FEJLESZTÉSI KÉZIKÖNYV

1. Fejlesztés folyamata

Az alkalmazás fejlesztését alapvetően a rendszertervnek megfelelően fogom végezni. Mivel egy meglehetősen szerteágazó feladat lesz a fejlesztés, ezért az alábbi fejlesztési sorrendet fogom követni:

- Alkalmazás alapjainak létrehozása
  - statikus osztályszerkezet létrehozása (mind szerver, mind pedig kliens oldalon), üres függvényekkel, külső fájl beolvasási és fájlba írási képességekkel
  - bemeneti fájl parancsainak segítségével tesztelés, üzenetszekvenciák
- Adatbázissal kapcsolatos feladatok
  - megtervezett adatbázisséma alapján az adatbázis létrehozása, majd tesztadatokkal feltöltése (térképi adatok nélkül)
  - szerver oldalon adatbázis-kapcsolat létrehozása, tesztelése
  - szerver oldali Modify és Select osztályok adatbázist elérő függvényeinek létrehozása
  - adatbázist elérő függvények tesztelése
- Kommunikáció felépítése a két modul között
  - üzenet osztály létrehozása, kommunikáció egyelőre csak osztályok között
  - kliens oldali kontroller osztály függvényeinek létrehozása (üzenetek összeállítása, válaszüzenet feldolgozása)
  - szerver oldali kontroller osztály függvényeinek létrehozása (üzenetek feldolgozása, válaszüzenet összeállítása)
  - alapszintű kommunikáció tesztelése
- Valódi kommunikáció létrehozása
  - kapcsolatkezelés kliens oldalon (kapcsolatfelépítés kérése a szervertől, üzenetküldés a kommunikációs csatornán)
  - kapcsolatkezelés szerver oldalon (több kliens kezelése egyszerre, felhasználók kezelése, üzenetek fogadása, küldése)
  - tesztelés
- Adatbázis felkészítése térképi adatok fogadására
- Kliens oldali felhasználói felület létrehozása
  - megjelenítés és akciókezelés
- Szerver oldali felhasználói felületek (adminisztrátori és hirdetői felület) létrehozása
  - a két tervezett weblap összeállítása
  - funkcionalitás rendszerhez illesztése

2. Fejlesztési környezet

Az alkalmazás fejlesztésének nyelve a rendszertervben leírtaknak megfelelően a Java lesz. A fejlesztés során a JDK (Java Development Kit) 1.6 Update 3 verzióját használom. Konkrét fejlesztői környezetnek a JDeveloper-t választottam, mivel az alkalmazásfejlesztéshez rengeteg hasznos kiegészítő szolgáltatással rendelkezik. A fejlesztés során én ezek közül főleg az adatbázis-tervezéssel kapcsolatos szolgáltatásokat veszem igénybe. A fejlesztést a 10.1.2.3.0 verzióval kezdtem el. Időközben megjelent a 11-es verzió is, ki is próbáltam, de nem találtam annyival hasznosabbnak ezen fejlesztés szempontjából, hogy az új verziót használjam a továbbiakban.
Az alkalmazás alapját jelentő adatbázis egy Oracle Database 10g Release 2 (10.2.0.1.0) verziójú adatbázis. Eredeti terveimben egy 11g adatbázis szerepelt, ez azonban a fejlesztői gép erőforrásainak szűkössége miatt nem kivitelezhető, így döntöttem a 10g mellett. Funkcionalitás tekintetében ez a választás nem befolyásolja az alkalmazás működését.
Az adatbázis eléréséhez az Oracle SQL Developer 1.5.1 (1.5.1.54.40) verzióját használom, ennek segítségével hoztam létre az adatbázist, itt töltöttem fel adatokkal is, valamint SQL utasítások tesztelésére is alkalmas.

3. Kiindulási állapot

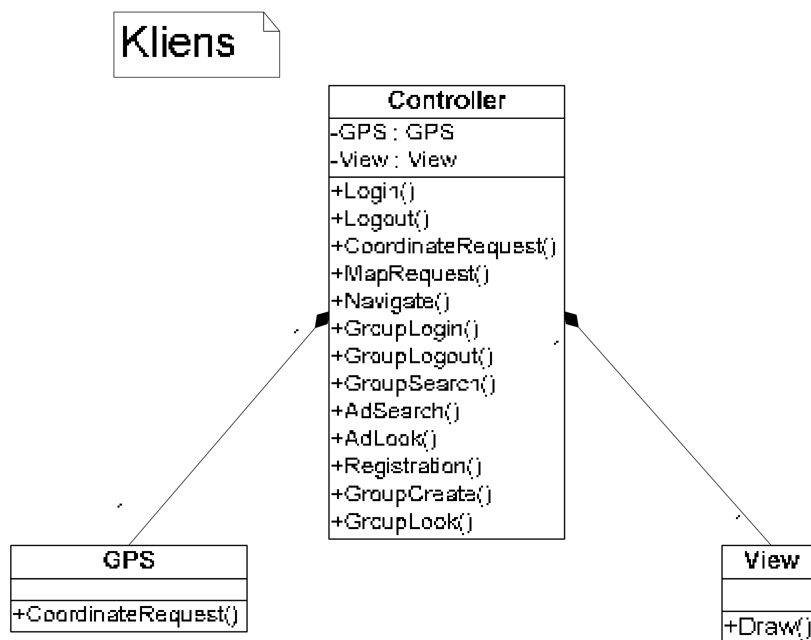
A fejlesztés kiindulási alapjának a rendszerterv tekinthető. Az egész fejlesztés során igyekszem a rendszerterv szerint létrehozni mindent. Természetesen vannak olyan esetek, amire a rendszer tervezésekor egyszerűen nem gondoltam, így ezeket most kell pótolnom. Ezeket az esetek természetesen külön feljegyzem, így egy következő alkalommal már sokkal pontosabban készíthetem el a rendszertervet.
A rendszerterv egyes részeit gyakorlatilag azonnal tudtam alkalmazni a fejlesztésben, ezek az adatbázis modellt, valamint a megvalósítási terv modelljeit és diagramjait jelenti. A terv többi része (például az üzemeltetési terv) majd a fejlesztés későbbi szakaszában kerül felhasználásra.

4. Alkalmazás alapjai

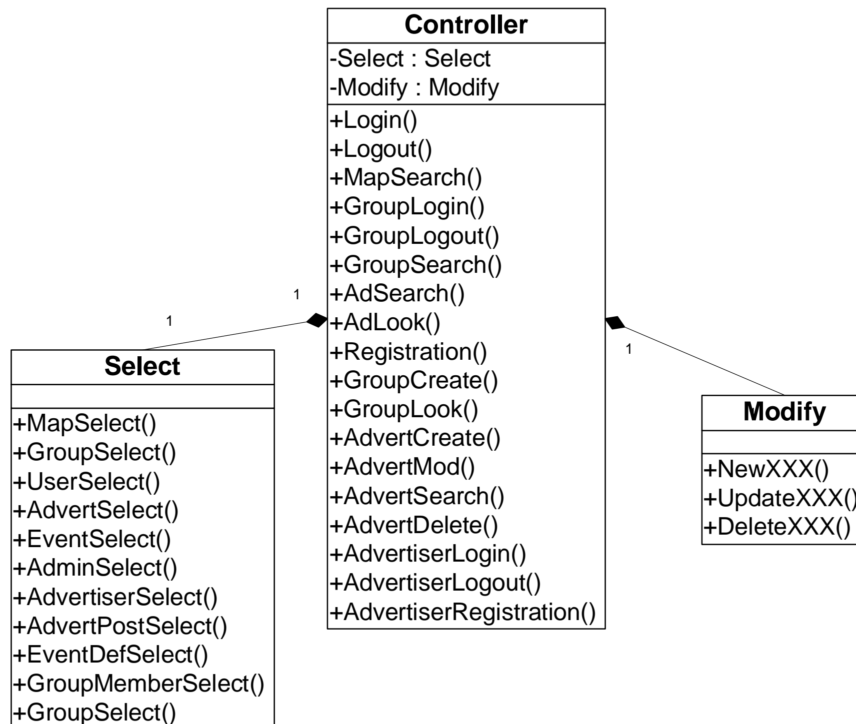
4.1. Statikus szerkezet

A fejlesztés legelső lépése az alkalmazás alapjainak összeállítása volt. Mivel a végleges alkalmazás kliens – szerver architektúrájú lesz, ezért tulajdonképpen két külön alkalmazást kell fejlesztenem, melyek majd előre meghatározott interfészeken kommunikálnak egymással.

Elsőként tehát a rendszerterv megvalósítási tervében megalkotott statikus osztálydiagramból [1., 2. ábra] generáltam Java kódot a JDeveloper segítségével. Minden osztálynak elkészült így egy üres konstruktora és minden függvénye, melyek szintén nem tartalmaznak semmilyen parancsot.



1. ábra: Kliens statikus osztálydiagram



2. ábra: Szerver statikus osztálydiagram

Ezt követően a tesztelhetőség alapját jelentő beolvasási és kiírási funkciókat valósítottam meg, vagyis a program egy bemeneti fájlból olvassa be a parancsokat, majd működése során egy kimeneti fájlba ír minden információt. A bemeneti fájl egy sora tartalmaz egy parancsot, a sor első szava a parancs, míg a paraméterek szóközökkel elválasztva következnek. A fájl feldolgozásakor soronként, szekvenciálisan haladok. A sort a szóközök mentén felbontva az első szó alapján dől el, hogy melyik függvényt szükséges meghívni, a paramétereket pedig változatlanul továbbadom a függvénynek.

A program működésének alapját a függvényhívások egymásutánja adja meg. A függvénytörzsekbe bekerültek a különböző függvényhívások a megvalósítási terv alapján, majd a tesztelhetőség érdekében minden függvényben és konstruktorban kiírtam, hogy éppen milyen függvény hajtódik végre, vagy melyik osztály jött létre.

Mivel ekkor még a két modul között nincsen kapcsolat, ezért ezt mindkét modulon külön-külön végre kellett hajtani és le is kellett tesztelni. A teszteléshez külső parancsfájlokat használtam, a kimenetet szintén külső fájlba írta az alkalmazás a tesztelés során.

4.2. Kliens oldal tesztelése

A bemeneti parancsfájlban tehát minden egyes sor egy parancsot jelent, az első szó a parancs neve, míg utána szóközökkel elválasztva a parancs argumentumai találhatóak. Egy lehetséges parancsfájl:

```
Registration schumy schumy
Login schumy schumy
Map
Navigate
GroupSearch csoport*
Logout schumy
```

A kimenet sorai kétfélék lehetnek. A '##'-kal kezdődő sorok olyan üzenetek, melyek objektum létrejöttét vagy függvényhívást jeleznek. A többi sor az alkalmazás futása során keletkezett üzeneteket jelenti, mint például egy kérésre érkező válasz. A fenti parancsfájlhoz elvárt kimenet:

```
## Object created: MyPackage.Controller_client@3
## Object created: MyPackage.GPS@4
## Object created: MyPackage.View@5
## Controller_client.Registration()
Message send to server
Message received from server
Successful registration
## Controller_client.Login()
Message send to server
Message received from server
Successful login
## Controller_client.MapRequest()
## GPS.CoordinateRequest()
Message send to server
Message received from server
## View.Draw()
Draw map
## Controller_client.Navigate()
## Controller_client.MapRequest()
## GPS.CoordinateRequest()
Message send to server
Message received from server
## View.Draw()
Draw map
## Controller_client.GroupSearch()
Message send to server
Message received from server
Csoportok: csoport1
           csoport2
## Controller_client.Logout()
Message send to server
Message received from server
Successful logout
Exit
```

Ebben az állapotban a kliens oldali függvényhívások nagy része nem okoz további függvényhívásokat, így az üzenetszekvenciák nem bonyolultak. Két esetben jön létre hosszabb szekvencia, a térkép lekérésénél, illetve a térképen navigáláskor. Mindkét esetben a kliens oldali kontroller modul előbb a GPS modulhoz fordul, majd a kapott koordináták alapján térképet kér a szervertől. A visszakapott térképet aztán továbbküldi a megjelenítő modul rajzoló metódusának.

4.3. Szerver oldal tesztelése

A szerver oldalon a tesztelés nagyon hasonló a kliens oldalon tapasztaltakhoz. A parancsfájl szerkezete teljesen megegyezik a kliensnél látottakkal:

```
Registration schumy schumy
Login schumy schumy
Map
GroupSearch csoport*
Logout schumy
```

A kimenet szerkezete is megegyezik, a fenti parancsfájlhoz elvárt kimenet:

```
## Object created: MyPackage.Controller_server@3
## Object created: MyPackage.Modify@4
## Object created: MyPackage.Select@5
## Controller_server.Registration()
## Select.UserSelect()
## Modify.NewUser()
Message sent: Successful registration
## Controller_server.Login()
## Select.UserSelect()
Message sent: Successful login
## Controller_server.MapSearch()
## Select.MapSelect()
Message sent: map
## Controller_server.GroupSearch()
## Select.GroupSelect()
Message sent: csoport1 | csoport2
## Controller_server.Logout()
Successful logout
Exit
```

A szerver oldalon sem bonyolultak a függvényszekvenciák, a kontroller modul szinte minden esetben egy vagy több alkalommal az adatbázishoz fordul a Select és a Modify osztályokon keresztül, majd válaszüzenetet küld a kérésre.

5. Adatbázissal kapcsolatos feladatok

5.1. Adatbázis létrehozása

Az adatbázis létrehozásánál az első feladat az adatbázis-kezelő feltelepítése volt. Eredeti terveim szerint egy Oracle 11g-t használtam volna, és az első próbálkozások ezen is történtek meg. Mivel azonban a fejlesztői gép erőforrásait túlzottan igénybe vette ez a verzió, ezért választottam a 10g Release 2 verziót. Ez sokkal inkább megfelelt a fejlesztői számítógép igényeinek, nagyobb tapasztalatom is volt ennek a verziónak a kezelésében, valamint igen fontos szempontként az alkalmazás által megkívánt funkcionalitást ez a verzió is tudta nyújtani. A telepítés után létrehoztam egy felhasználót az alkalmazás számára, a megfelelő jogosultságokkal ellátva.

Az adatbázis felépítéséhez a rendszerterv adta az alapot. Az ott megtervezett adatbázissémából a JDeveloper segítségével SQL DDL utasítássorozatot generáltam, és SQL Developerben futtattam le az így kapott szkriptet.

A táblák nagy részében az elsődleges kulcs egy ID nevű oszlop. Mivel azt szerettem volna elérni, hogy ezek az azonosítók az egész adatbázisban minden egyes sorra egyediek legyenek, ezért létrehoztam egy közös számlálót. Minden ilyen táblára definiáltam egy triggert, mely új sor beszúrásakor a számláló következő értékét szúrja be azonosítóként. Azokban a táblákban, ahol az elsődleges kulcs több oszlopból áll, ott ezek az oszlopok idegen kulcsok is egyben, így ezek egyedisége is biztosított a triggerek által, viszont ezeknél a tábláknál a kulcsok beszúrását kézzel, programkódból kell elvégezni. Egy példa trigger az `ADMIN_USER` táblába beszúráshoz:

```
create or replace TRIGGER TRG_ADMIN_USER
BEFORE INSERT
ON ADMIN_USER
REFERENCING NEW AS NEW
FOR EACH ROW
BEGIN
SELECT INSERT_SEQ.nextval INTO :NEW.ADMIN_ID FROM dual;
END;
```

Az így összeállított üres adatbázist feltöltöttem tesztadatokkal, miközben vizsgáltam, hogy a beállított triggerrek és a táblákban lévő idegen kulcs kényszerek megfelelően működnek-e. A megtervezett séma egyelőre jónak bizonyult, nem kellett változtatást végrehajtanom rajta.

5.2. Adatbázis elérése az alkalmazásból

Ahhoz, hogy a programkódból el tudjam érni az adatbázist, az `oracle.jdbc.*` nevű osztályt kell a programkódhoz szerkeszteni, ez valósítja meg a program és az adatbázis közötti kapcsolatot biztosító thin drivert. Ez az osztály Oracle 10g-ben az `ORACLE_HOME\jdbc\lib\ojdbc14.jar` fájlban található, ezt az elérési utat meg kell adni a projekt tulajdonság lapján. Ezek után már használhatjuk az Oracle specifikus függvényeket és osztályokat. Egy kapcsolat létrehozásának kódja az alábbi:

```
protected OracleDataSource ods;
protected String url;
protected Connection con;

//-- kapcsolat paramétereinek definiálása
ods = new OracleDataSource();
url = "jdbc:oracle:thin:@//localhost:1521/orcl";
ods.setURL(url);
ods.setUser("ONLAB");
ods.setPassword("******");
//-- kapcsolat megnyitása
con = ods.getConnection();
```

A programrészlet futása után van egy megnyitott kapcsolatunk az adatbázis felé, amit a `con` változó reprezentál, ezentúl elegendő erre hivatkozni, ha az adatbázishoz szeretnénk fordulni. Ezt a kapcsolatot a program futása során nem, csak a futás legvégén zárom be.

SQL utasítások futtatásával érem el minden esetben az adatbázist. Ezek az utasítások két nagy csoportra bonthatók, az első az adatmódosító utasítások, a második pedig a lekérdező utasítások. Szintén megkülönböztethető két csoport aszerint, hogy az utasításnak ad-e a program paramétereket, vagy sem. A következő példában két, a programkódból vett részlettel mutatom be a lekérdezések összeállítását és futtatását.

Az első eset egy paraméterezett módosító SQL utasítást mutat be. Egy sztringbe írhatjuk bele az utasítást, a paramétereket `'?'`-jel jelölve. Ezután létrehozzuk a futtatható paraméterezett utasítást (`PreparedStatement`), majd megadjuk a paraméterek értékét. Erre egy `setXXX` utasításcsalád áll rendelkezésre, ahol megadhatjuk a paraméter típusát (`XXX` a függvény nevében), hogy melyik változót akarjuk megadni (első paraméter, az SQL sztringben balról, 1-gyel kezdve számozódnak a paraméterek) illetve a behelyettesítendő értéket. Nem történik automatikus típuskonverzió, így nekünk kell figyelni, hogy a megfelelő `setXXX` függvényt használjuk. Módosító utasítás futtatásához az `executeUpdate` parancs használható, melynek visszatérési értéke a módosított / törölt / beszúrt sorok száma.

```
try
{
     String sql = "insert into users (name, pwd) values
      (?, ?)";
     PreparedStatement insertStmt =
      con.prepareStatement(sql);
     insertStmt.setString(1, userName);
     insertStmt.setString(2, pwd);
     long rowCount = insertStmt.executeUpdate();
     insertStmt.close();
}
catch (Exception ex) {}
```

A második esetben egy nem paraméteres lekérdező utasítást láthatunk. Itt először hozzuk létre a futtatható utasítást (`Statement`), majd megadjuk az SQL sztringet. A futtatáshoz az `executeQuery` parancs használható, melynek visszatérési értéke egy `ResultSet` típusú változó. Ebben a `next()` függvénnyel léphetünk a következő sorra (visszatérési értéke `true`, ha létezik még nem olvasott sora az eredménynek), a soron belül pedig a `getXXX` függvényekkel választhatjuk ki a számunkra szükséges eredményeket, ahol `XXX` a változó típusát adja meg, a függvény paramétere pedig, hogy hányadik oszlop értéke szükséges (szintén balról, 1-gyel kezdődve számozódik). Paraméterben itt megadható az oszlop neve is, ez azonban némileg lassabb, de kényelmesebb. Ebben az esetben történik automatikus típuskonverzió, azonban ha ez nem lehetséges, akkor `SQLException`-t kapunk.

```
try
{
     Statement selectStmt = con.createStatement();
     sql = "select max(user_id) from users";
     ResultSet rs = selectStmt.executeQuery(sql);
     if (rs.next()) {i = rs.getLong(1); }
     selectStmt.close();
}
catch (Exception ex) {}
```

5.3. Módosító utasítások

Az utasítások két csoportja (lekérdező és módosító) alapján az alkalmazásban is két osztályban valósítom meg az SQL utasításokat. A lekérdező típusú utasítások a Select, míg a módosító jellegű utasítások a Modify osztályba kerültek.

Módosító utasítások esetén minden lehetséges SQL parancsot megvalósítottam, vagyis az adatbázis minden táblájához elkészítettem egy beszúrás, törlés és módosítás függvényt. Ez

alól csak az eseményeket tároló tábla kivétel, mivel ebben a táblába csak beszúrni lehet, törölni és módosítani nem, így a naplózás során egyetlen esemény sem veszhet el.
Beszúró függvényeknél (`NewXXX` függvények) minden paramétert ki kell tölteni a paraméterlistában, szám típusú mezők esetén a `0`, szöveg típusú mezőknél a `""` jelenti a `null` érték beszúrását. Természetesen azonosítót nem kell megadni a triggerek miatt, dátum típusú értékek esetén pedig az alkalmazás természete miatt mindig az aktuális rendszeridő szúródik be. Visszatérési érték minden esetben a beszúrt sor azonosítója lesz, -1 esetén a beszúrás nem volt sikeres.
Törlés (`DeleteXXX`) esetén csak az elsődleges kulcsot jelentő oszlop vagy oszlopok értékét kell megadni. Visszatérési értéket itt csak hibajelzésre használok, -1 jelenti a futás közbeni hibát.
Módosítás esetén (`UpdateXXX`) szintén minden paramétert meg kell adni, azonosítóval együtt. Az azonosító alapján keresi ki a módosítandó sort, a paraméterlistában kitöltött adatokat módosítja, a beszúrásnál ismertetett `null` értékekkel jelzett attribútumokat nem módosítja a függvény. Visszatérési érték a módosított sor azonosítója, a -1 itt is hibát jelez.

5.4. Lekérdező utasítások

Lekérdező utasítások esetén is minden lehetséges táblára összeállítottam egy általános lekérdezést, melyet tetszőlegesen lehet paraméterezni. Azokban a táblákban, ahol az elsődleges kulcs egy azonosítót tartalmazó oszlop, a keresés történhet erre az azonosítóra, vagy pedig az összes többi oszlopra egyszerre. Amelyik oszlopot nem kívánunk tesztelni, ott a módosító utasításokhoz hasonlóan a `0` vagy `""` értékkel jelezhetjük ezt. Lekérdezések esetén kezelni kell a dátumok bevitelét is, mivel természetesen dátum típusú attribútumokra is lehet keresni. Minden ilyen esetben két karaktersorozatot várnak a függvények, az egyik a keresés alsó határa, a másik a felső határ. Amennyiben egyezésre keresünk, úgy mindkét határnak a keresendő értéket állíthatjuk be. A karaktersorozat formátuma minden esetben `'éééé-hh-nn őő:pp:mm'` kell, hogy legyen.
Általános lekérdezéseken kívül kettő további lekérdezést is írtam. Az egyik a felhasználók bejelentkezésénél a név – jelszó párosra keresést segíti, míg hirdetések esetén az egy hirdetőhöz kapcsolódó hirdetések keresésénél egyszerűbb volt egy új keresést létrehozni, mint a meglévőkből építkezni. Ez az eset is rávilágít arra, hogy nem feltétlenül hasznosabb általános lekérdezések használata, mint néhány, speciálisabb feladatra létrehozott lekérdezésé.
Minden lekérdezés egy eredményhalmazzal (`ResultSet` típus Java-ban) tér vissza. Ezt az eredményhalmazt a kontroller osztálynak fel kell dolgoznia, majd ezek után továbbküldeni a kérés feladója felé. Hiba esetén a visszatérési érték `null`.

5.5. Adatbázis elérés tesztelése

Az adatbázist elérő függvények teszteléséhez nem fejlesztettem kiegészítő részeket az alkalmazáshoz. Az egyes függvényeket a szerver oldali kontroller osztályban hívom meg, a kimenet pedig a szokásos fájlba íródik. Kétféle visszatérési értékkel kellett dolgoznom, mivel a módosító függvények `long` típussal, míg a lekérdező függvények egy eredményhalmazzal térnek vissza. Mindkét esetre egy-egy általános `Print()` függvényt készítettem, mellyel az eredmény kiírása megoldható. Az első esetben egyszerűen kiíródik az eredményül kapott szám, illetve hiba esetén a hibaüzenet. Eredményhalmaz esetén először a metaadatokat kérdezem le, innen meghatározhatóak az oszlopnevek és a sorok száma. Ezután az eredményhalmazon soronként lépek végig, és írom ki az értékeket. Eredményhalmaz kiírására egy példa:

```
Event:

ID   EVENT_CODE USER_ID    DATETIME
------------------------------------------------
158  4          154        2008. 11. 20. 15:22:05
160  4          154        2008. 11. 20. 15:22:05
155  1          154        2008. 11. 20. 15:22:07
156  2          154        2008. 11. 20. 15:22:07
161  5          154        2008. 11. 20. 15:22:08
162  6          154        2008. 11. 20. 15:22:12
163  3          154        2008. 11. 20. 15:22:43
```

Az adatbázist elérő függvények mindegyikében alkalmaztam kivételkezelést, ezt egyébként a fejlesztőkörnyezet kötelezővé teszi, fordítási hibát jelez a hiánya esetén. Viszont hibát nem csak a beépített függvények okozhatnak, hanem futás közben a feldolgozott adatok minősége vagy felhasználói hibák miatt is bekövetkezhetnek. Ezeket az esetek is kiszűröm, új kivétel létrehozásával és eldobásával kezelem a hibát.

A függvények visszatérési értékébe nem fér bele a hiba leírása, csak a hiba jelzése (-1 érték szám esetén, `null` érték eredményhalmaz esetén). Így a hiba leírását a két modul egy-egy modul szinten globális változójába mentem, és ha feldolgozás közben hibajelzést észlel az alkalmazás, innen olvassa ki az utolsó hibaüzenetet.

Kimerítő tesztelést végeztem, azaz minden függvényt minden lehetséges paraméterkombinációval (természetesen kombináció alatt azt értem, hogy adunk-e értéket egy paraméternek vagy sem) kipróbáltam. A legtöbb probléma elgépelésből származott, például sokszor okozott gondot listák indexelése vagy az SQL utasítás megfelelő paraméterének kiválasztása. Valódi hibát fedeztem fel viszont a törléseknél több helyen is. Ha olyan sort szeretnék törölni az adatbázisból, amire más táblában hivatkozok, akkor

először a hivatkozó sort kell kitörölni, és csak utána lehetséges a hivatkozott sor eltávolítása. Ezt a sorrendet néhány esetben nem tartottam be, de a tapasztalt hibaüzenetből kiindulva megoldható volt a probléma.

5.6. Naplózás

A szerver oldalon minden egyes akcióról bejegyzést kell készíteni a napló számára. Ehhez egy módosító függvényt, a `NewEvent()`-et használom. A paraméterek között megadható egy eseménykód, mellyel az elmentendő esemény típusát jellemezhetjük, a többi paraméter az eseményben részt vevő objektumok azonosítóit adja meg. A naplóbejegyzéshez a dátum minden esetben az aktuális rendszeridő, így ezt nem kell manuálisan megadni.
A naplózó függvényt minden kontrollerbeli függvényhívásban használom, méghozzá a függvényhívás legelején, így nem fordulhat elő, hogy egy esetleges hiba esetén nem kerül a naplózó függvényre a végrehajtás. Így valóban minden eseményről készül bejegyzés a naplóba.

6. Kommunikációs alapok

6.1. Üzenet osztály

A szerver és a kliens oldal, illetve a szerver és az ahhoz kapcsolódó adminisztrátori és hirdetői oldalak közötti kommunikáció megvalósításához létrehoztam egy üzenet osztályt. Az elküldendő üzenet minden esetben egy sztring, melyben az egyes paraméterek a `http` kérések paramétereihez hasonlóan jelennek meg. Egy minta üzenet-sztring:

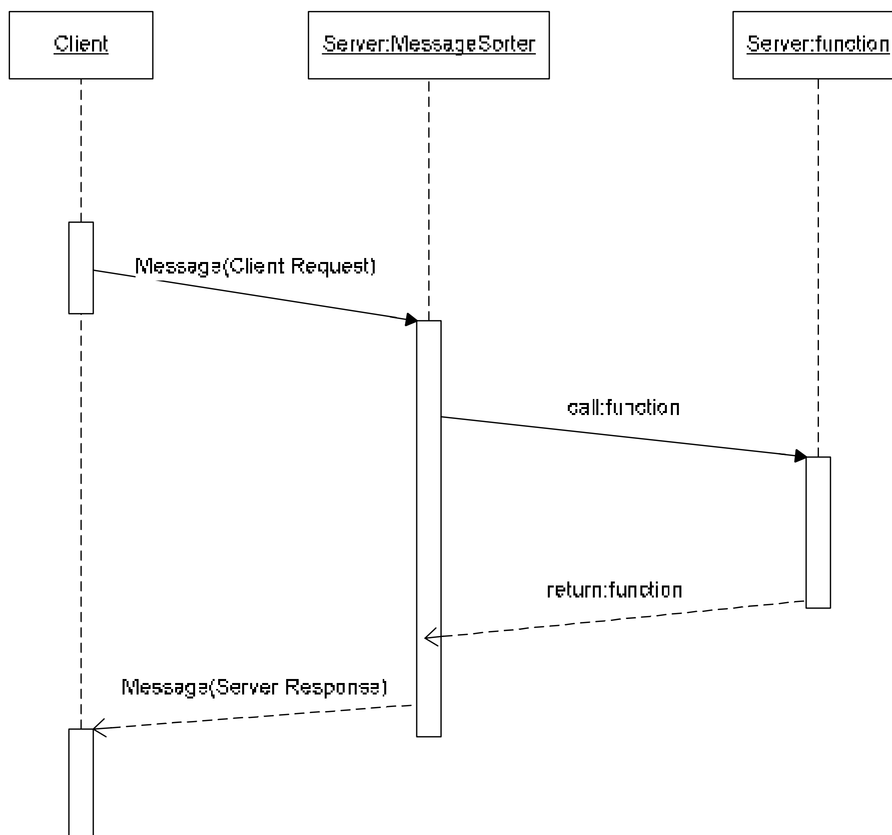
```
Response=AdvertiserLogin&Result=OK&AdvertiserId=1489721
```

Minden kérés első paraméterének neve `Request`, értéke pedig hogy annak a funkciónak a neve, amit el kívánunk érni. A további paraméterek a kérés típusától függenek. Válaszokban az első paraméter neve `Response`, értéke pedig kötelezően ugyanaz, mint a kérésben a `Request` paraméter értéke. Minden válasz tartalmaz egy `Result` paramétert, ami azt adja meg, hogy a kérés sikeresen végrehajtódott-e vagy sem. Ha egyszerű a válasz, mint például a mintában egy azonosítót küldünk csak vissza, akkor ezt további paraméterekben tehetjük meg. Összetettebb esetekben, mint például egy eredményhalmaz visszaküldésekor kötelező paraméter a `Count`, ami az őt követő paraméterek számát adja meg. Ilyenkor a további paramétereket 0-tól kezdve sorszámozom, így kizárom annak lehetőségét, hogy két azonos nevű paraméter legyen egy üzeneten belül.

Az üzenet osztályban kétféleképpen is tárolom az üzenet tartalmát. Egyrészt egy privát sztringként, amit egy-egy metóduson keresztül lehet írni és olvasni. Másrészt egy `Hashtable` adatszerkezetben, amelynek lényege, hogy kulcs – érték adatpárokat lehet benne tárolni, ezért kifejezetten alkalmas az üzenet paramétereinek tárolására. Ez a tárolási forma csak olvasható, mivel az írása bonyolult és felesleges is. A kulcs minden esetben a paraméter neve, míg az érték a paraméter értéke. Kiolvasásra főleg ez utóbbi tárolási formát használom, mivel közvetlenül és az adatszerkezetből kifolyólag gyorsan elérhető bármelyik paraméter értéke a paraméter nevével hivatkozva.

6.2. Kommunikáció

A modulok közötti kommunikáció során a szerver mindig vár bejövő üzenetekre, kérés nélkül sosem küld üzenetet. Kérés érkezhet a klienstől, az adminisztrátori és a hirdetői oldalról.
A kérés küldője a kapott utasítások alapján felparaméterez egy üzenetet. A kérést jelentő üzenet összeállítása után a szerver kontroller moduljának üzenetosztályozójához kerül, ami a kérés alapján eldönti, hogy melyik függvénynek továbbítja azt. Itt megtörténik az üzenet paramétereinek kiolvasása, majd feldolgozása. Az eredmény itt is egy üzenetbe ágyazódik, majd visszakerül az üzenetosztályozóhoz, ami visszaküldi a kérést küldőnek. A küldő feladata az eredmény feldolgozása, kiírása, esetleges további akciók végrehajtása [3. ábra].



3. ábra: Üzenetszekvencia kliens kérés esetén

6.3. Tesztelés

A kommunikáció és az egész eddig megvalósult rendszer tesztelésére kétféle módszert alkalmaztam.

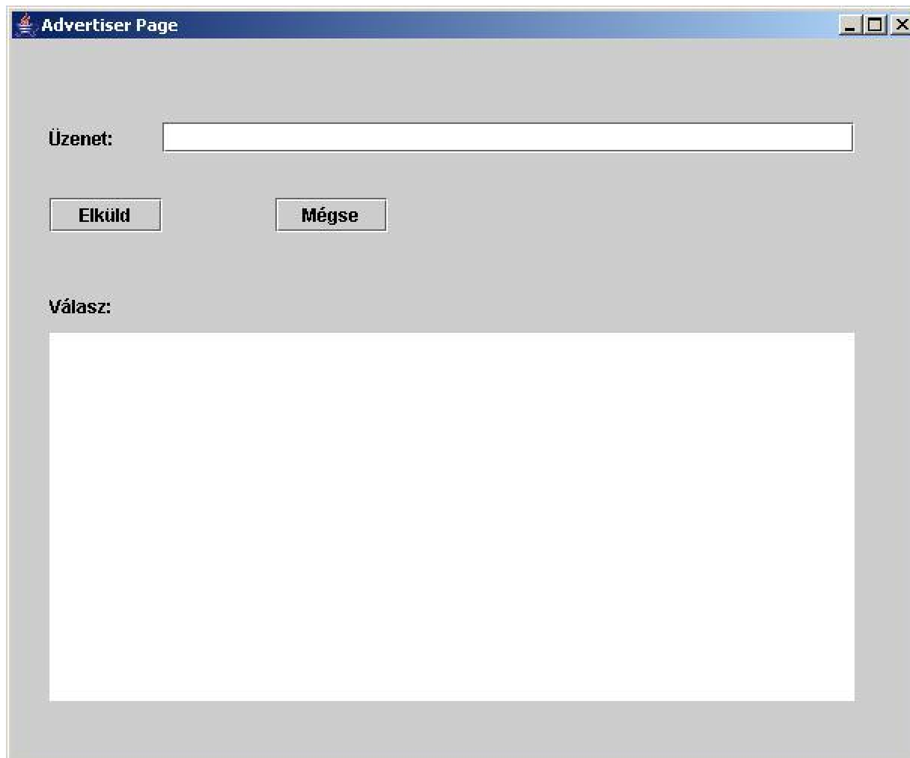
A kliens oldalon a már megismert bemeneti parancsfájl segítségével lehet irányítani a futást, a kimenet pedig szintén fájlba íródik. A parancsfájl nagyban hasonlít a 4.2. fejezetben bemutatottra, a kimenetnek azonban sokkal nagyobb a jelentősége, mivel most már valós adatok jelennek meg benne. A bemeneti tesztfájl, amivel az eddig elkészült funkcionalitást teszteltem:

```
Registration tamas tamas
Login tamas tamas
GroupCreate 154 csoport4
GroupCreate 154 csoport5
GroupSearch 154 csop*
GroupLogin 154 csoport2
GroupLogout 154 csoport2
Logout 154
```

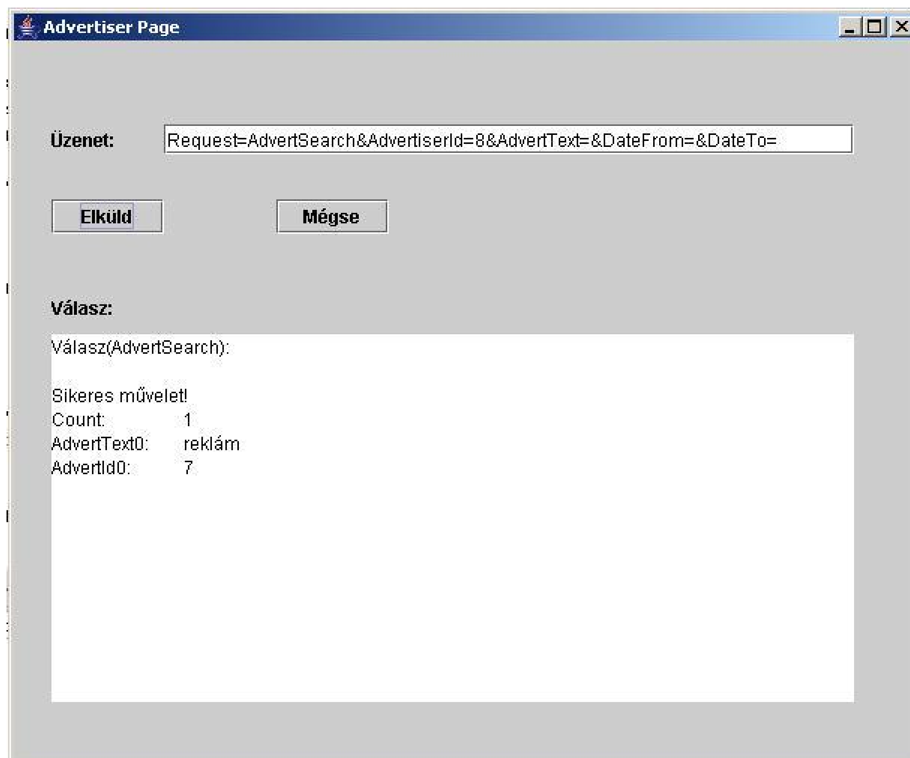
Ebben a tesztfájlban azt modellezem, ahogy egy felhasználó regisztrál, majd bejelentkezik a rendszerbe. Létrehoz két csoportot, végrehajt egy keresést a csoportok között. Belép egy csoportba, majd kilép onnan, végül kilép a rendszerből is. Az első két utasítás kivételével mindenütt az első paraméter a felhasználó azonosítója, amit a bejelentkezés után kap meg.
Az elvárt kimenet:

```
Sikeres regisztráció
Sikeres belépés
User id: 154
Csoport létrehozva
Csoport létrehozva
Csoportok:
     ID   NÉV
     ---------------------
     4    csoport1
     110  csoport2
     111  csoport3
     157  csoport4
     159  csoport5
Sikeres csoportjelentkezés
Sikeres csoportlejelentkezés
Sikeres kijelentkezés
```

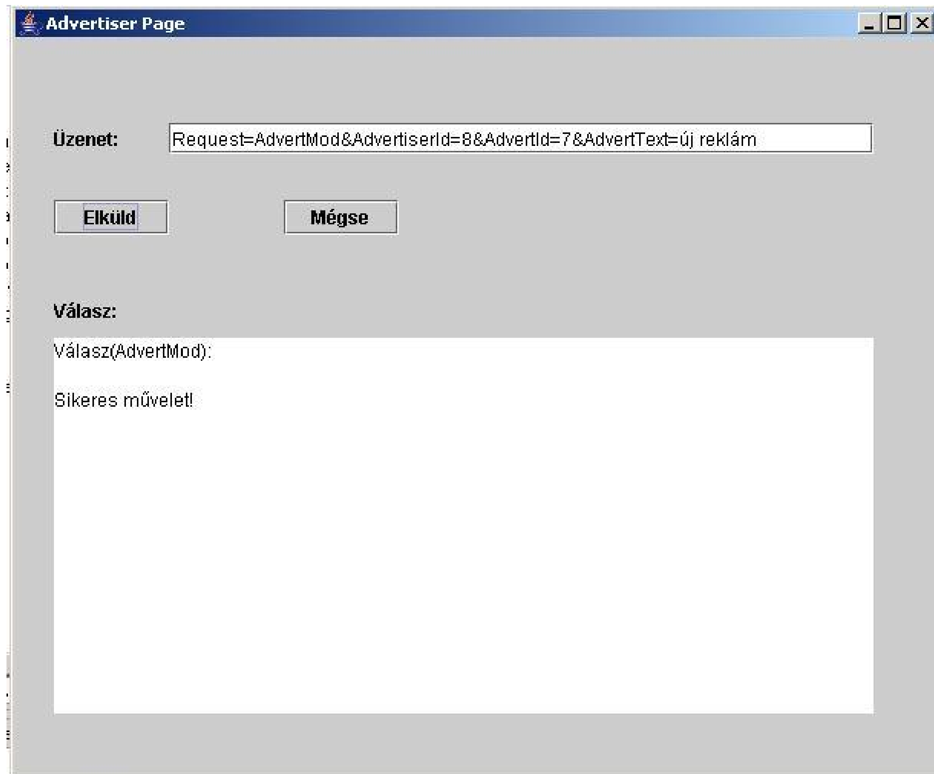
A hirdetői funkcionalitás tesztelésére egy egyszerű ablak szolgál [4. ábra], melyben egy szövegmezőben adható meg az elküldendő üzenet, pontosan a szerver által elvárt formátumban. Az üzenet elküldése és a válasz fogadása után az eredmény egy másik szövegmezőben jelenik meg, formázva [5., 6. ábra].



4. ábra: Hirdetői tesztoldal



5. ábra: Hirdetői tesztoldal, eredményhalmaz megjelenítése



6. ábra: Hirdetői tesztoldal, egyszerű válasz megjelenítése

A tesztelés során nem csak a kimeneti fájlt, hanem a napló tartalmát is figyeltem, mivel így tudtam ellenőrizni, hogy megfelelően működik-e a logolás. A tesztelés során hibát okozó futtatások nagy többségénél elgépelés okozta a hibát. Ezek mellett származott hiba többek között rossz utasítássorrendből, az kapcsolatleíró és a kurzor nem megfelelő kezeléséből. Minden esetben a hibaüzenetek kellő támpontot adtak a hiba javításához.