a Bejelentkezik() metódusnak ami 1-et ad vissza ha Vendég 2-t ha Szakács 3-at ha Szállodavezető 4-et ha Recepciós 5-öt ha Takarító egyébként pedig 0.



A bejelentkezik metódusom

```
public class Bejelentkezik {
    public int felhasznalo(String nev, String jelszo){
    Connection con = null;
        try {
            Class.forName("com.mysql.jdbc.Driver");

            con = (Connection)
DriverManager.getConnection("jdbc:mysql://localhost:3306/hotelservices",
"root", "admin");

            //System.out.println("kajshkjashd");
            PreparedStatement stmt = (PreparedStatement)
con.prepareStatement("SELECT * FROM vendeg where FelhasznaloNev = ? and
Jelszo = ?");
            stmt.setString(1, nev);
            stmt.setString(2, jelszo);
```

```
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                return 1;
            }

            stmt = (PreparedStatement) con.prepareStatement("SELECT * FROM
szakacs where FelhasznaloNev = ? and Jelszo = ?");
            stmt.setString(1, nev);
            stmt.setString(2, jelszo);
            rs = stmt.executeQuery();
            if (rs.next()) {
                return 2;
            }
            stmt = (PreparedStatement) con.prepareStatement("SELECT * FROM
szallodavezeto where FelhasznaloNev = ? and Jelszo = ?");
            stmt.setString(1, nev);
            stmt.setString(2, jelszo);
            rs = stmt.executeQuery();
            if (rs.next()) {
                return 3;
            }
            stmt = (PreparedStatement) con.prepareStatement("SELECT * FROM
recepcios where FelhasznaloNev = ? and Jelszo = ?");
            stmt.setString(1, nev);
            stmt.setString(2, jelszo);
            rs = stmt.executeQuery();
            if (rs.next()) {
                return 4;
            }
            stmt = (PreparedStatement) con.prepareStatement("SELECT * FROM
takarito where FelhasznaloNev = ? and Jelszo = ?");
            stmt.setString(1, nev);
            stmt.setString(2, jelszo);
            rs = stmt.executeQuery();
            if (rs.next()) {
                return 5;
            }

        } catch (java.lang.Exception ex) {
```

```
            ex.printStackTrace();
        }
    return 0;
        }
```

A PreparedStatement, és Connection, használatának magyarázata a JDBC részben van elmagyarázva.

10. Összefoglalás

A webes alkalmazásom fejlesztése során számos problémával találkoztam. Hogyan használjam hatékonyan a szervleteket, hogyan építsem fel az adatbázisomat, hogyan tervezzem meg a felhasználói felületet. Az alkalmazásom elkészültével számos dolgot másképp csinálnék, úgy gondolom, hogy az alkalmazásom igen alap funkciókkal rendelkezik és komoly használatra nem alkalmas, és továbbfejlesztésre szorul. Mindezek mellett igyekeztem egy könnyen módosítható szoftvert készíteni és úgy gondolom, hogy ezt a célt sikerült elérnem. Az alkalmazásomhoz elkészült 13 interface 13 tárolóosztály(DTO), és 15 tároló osztályt kezelő osztály(DAO), és 26 JSP oldal is. Véleményem szerint a célomat sikerült elérnem az alkalmazásom fejlesztése során, alap szinten elsajátítottam a JSP-k, a Servlet-ek és a JDBC használatát. Megismerkedtem alaposabban az MVC mdellel és a DAO, DTO mintákkal.

11. Használt Irodalmak

- Rendszerfejlesztés technológiája előadás jegyzet (2008/2009)
- Nyékyné Gaizler Judit Java2 Útikalauz programozóknak (1.3)
- Interneten található tutorialok.
- http://jankajanos.spaces.live.com/Blog/cns!C3E2695FC6F7B0A4!394.entry
- http://hu.wikipedia.org/wiki/Java_(programoz%C3%A1si_nyelv)

- http://hu.wikipedia.org/wiki/NetBeans
- http://hu.wikipedia.org/wiki/MySQL
- http://hup.hu/cikkek/20080419/mysql_workbench
- http://hu.wikipedia.org/wiki/JDBC
- http://www.javaforum.hu/javaforum/7/cikkek/cikkek/15/show/statement
- http://hu.wikipedia.org/wiki/Servlet
- http://hu.wikipedia.org/wiki/JSP
- http://csujo.blog.hu/

12. Köszönetnyilvánítás:

- Dr. Kuki Attilának, a lehetőségért, hogy a szakdolgozatomat nála írhattam.
- Szauervein Szabolcs-nak a segítőkészségéért.
- A szüleimnek a biztatásért.

13. Függelék

A főoldal:

A vendég kezdőoldala:

Ha a vendég takarítást szeretne kérni:

Ha a vendég szobát szeretne foglalni:

Ha a vendég ételt igényel bizonyos napokra:

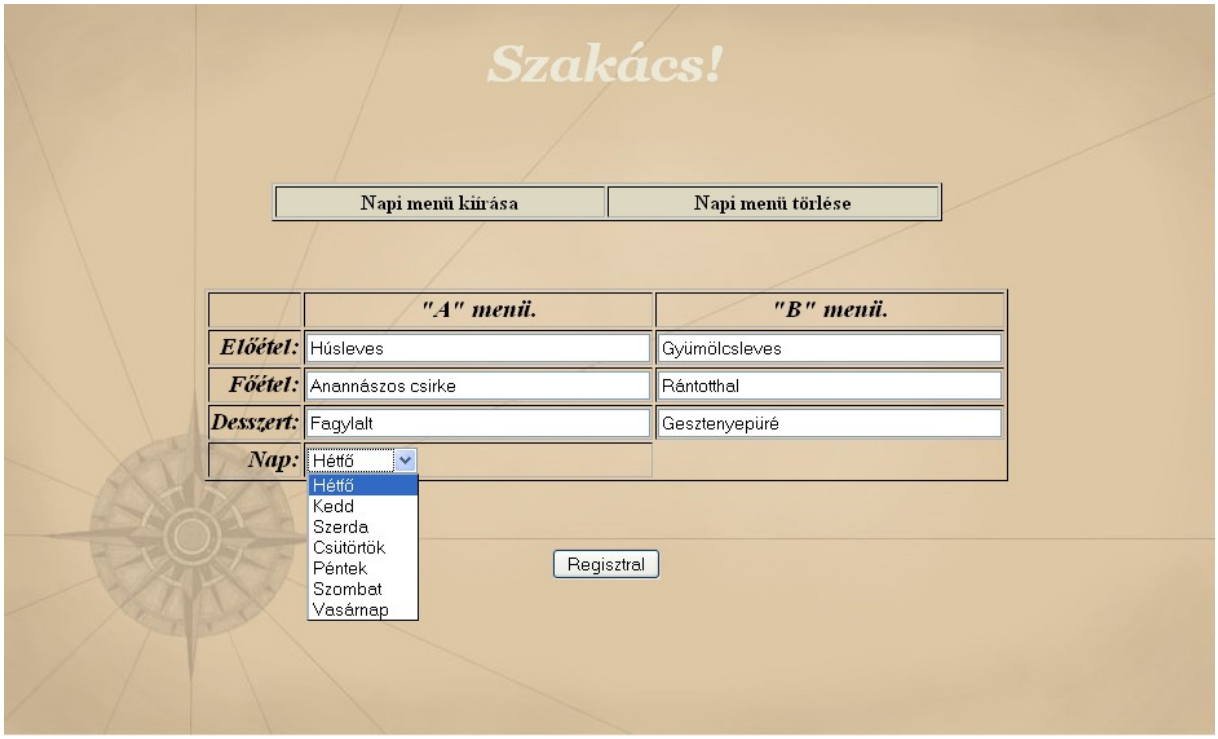
A szállodavezető kezdőlapja:

Ha a szállodavezető új alkalmazottat vesz fel:

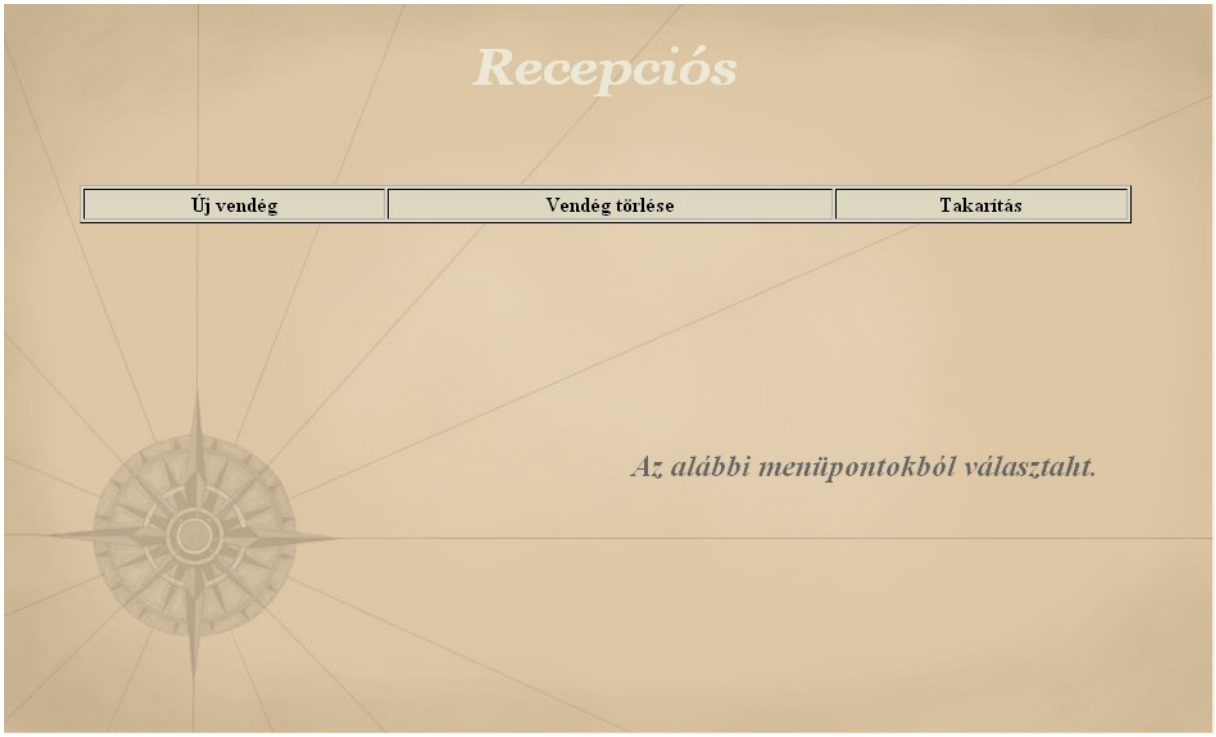
Ha a szállodavezető alkalmazottat töröl:

A szakács kezdőlapja:

A szakács menüket ír ki:



A recepciós kezdőlapja:



A recepciós új vendéget regisztrál: