ÉLETMÓDNAPLÓ-ADATBÁZIS MIGRÁLÁSÁT TÁMOGATÓ ALKALMAZÁS TERVEZÉSE ÉS MEGVALÓSÍTÁSA

Szakdolgozat beszámoló

Írta: Lénárt Bálint – JF0U1Y

Témavezető: dr. Vassányi István

Pannon Egyetem

Villamosmérnöki- és Információs Rendszerek Tanszék

Tartalom

1. Bevezetés

Napjainkban rendkívül fontos a tudatos táplálkozás. Az egészségtelen életmód nem csak betegségekhez, hanem sokszor halálhoz is vezet. Magyarország egyik vezető halálozási oka a keringési betegségek, melyek sok esetben a helytelen életmódra és táplálkozásra vezetethetők vissza. Minden olyan informatikai rendszer hatásos lehet ezek ellen, amit nyilvántartja a szervezetünkbe bevitt táplálékokat és azok összes adatát, valamint a testünkkel kapcsolatos fontosabb adatokat (pl. súly, betegségek, bevett gyógyszerek, …).

A Pannon Egyetem Villamosmérnöki- és Informatikai Rendszerek karán már hosszabb ideje folyik egy táplálkozás tanácsadó szakértő rendszer fejlesztése, a Lavinia Életmód-tükör. A rendszer még csak teszt üzemmódban működik, de már most is számos sikert ért el klinikai kísérletek terén. Az Android operációs rendszeren futó alkalmazást bárki letöltheti és használhatja.

http://www.lavinia.hu

Jelenleg a Lavinia kliens második verzióját fejlesztik a tanszéken. Ennek a rendszernek a felhasznált technológiái eltérnek a most használható verziótól. Egyik legfontosabb eltérése az adatbázis használata. Dolgozatom témája a régi adatbázis adatainak vizsgálata, valamint az új adatbázis-kezelő rendszerbe való migrálása. Ennek fő indoka, hogy a régi klienst használó felhasználók ne vegyék észre az átállást, valamint az összes tárolt adatukat vissza nézhessék az új kliensből.

A feladat nem egyszerű, hiszen egy dinamikus sémájú adatbázisból kell az adatokat kiolvasnom, melyek adatszerkezete dokumentumról-dokumentumra változhat, valamint egy olyan adatbázisba elhelyezni az adatokat, aminek kötött szerkezete van sok-sok kényszerrel és idegen kulcsokkal.

2. Specifikáció

A fentebb ismertetett Lavinia kliens MongoDB adatbázist használ a felhasználók és azok bejegyzéseik tárolására. A jelenleg készülő új kliens már egy másik adatbázis szervert fog használni, a PostgreSQL-t.

El kell érni tehát, hogy az átállásig való regisztrációk és bejegyzések átkerüljenek az egyik adatbázisból a másikba. Ezeknek az adatoknak a migrálása a dolgozatom témája. A következőkben ismertetett funkciókat szeretném a végleges rendszerbe implementálni.

A könnyen átláthatóság miatt a felhasználót tájékoztatni kell a műveletek eredményeiről. Ezt grafikus felületen kényelmesen lehet megoldani. Ez mind a felhasználónak, mind a rendszernek egy könnyítés az információk átadása szempontjából. A felületen a felhasználó még további beállításokat is végezhet egyszerűen (például beállítások, adatok megtekintése, migrálni kívánt felhasználók kiválasztása).

Gondolni kell arra, hogy egy ilyen migrálási folyamatot bármikor és bármennyiszer elindíthatunk. A második alkalomtól már figyelni kell azokra az adatokra, amik már az új adatbázisban is megtalálható. Ilyenkor a hatékonyság és az adatok védelme miatt célszerűbb a szoftverben megvizsgálni, hogy mely adatok estek már át migráción. Másik megoldás az lenne, hogy a migráció előtt az új adatbázis minden tábláját ürítjük. Ez egyben veszélyes és hosszadalmas folyamat lenne.

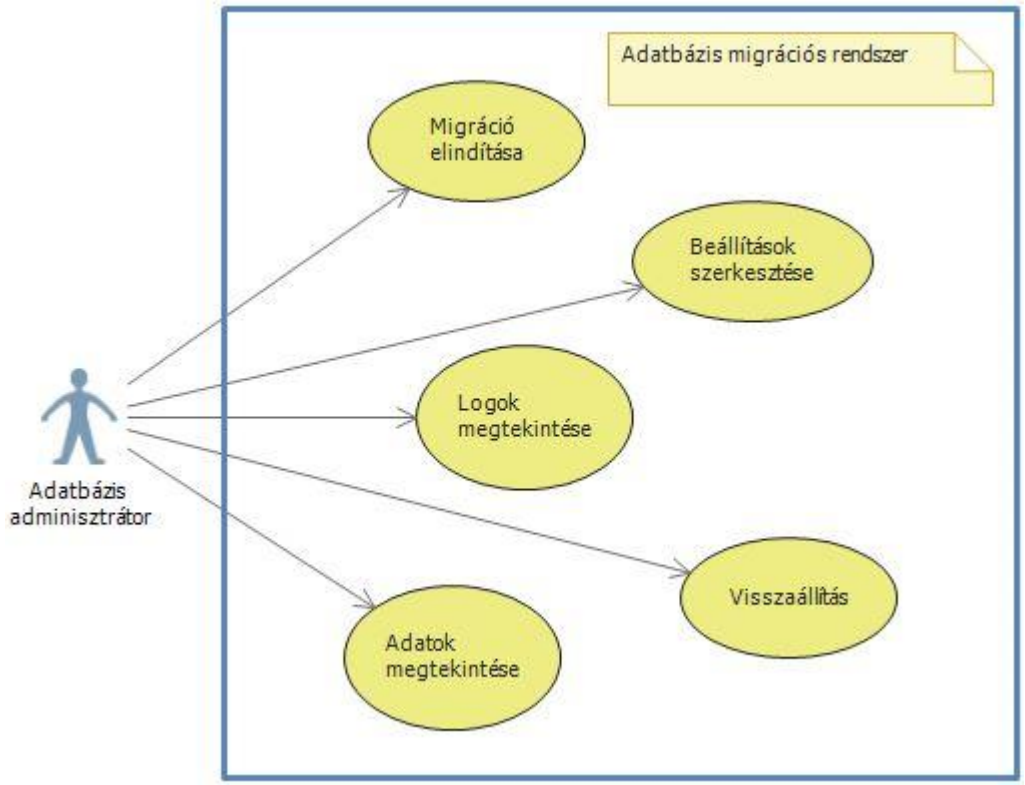
A Lavinia alkalmazás az egészségre vonatkozó indexeket és értékeket a felhasználó által bevitt bejegyzésekből számítja. E miatt rendkívül fontos, hogy egy felhasználóhoz minden ilyen bejegyzést migrálni kell a későbbi helyes működéshez. Ezt nevezhetjük felhasználó szintű tranzakciónak is, mely lényege, hogy egy felhasználóhoz tartozó minden adatot átviszünk a másik adatbázisba vagy egyiket se.

Amennyiben a szoftverben párhuzamosítás lesz alkalmazva, úgy meg kell fontolni a tranzakciók kezelését az adatbázis oldalon. Ezt legkönnyebben tárolt eljárásokkal lehet megoldani. Ezek a függvények végezzék el mindent, ami az adatbázishoz tartozik. Ezzel egyszerűsödik az adatbázis kezelése a szoftverben is.

A tranzakciók nagyobb biztonságot fognak jelenti, valamint segítségükkel ellenőrizni lehet az adatbázisban lévő hivatkozások helyességét. Hiba esetén az elhárítást lehet a tárolt eljárásban vagy a szoftverben egyaránt.

Elvárható, hogy minden migrációs folyamatról a szoftver valamilyen jelentést, log-ot készítsen szöveges formátumban. Ebben minden sikeres és sikertelen folyamatot fel lehet tüntetni a későbbi hibakeresés szempontjából. Legegyszerűbben egy sima szöveges fájlba lehet menteni ANSI vagy UTF-8 karakterkódolással. A felépítése a lehető legegyszerűbbnek kell lenni a könnyen olvashatóság és keresés szempontjából.

A felhasználó által elvárt funkciók – amik használni tud a felületen keresztül el tud érni – az alábbi használati eset tartalmazza:



A rendszer nem tartalmaz más aktorokat, hiszen a szoftver közvetlen kapcsolatot biztosít az adatbázisokkal.

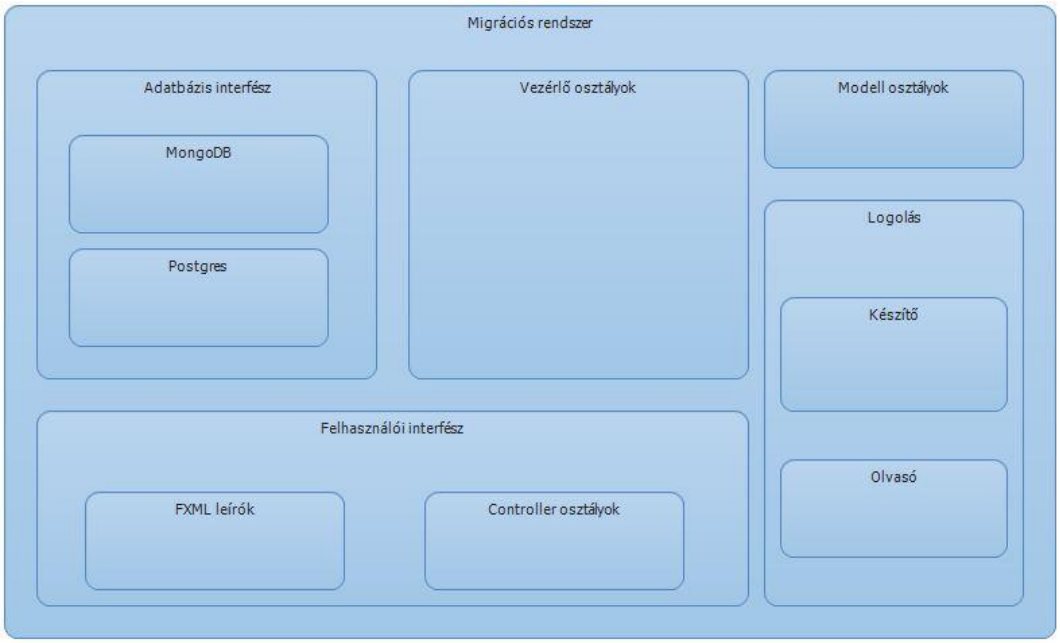
3. Rendszerterv

Az implementáció előtt rendkívül fontos a rendszer alapos megtervezése. A dokumentáció e részében kitérek a szoftver komponenseire, valamint az adatbázis oldali terveimre is.

3.1. Felhasználói program terve

A felhasználói programot több komponens együttes használatával szeretném megvalósítani. Ez azt jelenti, hogy a specifikációban ismertetett funkciókat külön modulként implementálom úgy, hogy ezek integrálása könnyed legyen.

A szoftver nagyobb komponenseit az alábbi ábra szemlélteti:



Az alábbi szerkezetet könnyű implementálni Java nyelven, hiszen az azonos komponensekbe tartozó osztályokat egy könyvtárban (Java nyelvben package-be) lehet szervezni. Az alábbi bekezdésekben az egyes komponenseket ismertetem meg.

Az adatbázis interfész az egyik legtöbbet használt osztályokat tartalmazza. Látható, hogy ez az egység két részre van bontva. Az egyik mindig a forrás, a másik mindig a cél adatbázis kommunikációjával foglalkozó osztályok. Ezen osztályok megtervezése az egyik legnehezebb feladat, mivel itt halad keresztül az összes adat.

A felhasználói interfész biztosítja a kapcsolatot a szoftver folyamataival és a felhasználó között. Az egyik legmodernebb és szerintem legkönnyebben kezelhető grafikus felhasználói technikát alkalmaztam, ami a JavaFX. Alternatíva között felmerülhet a Swing is, de ez már kezd elavult technikának számítani. A JavaFX technológia a Java 8-as verziója óta alapértelmezetten az alap környezet stabil részét képezi. Ennek bővebb ismertetése megtalálható a dokumentum későbbi fejezetében.

Egy migrációs folyamat során nagyon fontos a naplózás és ezek mentése valamilyen olvasható formátumban. Segítségével könnyen át lehet nézni később is egy adott folyamat során keletkezett üzeneteket, hibákat. Implementálni egy keretrendszer segítségével lehet legkönnyebben. Ilyenek például a Logging API, Log4j, SLF4J. Rengeteg beállításokkal rendelkeznek, ami megkönnyítik a munkát. Támogatni kell az elkészült naplók olvasását is, például egy olvasóval, ami a rendszer része.

A modell osztályok reprezentálják az adatbázisban lévő adatokat. Használatuk többek között azért is segíti a munkát, mert az egymáshoz tartozó adatokat egy egységbe zárjuk. A felhasználói felületnek a vezérlő utasításai is megkövetelik az ilyen osztályok használatát. Az itt található osztályok másik része egy saját adatszerkezet is, amit több alap adatstruktúrából állítottam össze. Ezek használatával csökkentem az adatbázis lekérdezések számát, valamint az egy táblához tartozó többszöri módosítást. Első felmérések azt mutatják, hogy ezek használatával jelentős másodperceket lehet megspórolni.

Az előbb bemutatott komponenseket a vezérlő osztályok fogják egymáshoz kötni. Az adatok, beavatkozások, események egy része ezeken az osztályokon fog egyik komponensből a másodikba haladni.

3.2. Adatbázis oldali tervek

A MongoDB adatbázison kevés módosítást kell végrehajtani. A lekérdezések pontos megtervezése után akadtak olyanok, ami elég hosszú ideig futottak vagy kerestek. Ennek elkerülése érdekében megfelelő indexeket helyeztem a nagyobb kollekciókra.

A Postgres adatbázisnál már ennél szigorúbb ellenőrzések kellenek, hiszen a kapott adatokat ezentúl ez a rendszer fogja tárolni.

Több szempontból is nagyon fontos a tranzakciók bevezetése és azok helyes kezelése. Egyik elvárás, hogy egy felhasználó adatait vagy teljes mértékben rögzíteni tudjuk vagy hagyjuk ki a migráció során. Ezt mind az adatbázis oldalon, mind a szoftver oldalon meg lehet tenni tárolt eljárásokban. Én az adatbázis objektumokat választottam a következő bekezdésben leírtak miatt.

A ma használatos szinte összes adatbázis-kezelő rendszer nyújtja a tárolt eljárások készítésének funkcióját. Ez egy függvény, ami az adatbázis szerveren fut, szintén a szerver fordítja ezeket le, így gyorsabb működést eredményez, mint egy lekérdezés. Ezen függvények használata kevesebb adatforgalmat bonyolít le a szerver és a kliens között. A tranzakciók használata is sokkal könnyebb a függvények segítségével. Számos előnye mellett meg kell említeni, hogy a függvények használata egy interfészt biztosít az adatbázis és a kliensek között, azaz a most megírt tárolt eljárásokat más kliens is fel fogja majd tudni használni.

4. Technológiák

A feladat központi elemei a két adatbázis, így először azokat ismertetném, majd áttérek a programozási nyelvre és egyéb kiegészítőkre. A két adatbázis-kezelő rendszer adottak, így ezeken a feladat megoldása során nem változtathatok.

4.1. MongoDB

A MongoDB adatbázis szoftver egyike az úgynevezett NoSQL adatbázis-kezelő rendszerek családjának, sőt ezek közül a legnépszerűbb.

http://www.mongodb.com/

A NoSQL – azaz Not Only SQL vagy más értelmezésben No SQL – adatbázis-kezelő rendszerek nem táblákban és relációkban tárolják az adatokat mint a relációs adatbázis-kezelő szoftverek (RDBMS), hanem általában valamilyen dokumentum formájában. Ez általában JSON formátumban történik meg. Ezen rendszerek sokkal kevesebb funkcionalitást nyújtanak, mint RDBMS társaik. Kompenzálva ezt ezek az adatbázisok erősen az olvasás és írási műveletekre vannak optimalizálva, valamint sokkal gyorsabbak és skálázhatóak.

A MongoDB egy nyílt forráskódú, platformfüggetlen NoSQL adatbázis-kezelő rendszer, amit 2007 óta fejleszt a 10gen vállalat. A dokumentumokat bináris JSON szerű formátumban tárolja, amit BSON-nak nevezünk. Az alábbi kódban egy JSON dokumentumra láthatunk példát:

```
{
id : ObjectId(”1”),
      name : ”Teszt Elek”,
      address : {
            city : ”Siófok”,
            zipcode : 8600
      },
      languages : [”Java”, ”C”, ”C++”]
}
```

A MongoDB a legelterjedtebb NoSQL adatbázis. Ezt jól mutatja, hogy az alábbi világméretű cégek is használják: Adobe, eBay, Foursquare, LinkedIn, McAfee, SAP PaaS, ….

Több számos grafikus felület létezik az adatbázis eléréséhez. Az egyik legelterjedtebb és leggyorsabban fejlődő program, amit én is használok a Robomongo.

http://www.robomongo.com

4.2. PostgreSQL

A PostgreSQL vagy más néven Postgre egy relációs adatbázis-kezelő rendszer, amit a kaliforniai Berkeley Egyetem kezdett fejleszteni. Mára egy nyílt forráskódú szoftverré nőtte ki magát, amit a postgresql.org közösség koordinál a fejlesztésben.

http://www.postgresql.org/

A relációs adatbázis-kezelő rendszerek – azaz az RDBMS-ek – logikai adatbázisát kizárólag relációs adatmodellek elvén építik fel. Ez a matematika halmazműveletein, legfőképpen a Descartes-szorzat fogalmán alapul. Az adatokat SQL (Structured Query Language) utasítások révén érhetjük el. Ez egy szabvány programozási nyelv, amit alapszinten a legtöbb RDBMS támogat. Sajnos sok esetben a gyártók eltérnek a szabványtól, így eltérések mindig adódhatnak.

A PostreSQL elterjedésének egyik oka, hogy ingyenes, valamint nyílt forráskódja mellett sok olyan funkciót tartalmaz, amiket más gyártók ilyen rendszereinél drága költség mellett lehet csak beszerezni. Erre a Postgres fejlesztői

is felfigyeltek és létrehozták az EnterpriseDB-t, amit nagyvállalati célokhoz ajánlanak.

Az adatok és az adatbázis szerkezetének tanulmányozásához és szerkesztéséhez a pgAdmin és a JetBrain DataGrip programokat választottam.

4.3. Java

A kitűzött feladatomat Java nyelven implementálom. A két adatbázis közé kell egy szoftver, ami menedzseli az adatátvitelt, valamint a felhasználónak erről tájékoztatást ad.

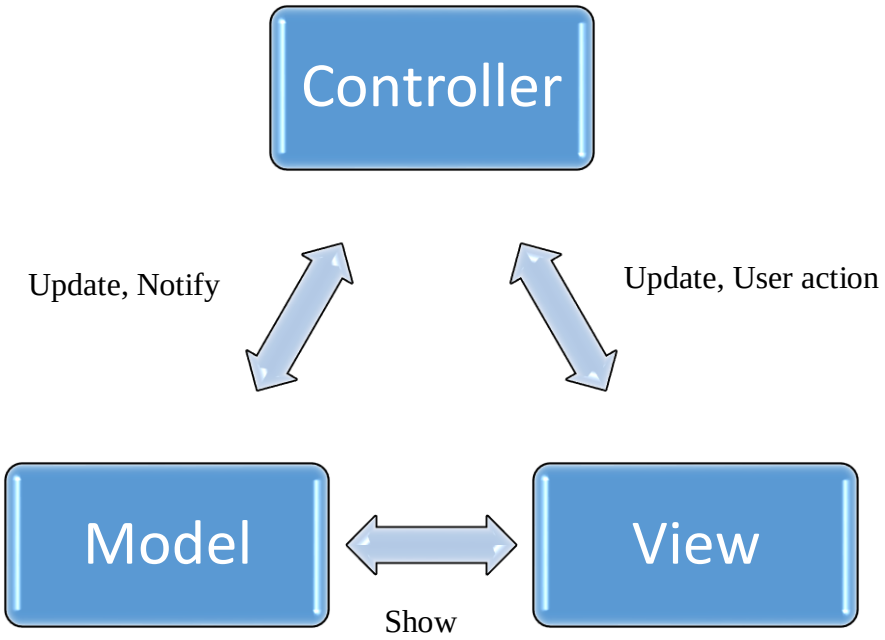
A Java egy általános célú, objektumorientált nyelv, amit a Sun Microsystem cégnél kezdek fejleszteni, de 2009-től az Oracle gondozásában áll. A Java alkalmazásokat bájtkód formátumra fordulnak, amit a Java Virtual Machive (JVM) futtat és alakít át gépi kódra. Ennél fogva valamivel lassabb működést tesz lehetővé, mint azok a programozási nyelven írt szoftverek, amik azonnal gépi kódra fordítanak.

A program fejleszéséhez a JetBrains IDEA IntelliJ fejlesztő környezetet választottam. A program futtatásához Java 8-as környezet szükséges.

Azért a Java nyelvet választottam, mert az áll hozzám a legközelebb, valamint a specifikációban ismertetett funkciókat könnyű vele implementálni. Erős objektumorientált megkötései vannak, ami segít az emberi gondolkodásban, valamint az adatbázisban tárolt entitásokból könnyen lehet objektumokat felépíteni. Gyorsasága nem éri el a hardware közelebbi nyelveket, ugyanakkor ennél a feladatnál nem jelentkezik szignifikáns különbség.

4.3.1. JavaFX

A grafikus felület létrehozása elég egyszerű a JavaFX technológiának köszönhetően. A platform segítségével könnyen lehet követni az MVC (Model-View-Controller) szerkezeti mintát. Az MVC tervezési mintát az alábbi ábra szemlélteti:



A grafikus felület egy XML fájl segítségével van definiálva – ezeket FXML fájlokban tároljuk – ez a View (Nézet) réteg, ami egyszerűen összeállítható az ingyenes Scene Builder programmal. Ezek segítségével könnyen szerkeszthető komplex felület a Java programunkhoz. A JavaFX keretrendszer további újításokat hozott több téren is (pl. média és szálkezelés, …).

4.3.2. Maven

A Maven egy szoftver, amit az Apache szervezet keretein belül fejlesztenek 2002 óta. Feladata a szoftverprojektek menedzselése a build folyamat automatizálása. Funkcióban nagyon hasonló az Apache Ant nevű eszközre, de sokkal modernebb és könnyebben kezelhető. Bevezeti a POM (Project Object Model) fogalmát, ami a szoftvert és annak függőségeit írja le. A függőségeket hálózaton keresztül tölti le, amit a projekt elérési útvonalához csatol automatikusan.

https://maven.apache.org/

4.3.3. Java Database Connectivity (JDBC)

A Java Database Connectivity (rövidebben JDBC) egy olyan interfész a Java nyelvhez, ami az adatbázis elérését és manipulálását támogatja. Ebben a könyvtárban minden olyan osztály definiálva van, ami szükséges ezekhez az utasításokhoz. A JDBC minden esetben a relációs adatbázis modellekhez igazodik.

A szükséges osztályok a `java.sql` csomagban találhatóak. A kapcsolatot az adatbázissal egy `Connection` példány reprezentál. Egy ilyen objektumot a `DriverManager.getConnection()` metódussal hozhatunk létre. Paraméterként megadhatjuk az adatbázis elérési útvonalát is.

```
Connection conn = DriverManager.getConnection(
   ”jdbc:postgresql://localhost:5432/testdb”,
   ”user”,
   ”password”
);
```

Egy parancs elküldéséhez a `Statement` osztály végzi el. Az utasítást nekünk kell megadni egy szöveg formátumban, a helyes SQL szintaktikát betartva.

Az adatbázistól kapott valamennyi információt a Java egy `ResultSet` osztályba csomagolja. Az adatokat innen csak iterátor segítségével lehet kiolvasni.

```
Statement stmt = conn.createStatement();
ResultSet result = stmt.executeQuery(”SELECT *
FROM MyTable”);

while( result.next() ) {

   // adott sor feldolgozása

}
```

Nagyon fontos az adatbázis kapcsolatok zárása. A programot futtató operációs rendszer feleslegesen tart fenn kapcsolatokat, valamint adatbázis oldalon lehetnek felszabadítatlan kurzusok, amik feleslegesen foglalják a memóriát. Ehhez a `Connection` objektum `close()` metódusát kell meghívni.

Amennyiben bármilyen hiba lép fel az adatbázis művelet során úgy a keretrendszer egy `SQLException` kivétel osztályt fog dobni.

4.3.4. Java MongoDB Driver

A Driver lehetőséget ad, hogy egy Java alkalmazásból elérjünk egy MongoDB adatbázist. Mivel a JDBC erősen a relációs adatbázisokat támogatja, így egy Mongo-s kapcsolathoz feltétlen szükséges egy külső könyvtár, ami segítségével ezt meg lehet valósítani.

A könyvtár létrehozói egyszerűsíteni próbálták az adatbázis műveleteket megvalósító osztályokat. Az osztály, valamint a metódusok nevei a parancssoros lekérdezésekéhez hasonló. A lekérdezések, valamint a manipuláló műveletek is ugyanarra a sémára vannak szabva.

A JSON formátumot két osztály segítségével lehet reprezentálni. Az elterjedtem a `Document`, a másik a `Bson` osztály. Mindkettő a `java.util.Map` interfész implementációja, mely kulcs-érték párokat tárol. Felismerik az egyéb adatszerkezeteket is, amiket automatikusan a megfelelő JSON struktúrára alakít. Egy lekérdezést nem feltétlenül kell JSON formátumban megírni Java nyelvben, de a háttérben a `Document` egy példányát JSON üzenetté alakítja, ami a szerverhez megy. A szerver ugyancsak egy JSON választ ad, amit a keretrendszer egy iterálható szerkezetre alakít, amiben már `Document` típusú objektumok vannak.

5. A meglévő adatbázisok modelljei

A feladatom már meglévő adatbázisokra épül. Ezen adatbázisokon nagyobb változtatást nem végezhetek a kompatibilitás miatt. Kisebb változásokat viszont végrehajtottam, mint például indexek létrehozása, valamint más sémákra mutató idegen kulcsok törlése.

Ebben a fejezetben a két adatbázis szerkezetét szeretném bemutatni.

5.1. MongoDB

A kapott és jelenleg üzemben lévő adatbázisnak 3 db kollekciója van.

5.1.1. User

Ez a kollekció tárolja a felhasználók adatait. Egy felhasználóról az alábbi adatok állnak rendelkezésre: név, jelszó, e-mail cím, komment, típus, aktív-e, valamint az Andorid operációs rendszeren futó Java kliens osztályának a neve. A teszt adatbázisban körülbelül 200 dokumentum található.

5.1.2. Device

Azon eszközök dokumentumai, melyek segítségével a felhasználók adatokat, log-okat visznek fel a rendszerbe. Egy ilyen bejegyzés minden esetben egy eszközhöz kapcsolható. A teszt adatbázisban közel 200 dokumentum található (3 dokumentummal kevesebb, mint amennyi felhasználó van).

Egy eszközről eltároljuk a nevét (általában egy E-mail cím), leírását, tulajdonosát (ez egy referencia a User kollekció egy dokumentumára), hardware azonosítót, illetve azokat a felhasználókat, akik megtekinthetik az eszközhöz tartozó bejegyzéseket (pl. az alkalmazást használó kezelőorvos ellenőrzés céljából).

5.1.3. Observation

Ez az adatbázis legnagyobb kollekciója. Ebben vannak eltárolva az egészségügyi bejegyzések, étkezések. A teszt adatbázis kicsivel több, mint 91000 dokumentumot tartalmaz.

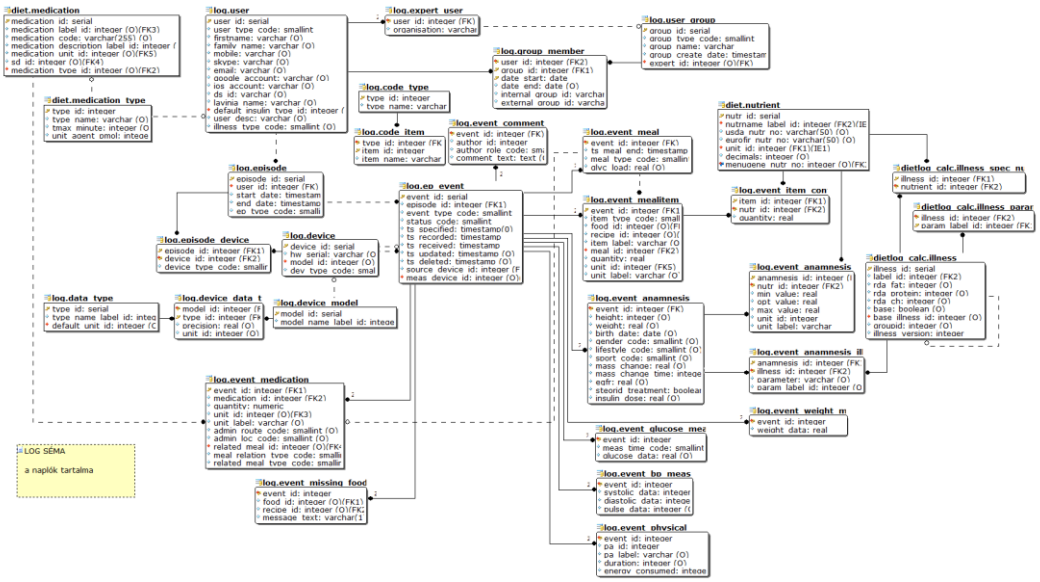
Ezeknek a dokumentumnak a sémái eltérnek a bejegyzés típusától függően. Jelenleg 11 féle séma létezik. Ezen sémák közös kulcsai: mely eszközzel lett

felvéve (referencia), mikor mentette el a felhasználó, mikor került be az adatbázisba, mikor módosították utoljára, törölték-e.

5.2. PostgreSQL adatbázis

Ez az adatbázis egy szakértői rendszer felett működik. Az ilyen rendszerekre jellemző a sok táblás adatbázisok. A PostgreSQL bevezeti a sémák fogalmát, ami nem azonos a MongoDB séma definíciójával. Az új adatbázisnak csak egy sémájával kell foglalkoznom, ez a log séma.

Az adatbázis logikai sémája a következő:



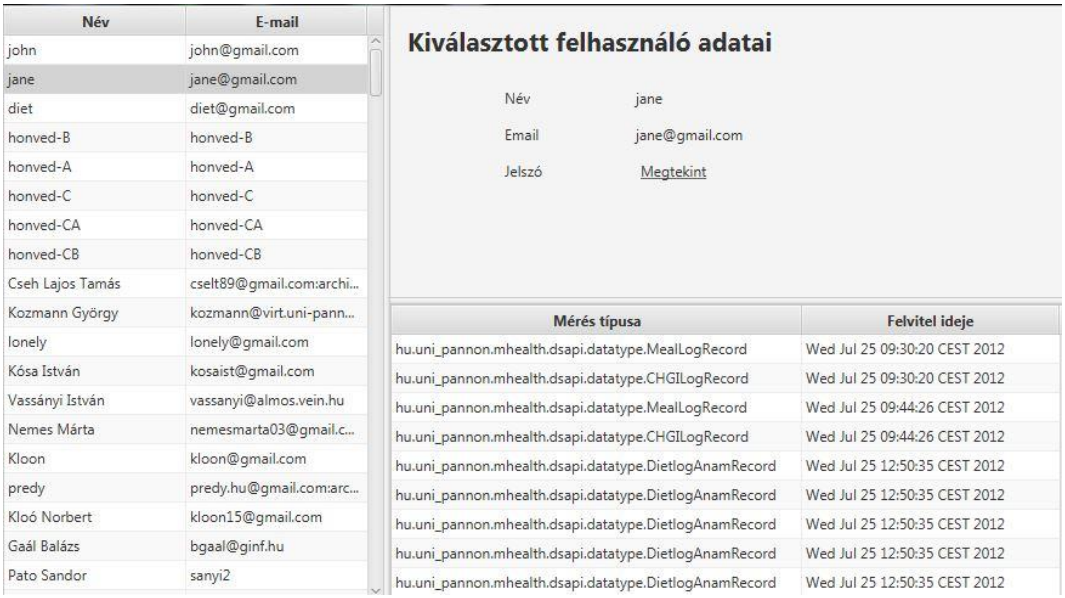
A táblákra és azok kapcsolataira nézve látszik, hogy ez egy esemény alapú adatbázis. Ez onnan látszik, hogy a táblák nagy része az `event` táblára hivatkozik külső kulccsal.

A sémát három nagyobb részre lehet osztani. Az egyik csoport a felhasználók és a hozzá tartozó eszközök adatai. Ebbe beletartoznak a felhasználók és csoportjaik, valamint az eszközök pontosabb adatai vannak. A másik fontos rész a felhasználó által felvitt étkezések tartalma. Ezek a táblák a fenti képen az `event` táblától jobbra találhatóak. Az utolsó csoport a bevett gyógyszerek tárolására alkalmas táblákat tartalmazza.

6. Az eddig elkészült rendszer

Az adatbázis kapcsolatok tesztelésére, illetve egy alapvető művelet elvégzése érdekében egy eldobható prototípust készítettem a rendszer egy kis szeletéről.

A felületen meg lehet tekinteni a MongoDB-ben tárolt adatokat. Egy reszponzív ablak bal oldalán található táblázatból ki lehet választani a felhasználókat. Ekkor a kijelölt felhasználó minden adata megjelenik az ablak bal oldalán. Ennek az alsó részén látható a hozzá tartozó bejegyzések idő szerint sorrendbe szedve:



Ez segítette az adatszerkezetek megértését valamint kezelését. A kész szoftver kinézetének a stílusán nem fogok változtatni. Ez az alapértelmezett stílus a JavaFX környezetben (Modena néven).

A másik szelet a rendszerből egy pár táblás migráció. Ennek a szerkezete nem illik a fent ismertetett rendszertervbe, de az elindulásban segített. A MondoDB-ben lévő `User` és `Device` kollekciókat hiánytalanul át tudom másolni a Postgres adatbázisba. Az utóbb említett adatbázisban ezek a `user`, `episode`, `episode_device`, `device` táblákat jelenti. Ez 778 rekordnak felel meg.

Ennek az átlagolt időigényét fejezi ki az alábbi táblázat:

Művelet	Időigény
Felhasználók másolása	2200 msec

Eszközök másolása	400 msec
Kapcsolótáblák feltöltése	100 msec

Fontos megjegyezni, hogy ezeknél a folyamatoknál csak másolás történik. Ez azt jelenti, hogy nincs még benne az a vizsgálat, ami ellenőrzi, hogy az adott entitás már tárolva van-e a cél adatbázisban.

A kapcsolótáblák feltöltése saját adatszerkezetből történik, ezért gyorsabb a többinél. Ez azt jelenti, hogy egy osztályban vannak eltárolva a külső kulcshoz kapcsolódó adatok. A táblák helyett a memóriában tárolt adatokat sokkal gyorsabban el lehet érni és nem foglalnak nagy helyet sem.

7. A következő félév ütemezése

A következő félévben ki kell bővíteni a meglévő rendszert a specifikáció és a rendszerterv alapján. Fontosabb és sűrűn használt komponensektől haladok a kevésbé fontosabb területek felé. A feladat elkezdését a pontos adatbázis interfész kialakításánál fogom kezdeni. Egyes komponenseket párhuzamosan is lehet fejleszteni idővel.

A rendszert iteratívan fogom implementálni. Ez segít nekem is és a témavezetőnek is a feladat átlátásában valamint előremenetelésben. Előre definiált időpontok nincsenek, de egy teljes modulra nincs sok idő, így nem lehet hosszú iterációkat tervezni.

A felhasználói felületeket viszonylag kevés idő alatt el lehet készíteni a Scene Builder segítségével. Fontos, hogy a felületre nézve az könnyen kezelhető legyen, valamint minél több információt nyújtson a felhasználó számára.

Továbbá fontos, hogy lépést kell tartani az új kliens fejlesztésével is. Ennek a rendszernek előbb el kell készülnie, mint az új Lavinia kliensnek. Ennek az egyetlen egy oka az, hogy az új kliens is használható legyen a régi felhasználóknak. Biztosítani kell tehát a zökkenőmentes átváltást számukra.