Debreceni Egyetem

Informatika Kar

Alkalmazásfejlesztés JAVA nyelven

Témavezető:

Dr. Boda István

Egyetemi adjunktus

Készítette:

Viraszkó László

Programozó matematikus

Debrecen

2007

Tartalomjegyzék

Ábrák jegyzéke

1. Bevezetés

1.1. A szakdolgozatom témája

Ez a dolgozat egy konkrét alkalmazásfejlesztési folyamatot ír le Java nyelven. Az alkalmazás célja egy pékség információs rendszerének a megvalósítása. A bevezetés további szakaszaiban bővebben kifejtem a témát és a feladatot, megnevezem és bemutatom a felhasznált technológiákat és módszertanokat.

1.2. Témaválasztás

Az egyetemi képzés során a programozó matematikus hallgatók főként elméleti tudásukat bővítik és jóval kevesebb gyakorlati ismeretet szerzek. Ennek hiánya a munkaerőpiacon való elhelyezkedéskor fokozottan jelentkezik. A gyakorlati tudás megszerzésére és az elméletben tanultak alkalmazására csak néhány lehetőség adódik, melyeket érdemes kihasználni. Az egyik ilyen lehetőség a rendszerfejlesztés technológiája tantárgy, a másik pedig a szakdolgozat. Emiatt választottam szakdolgozati témámnak egy konkrét, valós alkalmazás elkészítését, melynek a fejlesztése során az elméletben tanultaknak nagy részét ki tudom próbálni, jobban megismerem, ezáltal a tudásomat is mélyíthetem. Szakdolgozatom írása során megpróbálok egy teljes alkalmazásfejlesztési módszert *"végig vinni"* a tervezéstől kezdve, a fejlesztésen keresztül a tesztelésig. Az elkészült programot továbbfejlesztve az állásinterjúk során referenciaként bemutatva próbálom majd a felvételi esélyemet növelni.

A Java nyelvet azért választottam, mert a Debreceni Egyetem programozó matematikus szakán a programozási tantárgyak jelentős részében erről a tárgyról tanultunk, a legtöbb ismeretet ezen a nyelven sajátítottuk el. Az egyetemi évek alatt ezt a nyelvet és az objektum orientált technológiát megszerettem és saját fejlesztéseim során is ezt alkalmazom. Ezért döntöttem a mellett, hogy ilyen téren próbálom még bővíteni a tudásomat és szerzek további tapasztalatokat.

1.3. Felhasznált tervezési és fejlesztési módszerek és szoftverek

A dolgozatom írása és az alkalmazás fejlesztése során törekedtem arra, hogy a Rendszerfejlesztés technológiája órán tanultakat a gyakorlatban használjam. Ezért az alkalmazás tervezése során számos UML diagramot (Felhasználó esetdiagram, Komponens diagram, Alkalmazás diagram) és egyéb dokumentumokat (Fogalomszótár, Forgatókönyv, Folyamatok) is készítettem, melyeket a dolgozatban részletesen bemutatok.

A fejlesztés során csak ingyenes szoftvereket használtam, főként azokat, melyeket a gyakorlatokon is alkalmaztunk. Ezek közül néhány: NetBeans, HSQLDB.

A rendszerfejlesztés során alkalmazott technológia a feltáró fejlesztés volt, mely egy evolúciós fejlesztési modell. Számomra azért volt ez kézenfekvő, mert a rendszer irányában támasztott igények a tervezés elején még nem voltak teljesen tisztázottak.

2. Alkalmazás megtervezése, diagramok, felületek, elvárások

Egy alkalmazás készítése estén az elkészült termék minősége és a fejlesztés során előálló nehézségek mennyisége nagyban függ attól, hogy a konkrét kódolás előtt mekkora hangsúlyt fektettünk a tervezésre. Egy átfogó terv készítése sok előnnyel jár:

- Elkerülhetőek a konfliktusok a megrendelővel, mindkét fél tisztán látja a feladatot, a megrendelő is tisztázza, hogy pontosan mit szeretne és a fejlesztő is tudja, hogy mit várnak el tőle
- Több fejlesztő esetén mindenki tudja, a pontos feladatát, ezáltal a külön-külön elkészített részek integrációja egyszerűbb lesz
- Egy igazán jó tervezés során maga a kódolás szinte automatikusan végezhető, a fejlesztőnek nincs akkora felelőssége, nem kell az alkalmazás működésére, kinézetére vonatkozó döntéseket hoznia
- Jobban tervezhető az alkalmazásfejlesztés időtartalma, elkerülhetőek a csúszások

A tervezés során el kell készítenünk az alkalmazás modelljét. Ez a programkódhoz képest egy magasabb absztrakciós szint, mivel eltekintünk a konkrét megvalósítási technikák zavaró részleteitől. A modellek elkészítése során ellenőrizhető az alkalmazás megvalósíthatósága és esetleges hiba vagy hiányosság esetén a modell jóval könnyebben módosítható, mint maga a programkód. A modellek absztraktságából kifolyólag sokkal érthetőbbek és a megrendelővel való egyeztetést is nagyban elősegítik. Főként vizuális és szöveges eszközökkel készítjük ezeket a modelleket, a megértés és a használhatóság miatt.

Célszerű olyan modellező nyelvet vagy módszert választani, amit sokan megértenek, egyszerű, de mégis megoldást szolgáltat minden esetre. Én emiatt választottam az alkalmazás tervének elkészítéséhez az UML modellező nyelvet.

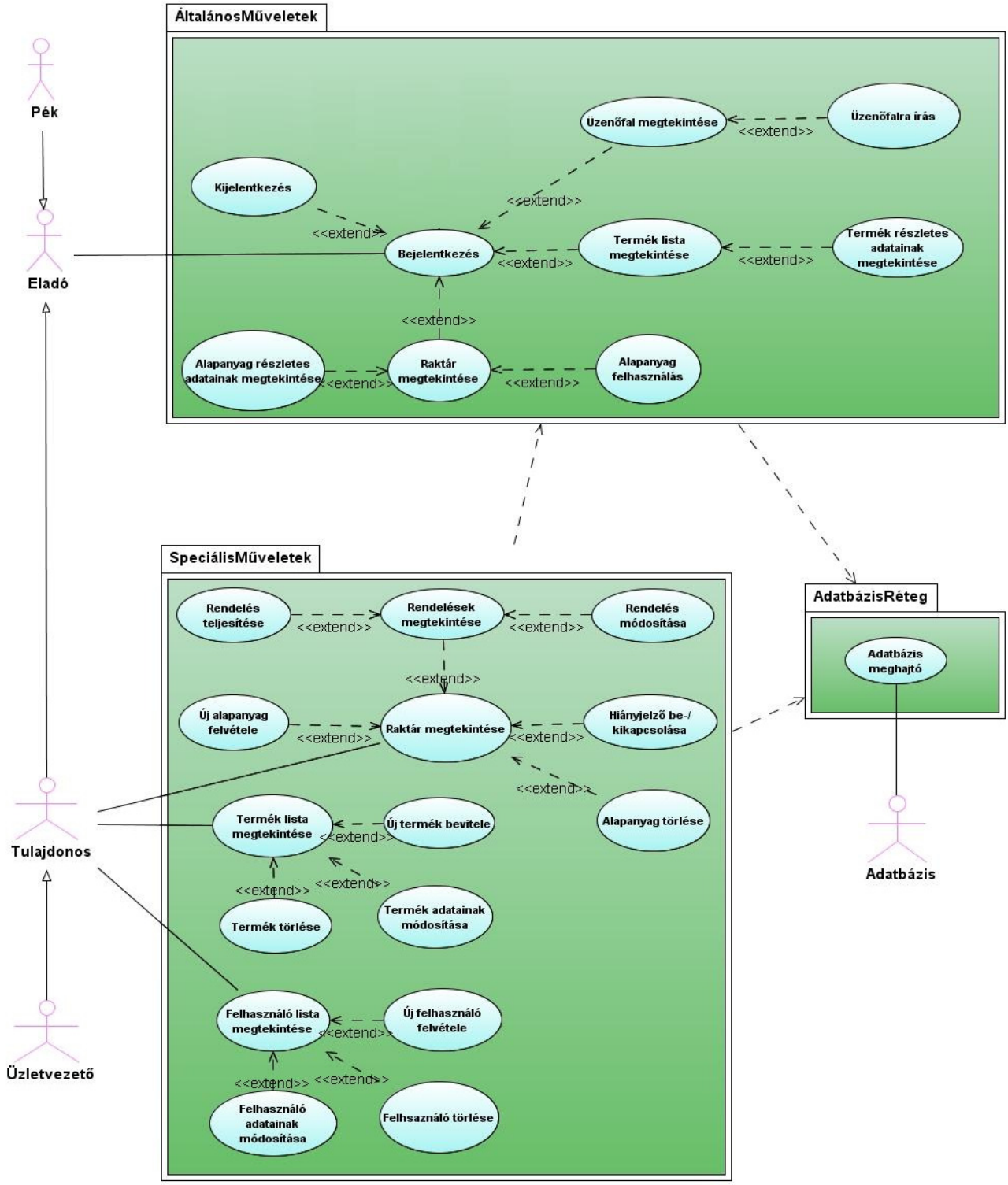
Gondolati modell → modellező nyelv → programozási nyelv → gépi nyelv

2.1. Az alkalmazásfejlesztés során elkészített UML diagramok

2.1.1. Használati eset diagram

Ezzel a diagrammal összegyűjthetjük az alkalmazással szemben támasztott alapvető követelményeket, továbbá tisztázhatjuk a felhasználók kapcsolódási pontjait a rendszerhez. Ezáltal a különböző felhasználói szerepköröket is meghatározzuk. Definiáljuk a kapcsolódási pontok esetén a különböző interakciókat, kiemelhetünk egyes résztevékenységeket, illetve jelölhetünk megvalósítási módokat.

A használati eset diagramon a felhasználókat és egyéb a rendszerhez kapcsolódó külső „elemeket" *aktoroknak* (actor) nevezzük. A kifejlesztendő alkalmazás esetén az első tervezési lépés lehet az aktorok meghatározása. Jelen esetben az Eladó-t és az Adatbázis-t határoztam meg. Az Eladó-hoz további alváltozatokat is kijelöltem, melyek különböző feladatokat láthatnak el. Ebben az esetben az Eladó az egy általános szerep, melynek a Pék és a Tulajdonos egy-egy pontosítása. Az Üzletvezető pedig a Tulajdonos pontosítása. Az ábrán jól látszik, hogy az Eladó egy általános szerep, ezáltal általános funkciókat érhet el, mint például a Bejelentkezés. A Tulajdonos, aki az Eladónak egy pontosítása, elérhet olyan funkciókat is, melyeket egy általános Eladó nem. Megtekintheti a raktáron lévő árukat, kilistázhatja a termékeket, felhasználókat, stb. A diagramon lévő ellipszisek a használati esetek (use case), melyek a rendszer funkcióit, külső kapcsolódási pontjait írják le. A leghangsúlyosabb, legfontosabb használati esetekhez kapcsolódnak az aktorok. Ezeknek az eseteknek a változatai-t <<extend>> sztereotípiával jelölt szaggatott vonalú nyíllal kötjük össze. Például a Felhasználó lista megtekintése használati esethez tartozó változatok az Új felhasználó felvétele, a Felhasználó törlése és a Felhasználó adatainak a módosítása. Az alábbi UML diagram a tervezés kezdeti stádiumában készült, a végleges használati eset diagram annyiban változott, hogy kizárólag a tulajdonos és az üzletvezető számára elérhető funkciók: Új felhasználó felvétele, Felhasználó törlése, Felhasználó adatainak módosítása. Az összes többi funkció minden felhasználó számára elérhető.

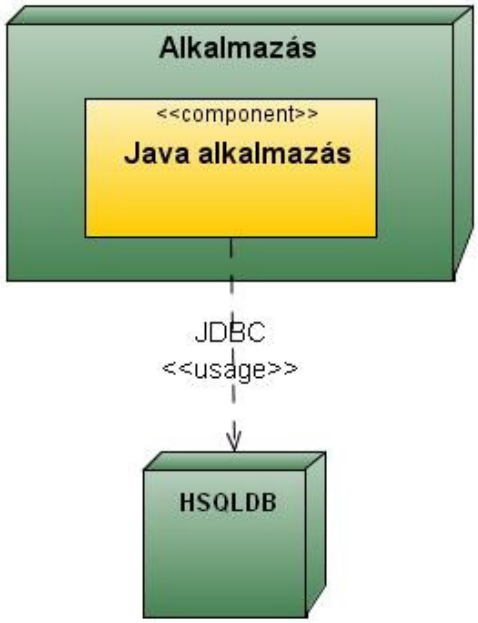


1. ábra – Használati eset diagram

A használati eset diagram segítségével egyértelművé válnak a rendszer határai, a rendszert használók szerepkörei és az általuk elérhető funkciók. A programozók számára pedig egyértelmű, hogy milyen funkciókat kell megvalósítani. Ezek a felhasználói esetek a fejlesztés menetének ütemezésére is felhasználhatóak. A legfontosabb funkciókat kijelölve és először azokat kifejlesztve, azokat több ideig lehet tesztelni, ezáltal azok megbízhatóbbak lesznek. A használati eset diagram tekinthető az alkalmazással szemben támasztott követelmények térképeként, a funkciók grafikus tartalomjegyzékeként.

2.1.2. Alkalmazási-diagram

Ezzel a diagrammal a rendszer elemeit és a közöttük lévő kapcsolatot reprezentálom. Az alkalmazási diagram alapeleme a csomópont, mely az egyes elemeket jelenti. Az én esetemben ez két csomópontot jelent az adatbázist és magát az alkalmazást. A diagramról az is leolvasható, hogy az adatbázis irányában JDBC protokollon keresztül tudunk kommunikálni. A JDBC miatt, ha később valamilyen oknál fogva le szeretnénk váltani az adatbázis kezelőt, akkor azt könnyen megtehetjük.



2. ábra - Alkalmazás diagram

2.2. Az alkalmazásfejlesztés során elkészített szöveges dokumentumok

2.2.1. A rendszer funkciói

Az alkalmazás által biztosított funkciók tervét összegyűjtöttem egy dokumentumban. Ez a dokumentum segít a funkciók pontosításában a megrendelővel, továbbá a konkrét kód megírásánál is nagy segítség a fejlesztő számára. Ezt a dokumentumot a felhasználók számára is át lehet adni, mely az alkalmazás funkcióiról tájékoztatja a végfelhasználókat.

***Bejelentkezés*:**
A rendszerbe való belépés. A pékségben a különböző alkalmazottak szerepkörüknek megfelelően a szoftver különböző részeihez férhetnek hozzá, valamint a bejelentkezés véd az illetéktelen hozzáféréstől is.

Kijelentkezés:
A rendszerből való kilépés.

Alapanyagok listázása:
Az alapanyagok és egyéb az üzlet működéséhez szükséges anyagok listázása a raktáron lévő mennyiségek feltüntetésével. A lista a rendszer összes felhasználója számára megtekinthető.

Új alapanyag felvétele:
A pékáruk előállításához szükséges alapanyagok adatainak rögzítése a rendszerben. Mindenki hozzáférhet.

Beszerzés:
A kiválasztott alapanyag rendelése rögzíthető a rendszerben. A rendszer automatikusan növeli a rendelendő mennyiséggel a raktárkészleti mennyiséget. Mindenki hozzáférhet.

Hiányjelzés:
A minimum limit alatti mennyiséggel (ami előzetesen be lett állítva) rendelkező alapanyagok listáját adja meg, jelezvén, hogy rendelni kell belőle.

Raktáron belüli adatok módosítása:
Módosíthatjuk a kiválasztott alapanyag beszerzési árát és a minimum limitet.

Alapanyag törlése:
Amennyiben egy a rendszerben lévő alapanyagra már nincs szükség a pékáruk előállításához, akkor ezzel a funkcióval törölhetjük ezt.

***Termékek listázása*:**
A pékség által forgalmazott pékáruk listáját jeleníti meg a fontosabb adatokkal (megnevezés, termékcsoport, raktári mennyiség). A rendszer összes felhasználója elérheti.

Új termék bevitele:
Új pékáru adatainak a rögzítése.

Termék részletes adatainak megtekintése:
A kiválasztott pékáru adatainak részletezése. A rendszer összes felhasználója elérheti.

Termék adatainak módosítása:
A kiválasztott pékáru adatainak a módosítása.

Termék törlése:
A kiválasztott pékáru törölése a rendszerből.

Termék előállítása:
A kiválasztott pékáru előállítása, aminek következtében fogynak a megfelelő alapanyagok mennyiségei a raktáron.

Eladás:
A kiválasztott pékáru eladása, aminek következtében csökken az eladott darabszámmal a raktáron lévő mennyisége.

Üzenő fal megtekintése:
Az alkalmazottak itt tekinthetik meg az eddig rögzítésre került észrevételeket. A rendszer összes felhasználója elérheti.

Üzenő falra írás:
Az alkalmazottak ide rögzíthetik észrevételeiket, hogy mire lenne szükség. Itt olyan dolgokat is megnevezhetnek, ami esetleg eddig még nem szerepelt az üzlet korábbi megrendeléseiben. Pl. új termék bevezetésénél új alapanyag javaslat. A rendszer összes felhasználója elérheti.

Felhasználó lista megtekintése:
A rendszerhez hozzáférési jogokkal rendelkező felhasználók listája. A funkció csak az adminisztrátor számára érhető el.

Új felhasználó felvétele:
Új felhasználó felvétele a rendszerhez. A funkciót csak az adminisztrátor érheti el.

Felhasználó törlése:
Felhasználó törlése a rendszerből. A funkciót csak az adminisztrátor érheti el.

Felhasználó adatainak módosítása:

A rendszerben lévő felhasználó adatainak módosítása. A funkció csak az adminisztrátor számára érhető el.

2.2.2. Forgatókönyvek

A követelmények pontosításának egyik módszere, ha az egyes használati esetekre megpróbáljuk leírni részletesen. Egy forgatókönyvben meg kell adni, hogy az adott funkció milyen párbeszédet igényel az aktor és az alkalmazás között. A forgatókönyv az elkészült alkalmazás estén tesztelésre is alkalmazható, melynek a segítségével bizonyítható, hogy valóban az készült el, amiben a megrendelővel megegyezett a készítő. Az általam készített teljes forgatókönyv a mellékletben megtalálható. Néhány fontosabb használati eset:

A felhasználó bejelentkezik a rendszerbe:

- Elindítja az alkalmazást
- A megjelenő párbeszéd ablakban beírja a felhasználói azonosítóját és a jelszavát
    - A bejelentkezés sikeres
    - A bejelentkezés sikertelen
        - Újra megpróbálja
        - Kilép az alkalmazásból

A rendszer funkciói csak a bejelentkezés után érhetők el.

A felhasználó módosítja egy alapanyag adatait:

- Megtekinti egy alapanyag részletes adatait
- Kiválaszt egy alapanyagot
- A módosítandó adatokat átírja
- Kiválasztja az ***Adatok módosítása***-t

A felhasználó új terméket hoz létre:

- Megtekinti a termékek listáját
- Kiválasztja az ***Új termék***-t
- Megjelenik egy új ablak, ahol megadja az új termék adatait
    - Megadja az új termék előállításához szükséges alapanyagokat és azok mennyiségét

A felhasználó elad egy terméket:

- Megtekinti egy termék részletes adatait
- Kiválasztja a terméket
- Kiválasztja az ***Eladás***-t.
- Megadja az eladni kívánt darabszámot

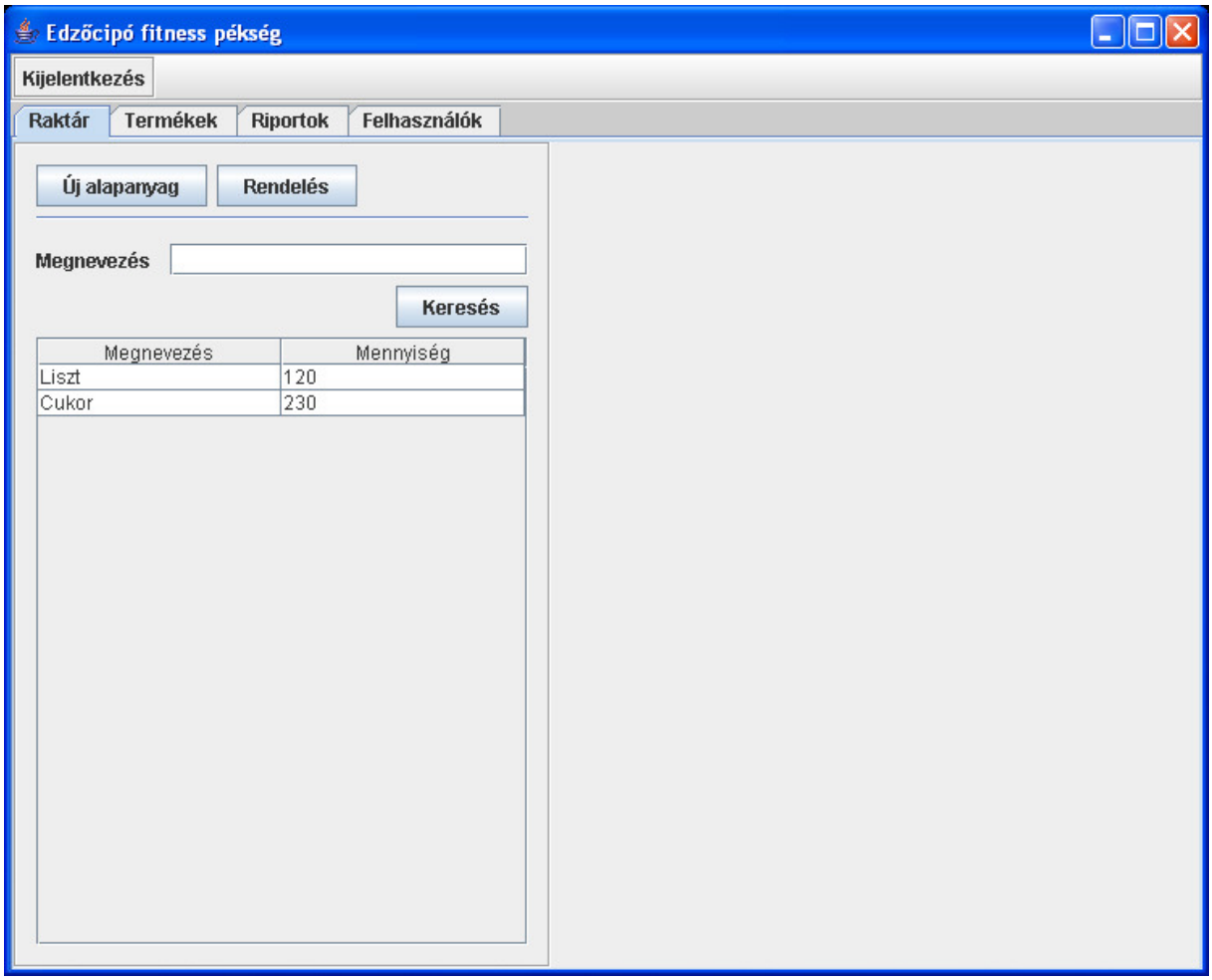
Az adminisztrátor töröl egy felhasználót:

- Megtekinti egy felhasználó részletes adatait
- Kiválasztja az ***Felhasználó törlése***-t.

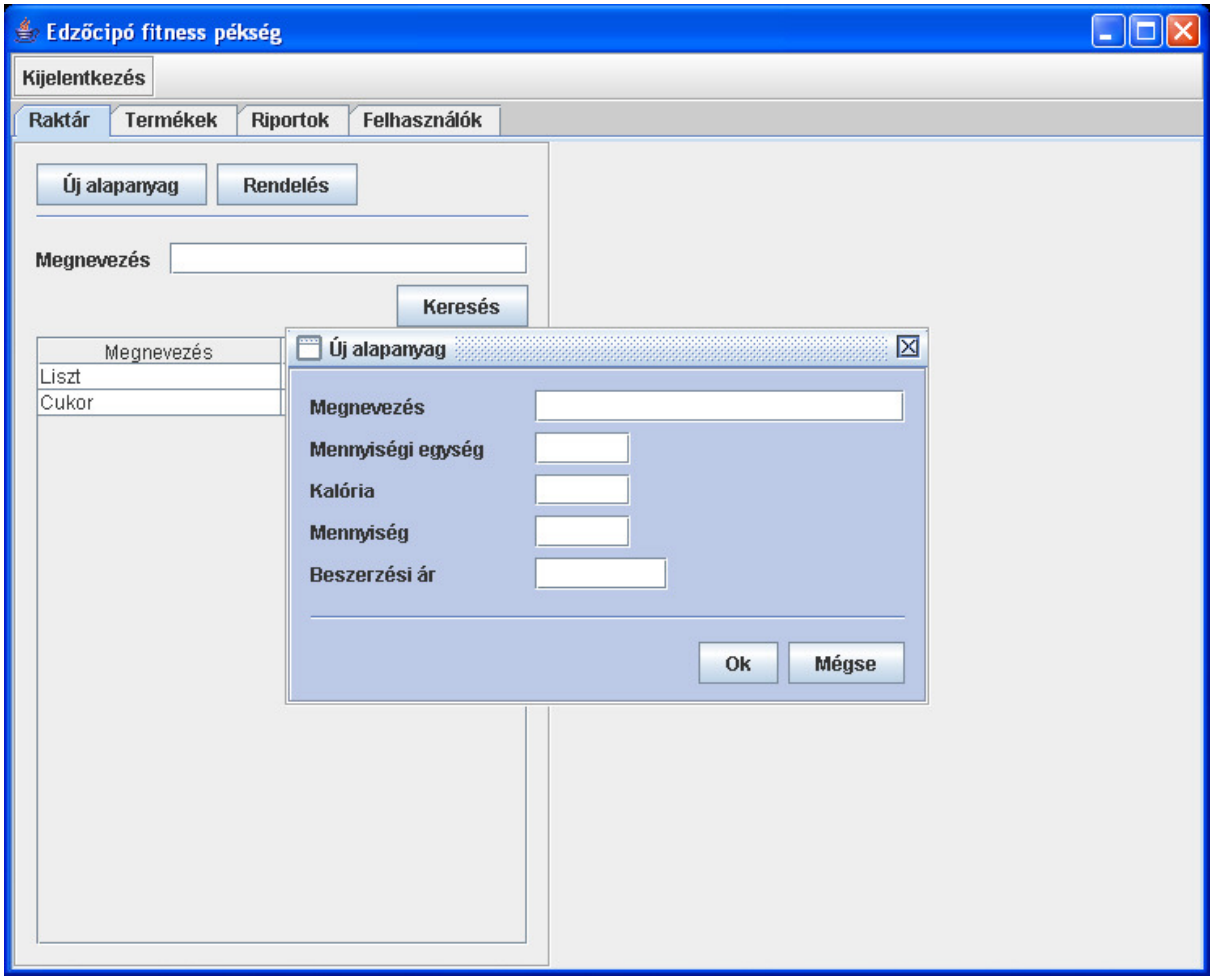
2.2.3. Felhasználói felületek

A funkciók leírásával és a forgatókönyvekkel pontosíthatjuk az egyes használati esetek lépéseit, azaz a felhasználóval történő párbeszédet vagy a végrehajtandó tevékenységsorozatokat. A forgatókönyvek egyes elemei az aktorként megjelenő felhasználóval vagy külső rendszerrel történő információátadást írnak le. A használati esetek pontosításának következő lépéseként célszerű megtervezni az információátadás eszközeit, a felhasználói felületeket. Ezek szintén alkalmasak a megrendelővel történő egyeztetésére. Továbbá ez a diagram is pontosítja a rendszer funkcióit. Ugyanis a felhasználó a felülettel fog találkozni, a funkciókat azon keresztül éri el. Tehát egy alkalmazásban csak azok a funkciók látszanak, melyeket a felületről el lehet érni. Az alkalmazásom tervezése során kiindulásként az alábbi felületeket terveztem meg, melyek a későbbiekben pontosításra kerültek:

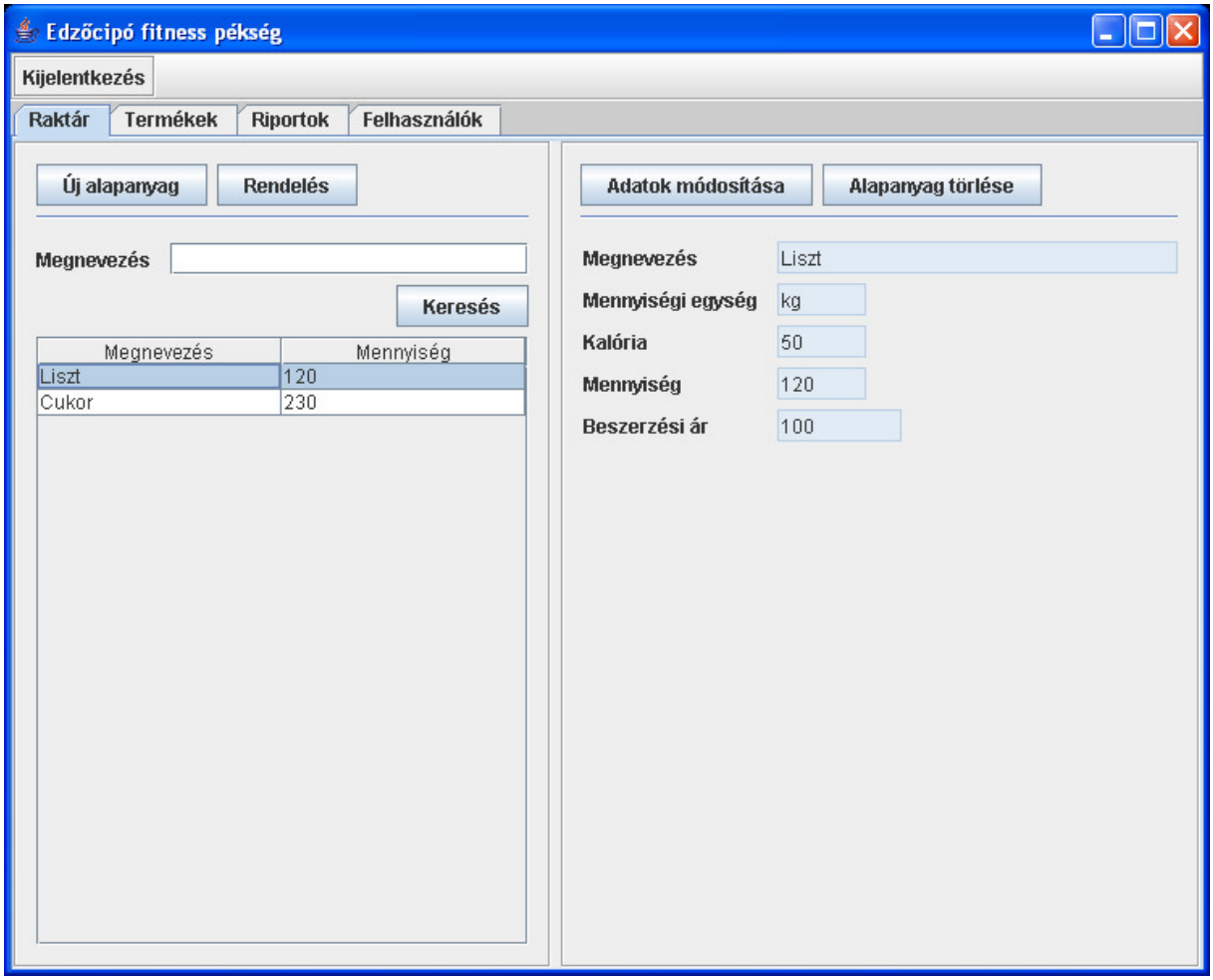
Alapanyagok kezelése



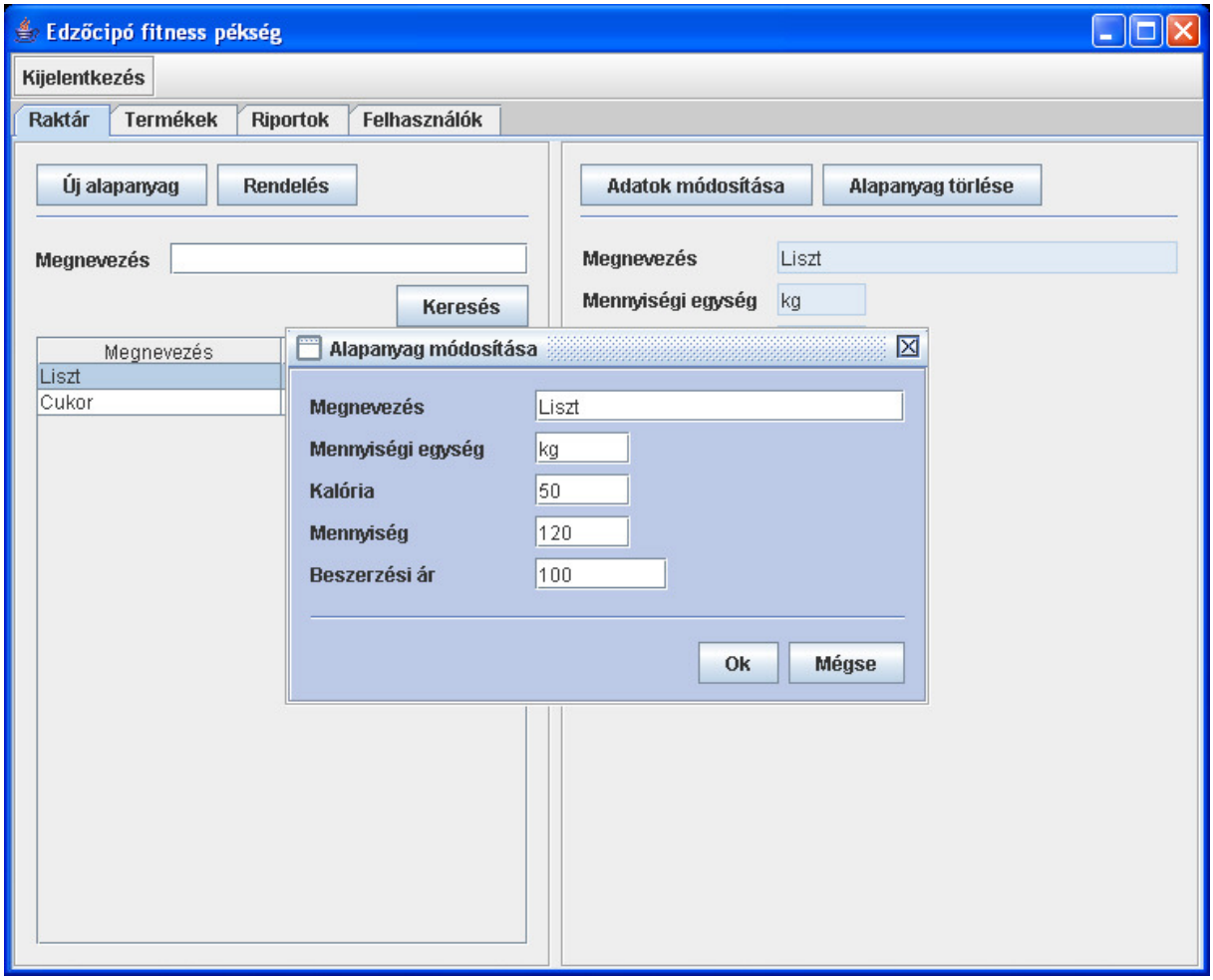
3. ábra - Alapanyagok listázása



4. ábra - Új alapanyag beszúrása

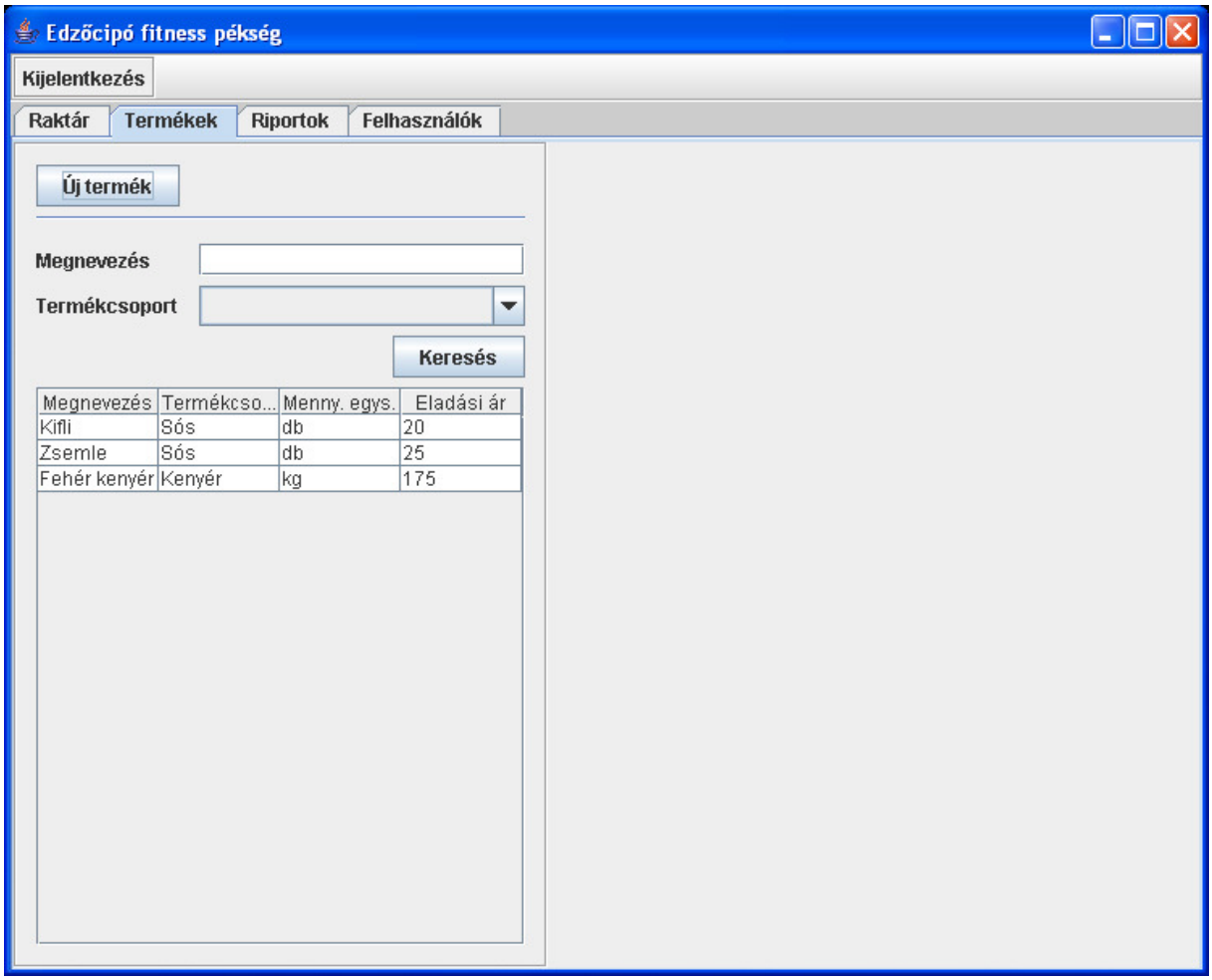


5. ábra - Alapanyag részletei

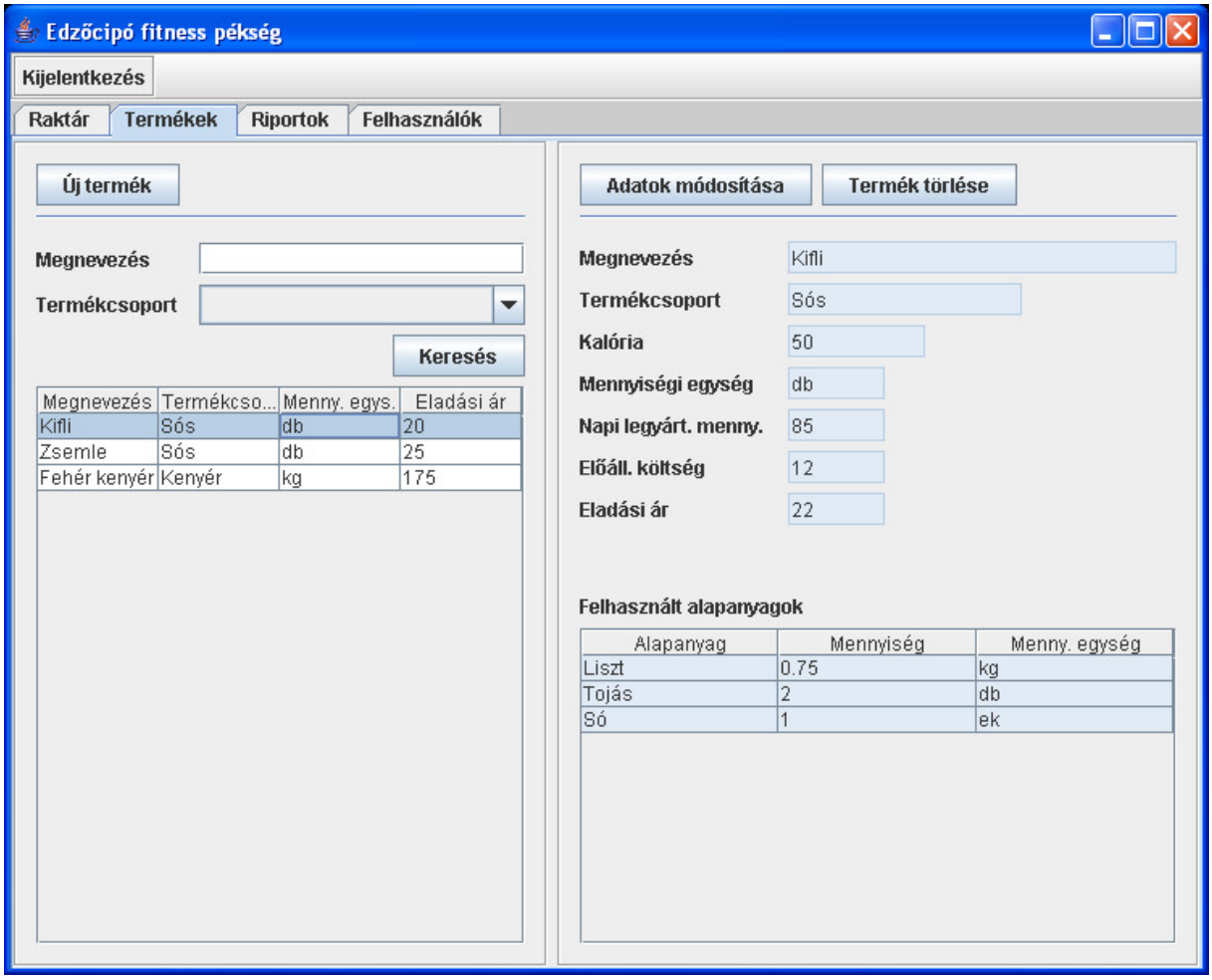


6. ábra - Alapanyag módosítása

Termékek kezelése:

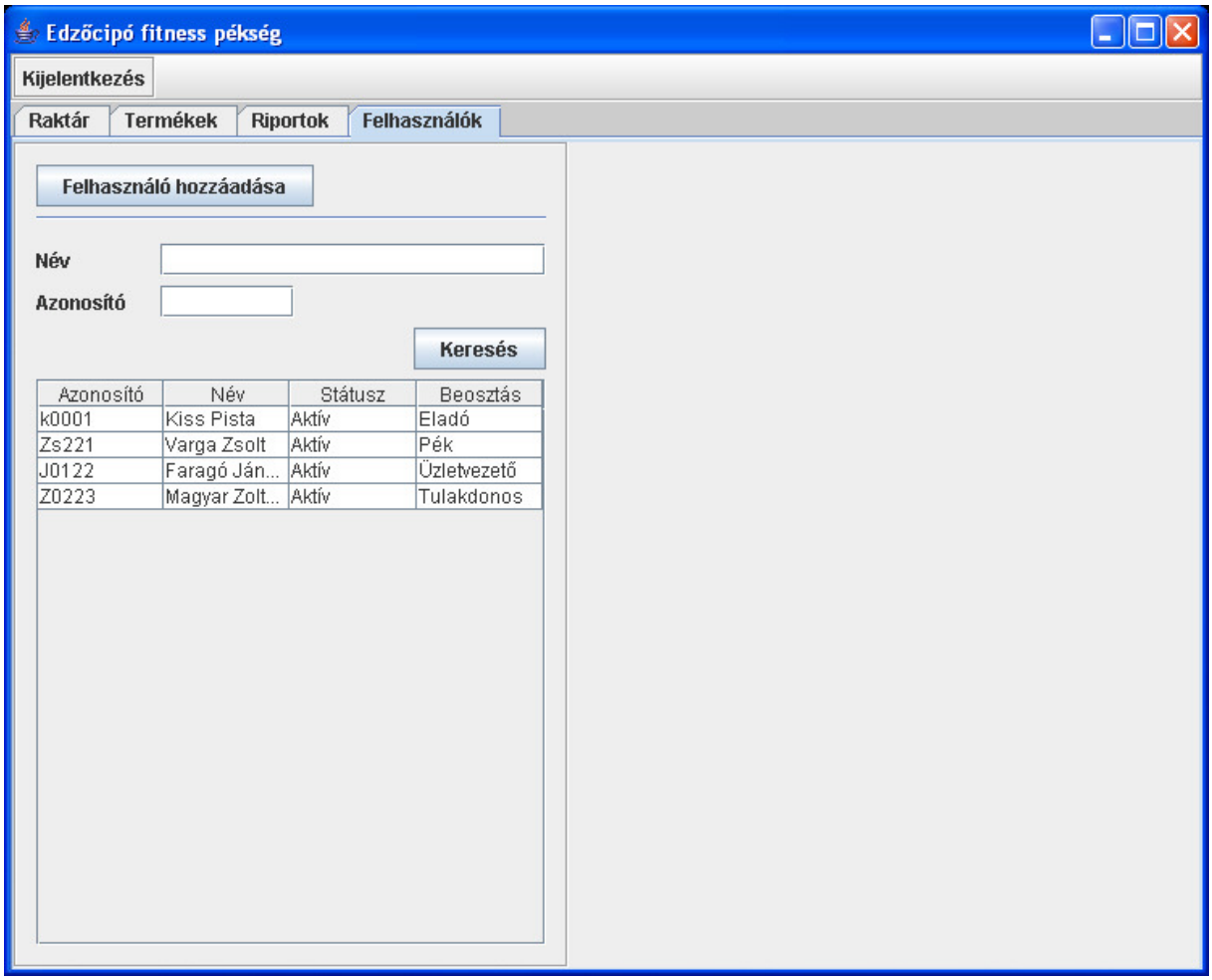


7. ábra - Termékek listázása

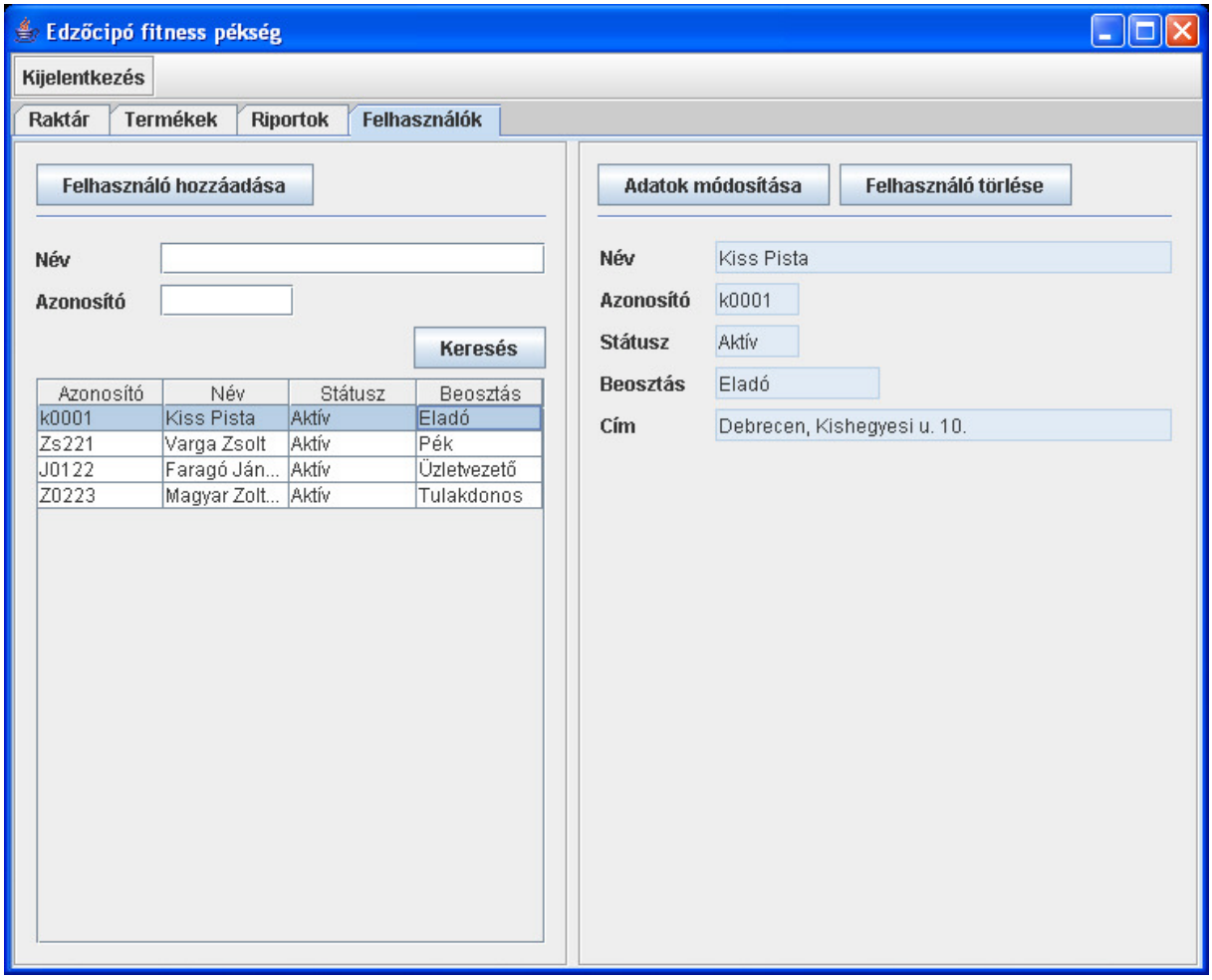


8. ábra - Termékek részletes adatai

Felhasználók kezelése:



9. ábra - Felhasználók listázása



10. ábra - Felhasználó részletes adatai

2.2.4. Fogalomszótár

Továbbá a rendszer fejlesztése során készítettem egy szöveges dokumentumot, a fogalomszótár-t, melyben leírom az egyes fogalmak rövid kifejtését. Ehhez az alkalmazáshoz készített fogalomszótáram:

***Pékáru*:**
A cég által forgalmazott termék. Jellemző adatai:

- megnevezés
- termékcsoport
- eladási egység
- raktári mennyiség
- kalóriatartalom
- előállítási költség

- eladási ár
- felhasznált alapanyagok

***Alapanyag*:**
A pékáru előállításához használt anyagok. Jellemző adatai:
- megnevezés
- raktáron lévő mennyiség
- minimum limit (ha ennél kevesebb, akkor rendelni kell)

***Egyéb anyagok*:**
Ide sorolandó minden alapanyagon kívüli anyag, melyre az üzlet működése során szükség lehet. Például:
- csomagoló anyagok
- tisztító szerek
- irodaszerek

Ezek jellemző adatai az alapanyagokéinak megfelelőek.

Beszerzés:
A pékáru előállításához használt alapanyagok beszerzése. Jellemző adatai:
- megnevezés
- mennyiségi egység
- kalória
- aktuális mennyiség
- beszerzési ár
- minimum limit
- rendelendő mennyiség

***Eladás*:**
A forgalmazott pékáruk eladási adatai. Jellemzői:
- megnevezés
- termékcsoport
- mennyiségi egység
- kalória
- mennyiség
- előállítási költség
- eladási ár
- darabszám

***Üzenő fal*:**
Az alkalmazottak közötti kommunikáció eszköze.

***Alkalmazott*:**
A cég által foglalkoztatott személyek. Ide tartoznak:
- pék (, a pékáru előállításáért felelős személy)
- üzletvezető (, irányítja az üzletet, felelős az árubeszerzésért)
- tulajdonos (, hivatali ügyeket intézi)
- eladó (, a pékáru értékesítését végző személy)

***Pékség*:**
Pékáru előállításával és forgalmazásával foglalkozó kereskedelmi egység.

Termékcsoport:
Pékárukat összetételük alapján az alábbi csoportokba soroljuk:

- Édes
- Sós
- Kenyerek

3. Az elkészült alkalmazás leírása

3.1. Az elkészült alkalmazás leírása a JAVA oldaláról

Az implementáció során a csomagrendszert két, funkcionalitásában jól elkülönülő csomagra osztottam. A következőkben e csomagok osztályait, ill. az osztályok funkcionalitását fogjuk áttekinteni. Az osztályok megtalálhatók a CD melléklet *Forráskódok/src/* mappában.

3.1.1. Osztály diagram

11. ábra - Osztály diagram

3.1.2. A `main` csomag

Ez a csomag tartalmazza az alkalmazás grafikus megjelenítéséért felelős osztályokat. A csomagban található másik, `MyComboBoxUtil` osztály segítségével a felületen lévő táblázat (JTable) cellájába helyezhetünk el legördülő listát (`JcomboBox-ot)`.

3.1.3 Az `Edzocipo` osztály

Ez az osztály tartalmazza az alkalmazást megjelenítő ablakot, egyéb grafikus megjelenítő eszközöket valamint a program belépési pontját képző statikus `main` metódust. Emellett ez az osztály tartja a kapcsolatot a `db` csomag `DBManager` osztályával, melyen keresztül az adatbázisban tárolt adatokhoz férhet hozzá.

Az attribútumok

- `screenWith`

  Az képernyő pixelekben mért szélessége.
- `screenHeight`

  Az képernyő pixelekben mért magassága.
- `SELLER_MODE`

  A pékekből és eladókból álló szerepkört reprezentáló nevesített konstans.
- `CHIEF_MODE`

  Az üzletvezető és a tulajdonos szerepkörét reprezentáló nevesített konstans.
- `mode`

  Az aktuális szerepkör azonosító változó.
- `ingredients`

  Az adatbázisban aktuálisan tárolt alapanyagokat reprezentáló dinamikus tömb.
- `products`

  Az adatbázisban aktuálisan tárolt termékeket reprezentáló dinamikus tömb.
- `groups`

  Az adatbázisban aktuálisan tárolt termékcsoportokat reprezentáló dinamikus tömb.

- `users`

  Az adatbázisban aktuálisan felhasználókat reprezentáló dinamikus tömb.
- `messages`

  Az adatbázisban aktuálisan tárolt üzeneteket reprezentáló dinamikus tömb.
- `currentUser`

  Az alkalmazásba aktuálisan bejelentkezett felhasználó.
- `currentIngredSearchedIndices`

  Az alkalmazásban aktuálisan beállított szűrési feltételeknek megfelelő `ingredients` tömbbeli indexek.
- `currentProdSearchedIndices`

  Az alkalmazásban aktuálisan beállított szűrési feltételeknek megfelelő `products` tömbbeli indexek.
- `currentUserSearchedIndices`

  Az alkalmazásban aktuálisan beállított szűrési feltételeknek megfelelő `users` tömbbeli indexek.

A `main()` metódus

Az osztály `main` metódusában önmagát próbálja példányosítani egy új szálban. Kivétel esetén a rendszerverem tartalma kerül kiírásra.

A konstruktor

Az osztály egyetlen, publikus konstruktorral rendelkezik. Példányosítás során a konstruktor a `DBManager.connect()` metódus hívásával csatlakozik az adatbázishoz. Majd a megfelelő `db.DBManager` osztálybeli metódusok segítségével inicializálódnak a `products`, `ingredients`, `users` ill. `messages` tömbök. Ezután az `initComponents()` eljárás inicializálja az alkalmazás grafikus komponenseit úgy, mint az ablakot, a paneleket, a gombokat, etc. Az `initListeners()` metódus a szükséges komponensekhez `Listener` objektumokat rendel, mely eseményfigyelést tesz lehetővé. A `showLoginDialog()` rutin segítségével megjelenítésre kerül a bejelentkezési párbeszédpanel. Végül egyéb járulékos inicializációs lépések következnek, mint például bejelentkezési képernyőt követő ablaknak a képernyő közepére történő helyezése.

3.1.4. A `db` csomag

Ahogy már a neve is utal rá ez a csomag, pontosabban az ebben definiált `DBManager` osztály felel az alkalmazás és a HSQL adatbázis közti kapcsolatért, valamint a lekérdezések, módosítások, ill. törlések végrehajtásáért. Szintén ebben a csomagban kerültek definiálásra az adatbázistábláknak megfelelő java osztályok:

- `Ingredient`,
- `ProductIngredient`,
- `Product`,
- `User`,
- `Message`.

3.1.5 Az `Ingredient` osztály

Ez a `products_ingredients` táblának megfelelő típus reprezentálja a pékség által előállított termékek gyártásához szükséges alapanyagokat.

```
public class Ingredient
{
        private String name;
        private String unit;
        private int    calorie;
        private double quantity;
        private int    initialCost;
        private int    minLimit;

        public Ingredient(String name, String unit, int calorie, double quantity,
                          int initailCost, int minLimit)


        public int    getCalorie()
        public void   setCalorie(int calorie)
        public int    getInitialCost()
        public void   setInitialCost(int initialCost)
        public int    getMinLimit()
        public void   setMinLimit(int minLimit)
        public String getName()
        public void   setName(String name)
        public double getQuantity()
        public void   setQuantity(double quantity)
        public String getUnit()
        public void   setUnit(String unit)
}
```

Az attribútumok

- `name`

  Az alapanyag nevét reprezentáló karakterlánc.

- `unit`

  Az alapanyag raktárban tárolt mértékegységét adja.

- `calorie`

  Az alapanyag energiatartalmát reprezentálja kilokalóriában.

- `quantity`

  Az alapanyag raktárban található, mértékegység szerinti pillanatnyi mennyiségét adja.

- `initialCost`

  Az alapanag beszerzési ára magyar forintban.

- `minLimit`

  Az a mértékegység szerinti minimális mennyiség, ami egy pékség egy napi működéséhez szükséges az adott alapanyagból.

Mivel mindegyik adattag `private` módosítóval rendelkezik, és így csak az osztályon belülről látható, beállító (`set`) ill. lekérdező (`get`) metódusok is implementálásra kerültek hozzájuk.

A konstruktor

Az osztály egyetlen, publikus konstruktorral rendelkezik. Példányosítás során az adattagok egyenként inicializásásra kerülnek.

3.1.6. A ProductIngredient osztály

Ez az osztály reprezentálja a konkrét termékösszetevőket egy adott termék esetén. Ahogy a neve is mutatja a `products_ingredients` táblának a java oldali megfelelője.

```
public class ProductIngredient
{
        private String name;
        private String unit;
        private int quantity;

        public ProductIngredient(String name, String unit, int quantity)

        public String getName()
        public void   setName(String name)
        public int    getQuantity()
        public void   setQuantity(int quantity)
        public String getUnit()
        public void   setUnit(String unit)
        public String toString()

}
```

Az attribútumok

- `name`

  A termékösszetevő nevét reprezentáló karakterlánc.

- `unit`

  Az adott termékhez előállításához szükséges összetevő mértékegységét adja.

- `quantity`

  Az adott termékhez előállításához szükséges összetevő mértékegység szerinti mennyisége.

Az előző osztályhoz hasonlóan az adattagok itt is a beállító ill. lekérdező metódusokon keresztül érhetők el.

A konstruktor

Az osztály egyetlen konstruktorral rendelkezik, melynek láthatósága publikus. Példányosítás során minden adattag inicializásásra kerül.

3.1.7. A `Product` osztály

Ez a `products` táblának megfelelő objektumtípus reprezentálja a pékség által előállított termékeket.

```
public class Product
{
        private String name;
        private String group;
        private String unit;
        private int    calorie;
        private int    quantity;
        private int    primeCost;
        private int    sellPrice;
        private Vector<ProductIngredient> ingredients;

        public Product(String name, String group, String unit,
                       int calorie, int quantity, int primeCost,
                       int sellPrice, Vector<ProductIngredient> ingredients)


        public int    getCalorie()
        public void   setCalorie(int calorie)
        public int    getPrimeCost()
        public void   setPrimeCost(int primeCost)
        public int    getSellPrice()
        public void   setSellPrice(int sellPrice)
        public String getName()
        public void   setName(String name)
        public int    getQuantity()
        public void   setQuantity(int quantity)
        public String getUnit()
        public void   setUnit(String unit)
        public String getGroup()
        public void   setGroup(String group)
        public void   setIngredients(Vector<ProductIngredient> ingredients)
        public Vector<ProductIngredient> getIngredients()
}
```

Az attribútumok

- `name`

  A termék nevét reprezentáló karakterlánc.
- `group`

  Azt a termékcsoportot reprezentálja, melybe a termék tartozik.
- `unit`

  A termék raktárban tárolt mértékegységét adja.
- `calorie`

  A termék energiatartalmát reprezentálja kilokalóriában.
- `quantity`

  A termék raktárban található, mértékegység szerinti pillanatnyi mennyiségét adja.
- `primeCost`

  A termék előállítási költsége magyar forintban.
- `sellPrice`

  A termék ára, vagyis az a forintösszeg, amennyiért a pékség az árut értékesíti.
- `ingredients`

  A termék összetevőit reprezentáló `ProductIngredient` objektumok kollekciója.

Mindegyik adattag rendelkezik `set` ill. `get` metódussal.

A konstruktor

Az osztály egyetlen konstruktorral rendelkezik, melynek láthatósága publikus. Példányosítás során az adattagok egyenként inicializásásra kerülnek.

3.1.8. A `User` osztály

Ez a típus reprezentálja a konkrét termékösszetevőket egy adott termék esetén. Ahogy a nevéből is következik a `users` táblának a java oldali megfelelője.

```
public class User
{
        private String name;
        private String address;
        private String username;
        private String password;
        private String role;
        private int    status;

        public User(String name, String address, String username,
                    String password, String role, int status)


        public String getRole()
        public void   setRole(String role)
        public String getName()
        public void   setName(String name)
        public int    getStatus()
        public void   setStatus(int status)
        public String getPassword()
        public void   setPassword(String password)
        public String getUsername()
        public void   setUsername(String username)
        public String getAddress()
        public void   setAddress(String address)
}
```

Az attribútumok

- `name`

  A felhasználó nevét reprezentáló karakterlánc.

- `address`

  A felhasználó címét reprezentáló karakterlánc.

- `username`

  A felhasználó belépéshez szükséges azonosítóját reprezentálja.

- `password`

  A felhasználó belépéshez szükséges jelszavát reprezentálja.

- `role`

  A felhasználó beosztását reprezentálja, mely az alkalmazás elérhető funkcióit szabályozza.

Az előző osztályokhoz hasonlóan az adattagok itt is a beállító ill. lekérdező metódusokon keresztül érhetők el.

A konstruktor

Az osztály egyetlen konstruktorral rendelkezik, melynek láthatósága publikus. Példányosítás során minden adattag inicializásásra kerül.

3.1.9. Az `Message` osztály

Ez a `messages` táblának megfelelő osztály reprezentálja a pékség alkalmazottai mint felhasználók közti kommunikáció üzeneteit.

```
public class Message
{
        private User      sender;
        private String    text;
        private Timestamp timestamp;

        public Message(User sender, String text, Timestamp timestamp)


        public User      getSender()
        public void      setSender(User sender)
        public String    getText()
        public void      setText(String text)
        public Timestamp getTimestamp()
        public void      setTimestamp(Timestamp timestamp)
}
```

Az attribútumok

- `sender`

  Az üzenetet feladó felhasználót reprezentáló referencia.

- `text`

  Az üzenet szövege.

- `timestamp`

  Az üzenetküldés időpontját reprezentáló időbélyeg.

A `get` és `set` metódusok funkciója megegyezik a fentebb ismertetettekével.

A konstruktor

A konstruktor pusztán az adattagok inicializálását végzi paraméterei segítségével.

3.1.10. A `DBManager` osztály

Az osztály implementálása során fontosnak tartottam, hogy – egyfajta átjáróként működve a java objektumorientált ill. az SQL relációs világa között – az alkalmazás többi része számára rejtve maradjanak a relációs adatbázistáblák közti kapcsolatok.

Az attribútumok

Az osztály két (statikus) adattaggal rendelkezik:

```
private static Connection conn;
public static boolean insertSucceeded;
```

A `conn` referencia az adatbáziskapcsolatot reprezentálja, míg az `insertSucceeded` logikai változó a legutóbb végrehajtott `insert` utasítás sikerességéről ad információt.

A konstruktor

Az osztály egyetlen implicit (paraméter nélküli) konstruktorral rendelkezik, ugyanis az osztály `funkcióihoz` statikus metódusain keresztül férünk hozzá, így példányosításra nincs szükség. A továbbiakban tekintsük át az osztály által implementált fontosabb metódusokat!

A connect() metódus

```
public static void connect()
{
    conn = null;
    try {
        Class.forName("org.hsqldb.jdbcDriver");
        conn = DriverManager.getConnection("jdbc:hsqldb:hsql://localhost/edzocipo",
                                           "sa", "");
        conn.setAutoCommit(false);
        } catch(ClassNotFoundException e) {
            System.err.println("Unable to load JDBC driver");
            System.exit(0);
        } catch(SQLException e) {
            printSQLException(e);
            disconnect();
            System.exit(0);
        }
}
```

A metódus a szükséges jdbc-driver betöltése után a megfelelő adatbázis-URL segítségével megpróbálja a `DriverManager`-től kapott `Connection` példányt értékül adni a `conn` attribútumnak, majd kikapcsolja az adatbázis autocommit funkcióját. Amennyiben a fenti műveletek során kivétel keletkezik (például hiányzó jdbc driver esetén), az hibaüzenetek kíséretében az alkalmazás befejeződésével jár.

A disconnect() metódus

```
public static void disconnect()
{
    if (conn != null)
        try { conn.close(); } catch(SQLException e) {}
}
```

Ez az előző metódussal ellentétben épp az adatbázis-kapcsolat lezárását végzi., amennyiben a kapcsolat fennállt.

A `verifyLogin()` metódus

```
public static String verifyLogin(String username, String password)
{
    String retValue = null;
    try
    {
        PreparedStatement ps = conn.prepareStatement("SELECT role FROM " +
              "users WHERE username = ? AND password = ? WHERE status = 1");
        ps.setString(1, username);
        ps.setString(2, password);
        ResultSet rs = ps.executeQuery();
        if (rs.next())
            retValue = rs.getString(1);
        ps.close();
    } catch(SQLException e) {
            printSQLException(e);
            if (conn != null)
                try { conn.rollback(); } catch(SQLException ex) {}
    }
    finally {
        return retValue;
    }
}
```

Ez a függvény végzi el az alkalmazásba való bejelentkezéshez szükséges autentikációt, mely során egy `PrepareStatement` objektum, valamint a paraméterként megkapott felhasználói név ill. jelszó segítségével lekérdezi az adott aktív státuszú felhasználó beosztását a `users` táblából. Majd, ha az eredményhalmaz nem üres, azaz egy autentikált felhasználó nevét ill. jelszavát kaptuk meg paraméterként, egy `String` példányban visszaadja a beosztást. Üres eredményhalmaz esetén `null`-t, ad vissza, ami az autentikáció sikertelenségét jelzi.

A `select` metódusok

Az osztályban minden főbb adatbázistáblához definiálásra került egy vagy több lekérdező metódus, mely az adott tábla esetlegesen projektált sorait szolgáltatja:

- `selectUsers()`
- `selectIngredients()`
- `selectProducts()`
- `selectMessages()`
- `selectProductGroups()`
- `selectIngredientUnits()`

- `selectUserRoles()`

A `products_ingredients` táblához azért nincs külön lekérdező rutin, mert ezen tábla soraira csak egy adott termék ismeretében van szükség, és ezt a funkciót a `selectProducts()` függvény is ellátja. Ezen metódusok működésüket tekintve nagyon hasonlóak a már ismertetett `verifyLogin()` rutinhoz, így az egyenkénti bemutatásuktól eltekinthetünk. Mégis érdekes lehet a következő metódus megtekintése.

A `selectProducts()` metódus

```
public static Vector<Product> selectProducts()
{
    Vector<Product> retValue = null;
    try{
        PreparedStatement ps = conn.prepareStatement("SELECT * FROM " +
                                                     "products ORDER BY name");
        ResultSet rs = ps.executeQuery();
        retValue = new Vector<Product>();
        while (rs.next()){
            Product p = new Product(rs.getString(1), rs.getString(2), rs.getString(3),
                                    rs.getInt(4), rs.getInt(5), rs.getInt(6),
                                    rs.getInt(7), (Vector<ProductIngredient>)null);
            PreparedStatement ps_;
            ps_ = conn.prepareStatement("SELECT ingredient_name,unit,quantity FROM " +
              "products_ingredients WHERE product_name = ? ORDER BY ingredient_name");
            ps_.setString(1, rs.getString(1));
            ResultSet rs_ = ps_.executeQuery();
            Vector<ProductIngredient> ingredients = new Vector<ProductIngredient>();
            while (rs_.next())
                ingredients.add(new ProductIngredient(rs_.getString(1),
                                rs_.getString(2), rs_.getInt(3)));
            p.setIngredients(ingredients);
            retValue.add(p);
        }
        ps.close();
    }catch(SQLException e) {
        printSQLException(e);
        if (conn != null)
            try { conn.rollback(); } catch(SQLException ex) {}
    }
    finally {
        return retValue.size()==0?null:retValue;}
}
```

A többi lekérdező metódushoz hasonlóan itt is a szóban forgó tábla szelekciójával indul a kód, azonban a már említett objektum-relációs átjáró funkciót betöltendő ezzel nem zárul le az SQL-lekérdezések sora, ugyanis a `Product` ill. `Ingredient` objektumok között 1:N kapcsolat áll fent. Így minden egyes eredményhalmazbeli elem egy újabb lekérdezést

indukál a products_ingredients kapcsolótáblán, mely a products ill ingredients táblák közti kapcsolatot reprezentálja. Ennek a belső lekérdezésnek az eredményeként előálló ResultSet halmaznak az elemei kerülnek beszúrásra az aktuálisan példányosított Product objektum termékösszetevőit reprezentáló, ProductIngredient elemeket tartalmazó dinamikus tömbbe. Végül az imént létrejött Product objektumot beszúrjuk abba a kollekcióba, mely a függvény visszatérési értékét szolgáltatja.

A selectProductGroups, selectIngredientUnits ill. selectUserRoles rutinok annyiban különböznek a selectProducts , selectIngredients ill. selectUsers függvényektől, hogy a szelektált soroknak csak egy projekció utáni eredményét szolgáltatják, mely projekciók rendre a group_name, unit ill. role oszlopokra vonatkoznak.

Az insert metódusok

A select függvényekhez hasonlóan ebbe a csoportba is négy függvény tartozik:

- insertIntoUsers
- insertIntoIngredients
- insertIntoProducts
- insertIntoMessages

Ezen eljárások funkciója nem más, mint a paraméterként megkapott objektum adattagjainak adott táblába (táblákba) történő beszúrása. Az előző metóduscsoporttal analóg módon itt is kimaradt a products_ingredients táblára vonatkozó beszúrás explicit implementálása, mely a select metódusnál leírtakhoz hasonlóan indokolható. Tekintsük ismét a products táblára vonatkozó eljárást!

```java
public static void insertIntoProducts(Product newProduct)
{
    insertSucceeded = true;
    try{
        PreparedStatement ps = conn.prepareStatement("INSERT INTO " +
                                "products VALUES (?,?,?,?,?,?,?)");
        ps.setString(1, newProduct.getName());
        ps.setString(2, newProduct.getGroup());
        ps.setString(3, newProduct.getUnit());
        ps.setInt(4,    newProduct.getCalorie());
        ps.setInt(5,  newProduct.getQuantity());
        ps.setInt(6,    newProduct.getPrimeCost());
        ps.setInt(7,    newProduct.getSellPrice());
        ps.executeUpdate();
        ps.close();
        for (ProductIngredient ingred : newProduct.getIngredients()) {
            ps = conn.prepareStatement("INSERT INTO products_ingredients " +
                                "VALUES (?,?,?,?)");
            ps.setString(1, newProduct.getName());
            ps.setString(2, ingred.getName());
            ps.setString(3, ingred.getUnit());
            ps.setInt(4,    ingred.getQuantity());
            ps.executeUpdate();
            ps.close();
        }
        conn.commit();
    }catch(SQLException e) {
        insertSucceeded = false;
        printSQLException(e);
        if (conn != null)
            try { conn.rollback(); } catch(SQLException ex) {}
    }
}
```

A lekérdező metódusnál leírtakkal analóg módon itt is beszélhetünk külső ill. belső `insert` utasításokról. Természetesen a külső beszúrás ismét a `products` táblára, míg a belsők a `products_ingredients` táblára vonatkoznak. Azonban most a belső `insert` utasítások száma nem a külső eredményhalmazának számosságától függ –, hiszen update műveletek esetén az nem kerül feldolgozásra –, hanem a beszúrandó új `Product` objektum mint termék összetevőinek a számától. Így minden egyes `ProductIngredient` példány beszúrásra kerül a `products_ingredients` kapcsolótáblába. Amennyiben a metódus futása közben `SQLException` kivétel váltódik ki –, mely például egy elsődleges kulcs megszorítás megsértése esetén adódhat –, az addig végrehajtott összes adatbázis-művelet visszagörgetésre kerül(, ha egyáltalán fennállt a kapcsolat), valamint a művelet sikerességét jelző, kezdetben igazra állított `insertSucceeded` változó is hamis értéket kap.

Az `delete` metódusok

Egyfajta naplózási funkciót betöltendő a `Messages` táblához nem volt szükséges sortörlő rutin implementálása, így három metódus tartozik ebbe a csoportba:

- `deleteFromUsers`
- `deleteFromIngredients`
- `deleteFromProducts`

Ezek a metódusok a paraméterként megkapott sort törlik az adott táblából. Az előzőekhez hasonló okokból itt is elmaradt a `products_ingredients` táblára vonatkozó törlés explicit implementálása. Nézzük azonban a `products` táblára vonatkozó eljárást!

```
public static void deleteFromProducts(String productName)
{
    try
    {
        PreparedStatement ps = conn.prepareStatement("DELETE FROM " +
                "products_ingredients WHERE product_name = ?");
        ps.setString(1, productName);
        ps.executeUpdate();
        ps.close();
        ps = conn.prepareStatement("DELETE FROM products WHERE name = ?");
        ps.setString(1, productName);
        ps.executeUpdate();
        ps.close();
        conn.commit();
    }catch(SQLException e) {
        printSQLException(e);
        if (conn != null)
            try { conn.rollback(); } catch(SQLException ex) {}
    }
}
```

Törlés esetén, ellentétben a lekérdezéssel ill. beszúrással –, hacsak nem váltódik ki kivétel –, mindig pontosan két adatbázis-művelet hajtódik végre. Az első vonatkozik a `products_ingredients` kapcsolótáblára, míg a második a `products` táblából töröl. Ez a sorrend kötött, ugyanis a tábla `product_name` oszlopán lévő külső kulcs megszorítás miatt minden egyes `products_ingredients` sorhoz léteznie kell egy `products` táblabeli sornak.

A `printSQLException` metódus

Ez a rutin a programfejlesztés során egyfajta debug funkciót látott el, azonban az esetleges hibák kijavítását megkönnyítendő az alkalmazás végleges verziójában is helyet kapott.

```
public static void printSQLException(SQLException e)
{
    if (e != null)
    {
            System.err.println("*** SQLException caught: ");
            while (e != null)
            {
                    System.err.println("SQLState: " + e.getSQLState());
                    System.err.println("Message: " + e.getMessage());
                    System.err.println("Vendor: " + e.getErrorCode());
                    e = e.getNextException();
            }
    }
}
```

Az eljárás az egyes adatbázis-műveletek végrehajtása közben esetlegesen bekövetkező kivételek esetén kerül meghívásra, így minden `try-catch` blokk `catch` ágban található egy printSQLException hívás. A metódus működése nem áll másból, mint a paraméterként kapott `SQLException` példány informatív adattagjainak a kiírásából.

3.2. Az adatbázistáblák leírása

Az alkalmazásban öt adatbázistáblát használunk. Ezekben a táblákban fogjuk tárolni a felhasználói adatokat, alapanyagok adatait, termékek adatait, termékek alapanyagait és az üzenőfalon megjelenő üzeneteket. Ezeket a táblákat létre kell hozni az adatbázisban az alkalmazás telepítésekor. Az SQL scriptek futtatására megfelelő eszköz lehet a hsqldb.jar-ban található grafikus adatbázis kezelő felület, melyet a CD mellékleten lévő Futtatható/runDatabaseManager.bat parancs fájlal indíthatunk el. Az összes SQL script megtalálható a CD melléklet Forráskódok/dbscripts/ mappában. A következő részben részletesen leírom a táblák szerkezetét.

3.2.1. Felhasználók tábla

Ebben a táblában tárolja az alkalmazás a felhasználók adatait. Az elsődleges kulcs a `username` ami a felhasználók azonosítóját jelenti. Ennek egyedinek kell lennie minden felhasználó esetén. Ezen kívül minden felhasználóhoz tároljuk a teljes nevét a `name`

mezőben, a címét az address mezőben. A felhasználók jelszavát a password mezőben tároljuk, ami jelenleg, mint az eddig említett mezők, VARCHAR típusú, de biztonsági okokból az alkalmazás továbbfejlesztésénél valamilyen titkosított mezőben lenne ajánlott tárolni. Ezeken kívül még eltároljuk a felhasználó jogait a role mezőben és a státuszát a status mezőben. A status mező logikai értéket tárol, INTEGER típussal hoztam létre. Ha a status mező értéke 1, akkor a felhasználó aktív, egyébként passzív.

Magát a táblát az alábbi SQL utasítással hoztam létre az adatbázisban:

```
CREATE TABLE users (
    name        VARCHAR(20),
    address     VARCHAR(40),
    username    VARCHAR(20) PRIMARY KEY,
    password    VARCHAR(20),
    role        VARCHAR(10),
    status      INTEGER
);
```

3.2.2. Alapanyagok tábla

Ebben a táblában tárolom a termékek gyártása során felhasználható alapanyagokat és azok mennyiségét a raktárban. A name mező lesz az elsődleges kulcs, melyben az alapanyagok nevét tárolom. A unit mezőben adom meg az alapanyag mértékegységét. Jelenleg a kg, liter és a darab szerepel az adatbázisban. Ez később is bővíthető az alkalmazás átírása átírása nélkül. A calorie mezőben az alapanyag egységének megfelelő kalória értéket tárolom. A quantity mezőben tárolom az aktuális mennyiséget, ami a raktáron található az adott alapanyagból. Az initial_cost mezőbe kerül az alapanyag beszerzési ára. A min_limit pedig azt a mennyiséget adja meg, ami alatt már a hiányjelző funkciónak jeleznie kell. Az utóbbi két mezőbe egész értékek kerülhetnek, ezért ezekhez INTEGER típust adtam meg. Az aktuális mennyiség nem csak egész lehet, ezért annak típusa DOUBLE lett. A többi mező pedig szöveges, ezért VARCHAR típussal hoztam létre.

Magát a táblát az alábbi SQL utasítással hoztam létre az adatbázisban:

```
CREATE TABLE ingredients (
    name        VARCHAR(20) PRIMARY KEY,
    unit        VARCHAR(10),
```

```
    calorie      INTEGER,
    quantity     DOUBLE,
    initial_cost INTEGER,
    min_limit    INTEGER
);
```

3.2.3. Termékek tábla

A termékek táblában tároljuk meg a pékségben előállított termékeket. Az elsődleges kulcs a `name` mező, azaz a termék neve lesz. A `group_name` mezőben tárolom el azt a termékcsoportot, melybe az adott termék tartozik. A `unit` mezőben tárolom a termék mértékegységét, a `calorie` mezőben a termék egységében lévő kalória mennyiségét. A quantity mezőbe kerül a raktáron lévő aktuális mennyiség. A `prime_cost` mezőben tárolom az előállítási költséget, míg a `sell_price` mezőben az eladási árat. Az egyes termékek esetén nem ebben a táblában tárolom el az alapanyagokat, mivel egy terméknek több alapanyaga is lehet. Erre a `products_ingredients` tábla fog megoldást szolgáltatni, mely az alapanyagok és a termékek táblát kapcsolja össze.

Magát a táblát az alábbi SQL utasítással hoztam létre az adatbázisban:

```
CREATE TABLE products (
    name         VARCHAR(20) PRIMARY KEY,
    group_name   VARCHAR(20),
    unit         VARCHAR(10),
    calorie      INTEGER,
    quantity     INTEGER,
    prime_cost   INTEGER,
    sell_price   INTEGER
);
```

3.2.4. Termékek - Alapanyagok kapcsoló tábla

Ennek a táblának a segítségével adom meg az egyes termékek esetén azokat az alapanyagokat, melyekből készül. Ennek a táblának az elsődleges kulcsa a `product_name` és a `ingredient_name` mezők lesznek együtt. Ezek külső kulcsok is. Előbbi a termékek táblának elsődleges kulcsára, míg utóbbi az alapanyagok tábla elsődleges kulcsára

hivatkozik. Ezen kívül a táblában még eltárolom az aktuális termék esetén az aktuális alapanyagra vonatkozóan a mértékegységet a `unit` mezőben és a mennyiséget a `quantity` mezőben.

Magát a táblát az alábbi SQL utasítással hoztam létre az adatbázisban:

```
CREATE TABLE products_ingredients (
    product_name    VARCHAR(20),
    ingredient_name VARCHAR(20),
    unit            VARCHAR(10),
    quantity        INTEGER,
    PRIMARY KEY     (product_name,ingredient_name),
    FOREIGN KEY     (product_name)        REFERENCES
    products    (name),
    FOREIGN KEY     (ingredient_name)   REFERENCES
    ingredients (name)
);
```

3.2.5. Üzenetek tábla

Ebben a táblában tárolom az üzenőfalra kikerülő üzeneteket. A `sender` mező és `text` mező együttesen alkotja az elsődleges kulcsot. Előbbiben az üzenet küldőjét, míg utóbbiban az üzenetet szövegét adom meg. Ezen kívül a `sender` mezőbe csak létező felhasználók kerülhetnek, ezért ez a mező külső kulcsként kapcsolódik a felhasználók tábla `username` mezőjéhez. A `timestamp` mezőben az üzenet létrehozásának pontos időpontját tárolom el.

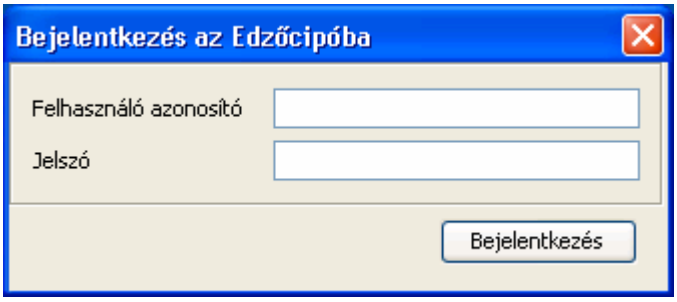
Magát a táblát az alábbi SQL utasítással hoztam létre az adatbázisban:

```
CREATE TABLE messages (
    sender          VARCHAR(20),
    text            VARCHAR(200),
    timestamp       TIMESTAMP,
    PRIMARY KEY     (sender,text),
    FOREIGN KEY     (sender) REFERENCES users
    (username)
);
```

3.3. Az elkészült alkalmazás leírása a felhasználó oldaláról

Az alkalmazás indításakor egy Bejelentkezési ablakkal találkozunk. Ezen megadhatjuk, a felhasználói azonosítót és a jelszót. A bejelentkezés gombra kattintva a program ellenőrzi, hogy a megadott felhasználói azonosító-jelszó páros helyes-e. Ha az autentikáció sikeres, akkor a megadott felhasználó jogosultságának megfelelően jelennek meg az alkalmazás funkciói. Hibás azonosító-jelszó párosról a bejelentkezési ablak tájékoztatja a felhasználót.

3.3.1. Bejelentkező képernyő



12. ábra - Bejelentkezési képernyő

3.3.2. Jogosultságkezelés

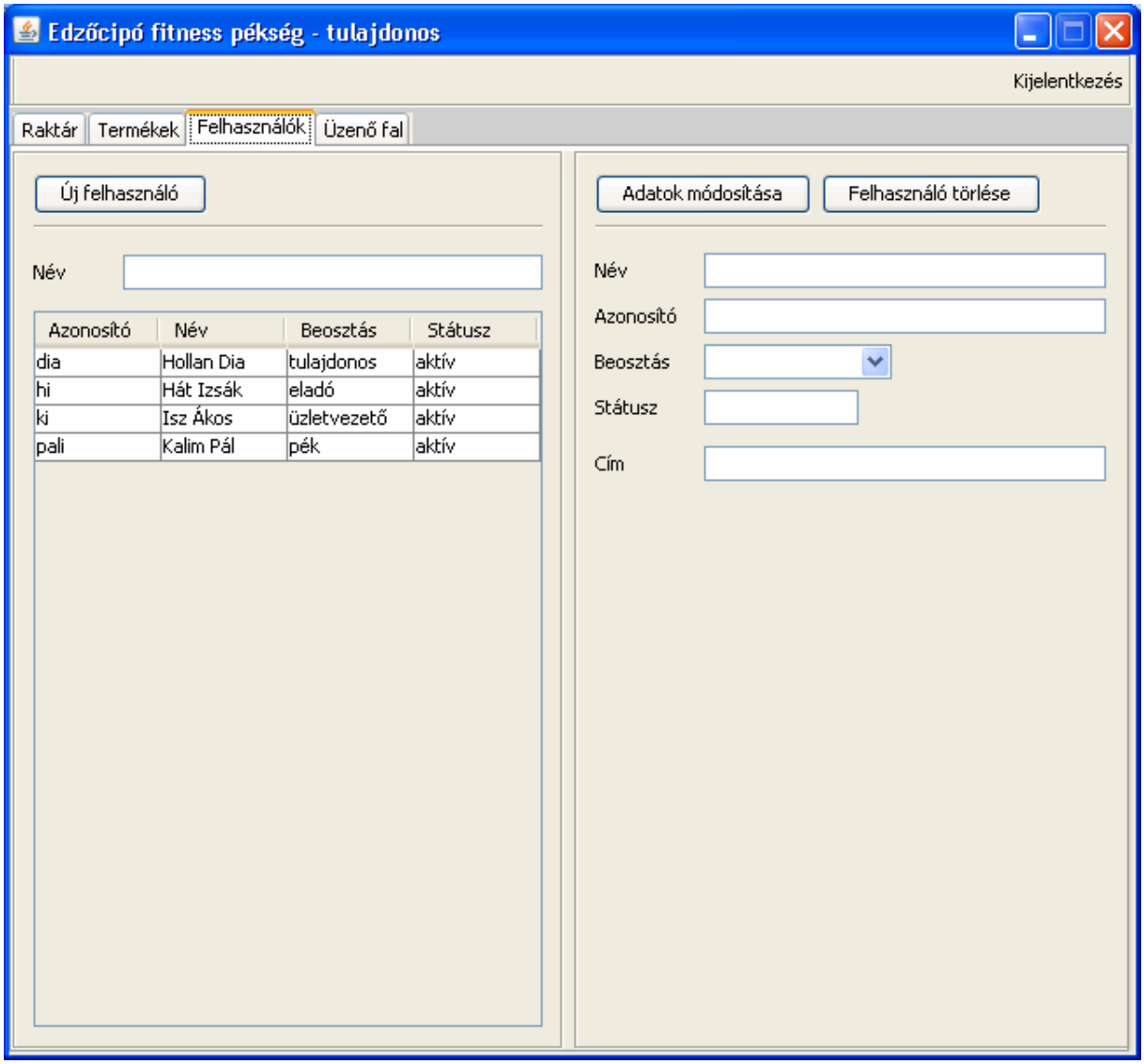
Az alkalmazásban az alábbi beosztások találhatóak:

- Eladó
- Pék
- Tulajdonos

- Üzletvezető

Az Eladó-nak és a Pék-nek ugyan azokhoz a funkciókhoz van hozzáférése: elérheti a Raktár, Termékek és Üzenőfal funkciókat. A Tulajdonos és az Üzletvezető beosztású felhasználók ezen kívül még elérik a Felhasználók funkciót is, ahol a rendszerben lévő felhasználók adatait tudják módosítani, új felhasználókat felvenni, törölni.

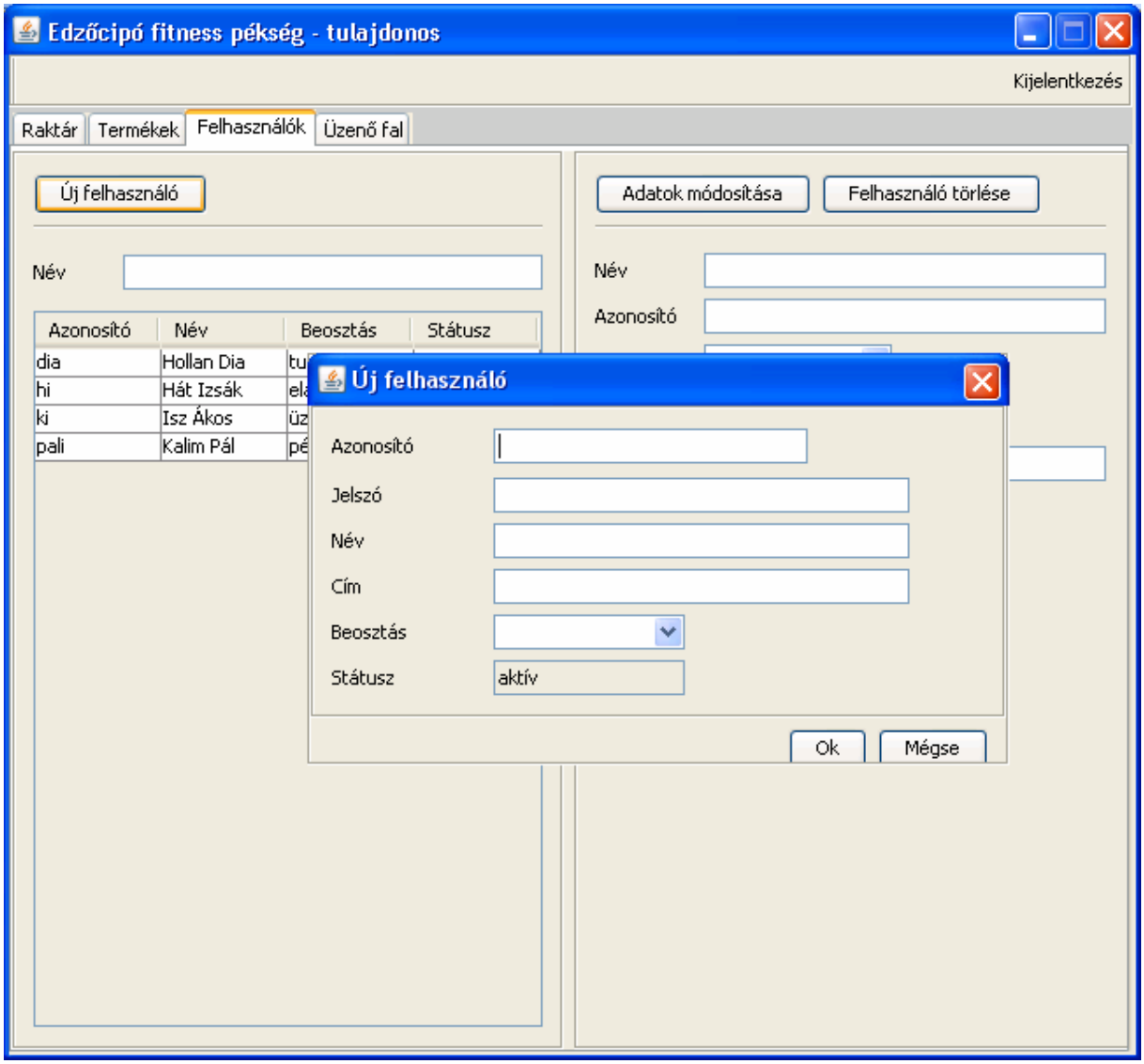
Felhasználók kezelése:



13. ábra - Felhasználók listázása

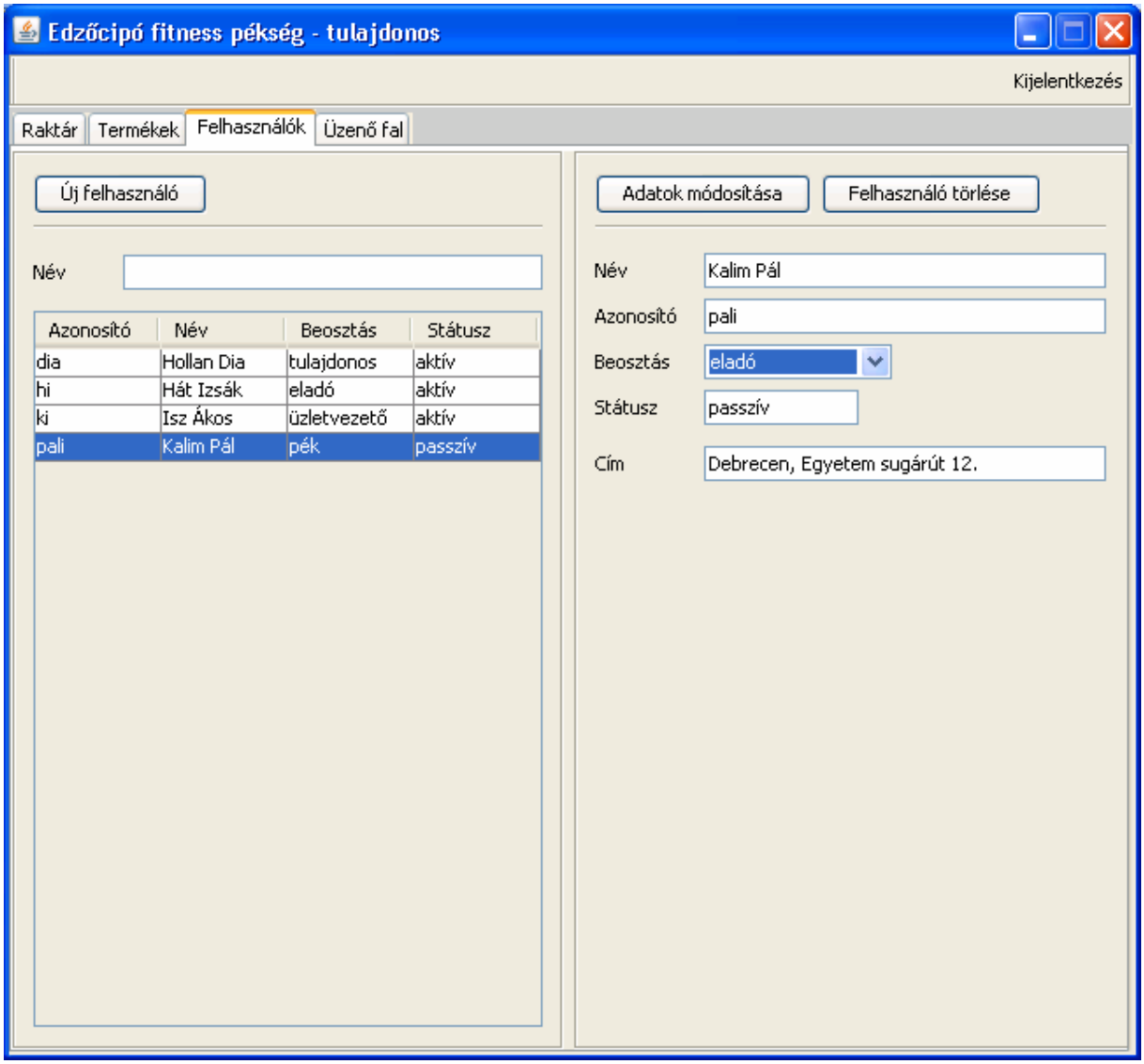
Ezen a fülön a Tulajdonos és az Üzletvezető beosztású felhasználók láthatják a rendszerben lévő összes felhasználót. Létrehozhatnak új felhasználókat az Új felhasználó gombra

kattintva. Ekkor egy új ablak jelenik meg, ahol megadhatjuk az új felhasználó adatait (Azonosító, Jelszó, Név, Cím, Beosztás):



14. ábra - Új felhasználó felvétele

A jelenlegi felhasználók adatai is módosíthatóak. A felhasználóra kattintva a jobb oldalsó panelban módosítható az összes adata, kivéve a jelszava. Lehetőségünk van a felhasználókat törölni is, de ha nem akarjuk törölni, csak átmenetileg vagy véglegesen letiltani, akkor erre használható a státus átírása. Az alkalmazás csak az aktív felhasználókat engedi belépni. Ha az adatokat módosítottuk, akkor az Adatok módosítása gomb megnyomásával tudjuk ezt a módosítást véglegesíteni.



15. ábra - Felhasználó adatainak módosítása

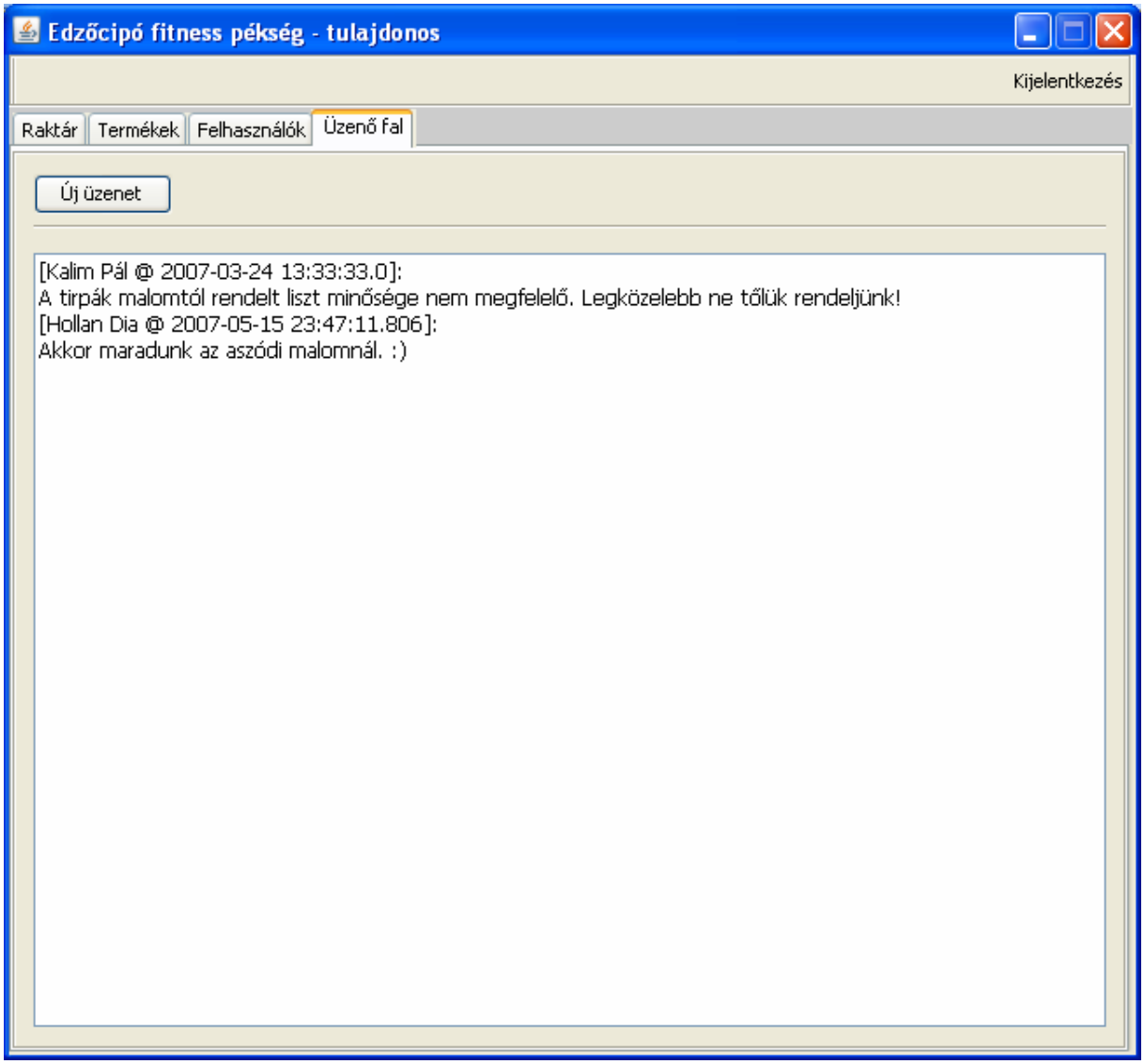
Felhasználót pedig úgy tudunk törölni, hogy a bal oldalsó listából kiválasztva a megfelelőt, a Felhasználó törlése gombra kattintunk.

További segítség, hogy ha sok felhasználónk van, akkor a Név mezőbe beírva a nevét vagy annak az elejét, akkor csak azokat a felhasználókat látjuk kilistázva, melyek nevére illeszkedik a beírt szövegrész. Ehhez a szűkítéshez nem kell megnyomni semmilyen gombot, hanem ahogy írjuk, betűnként végzi el a szűrést és frissíti a megjelenített felhasználó listát.

3.3.3. Üzenőfal funkció

Ez a funkció minden felhasználó számára elérhető. Bármilyen közérdekű üzenetet lehet ide elhelyezni, ami nagyban megkönnyíti a felhasználók közötti kommunikációt. Ez a funkció

különösen fontos a pékség esetén, ahol a főnökség és az eladók más munkaidőben dolgoznak, mint a pékek. A használata nagyon egyszerű.

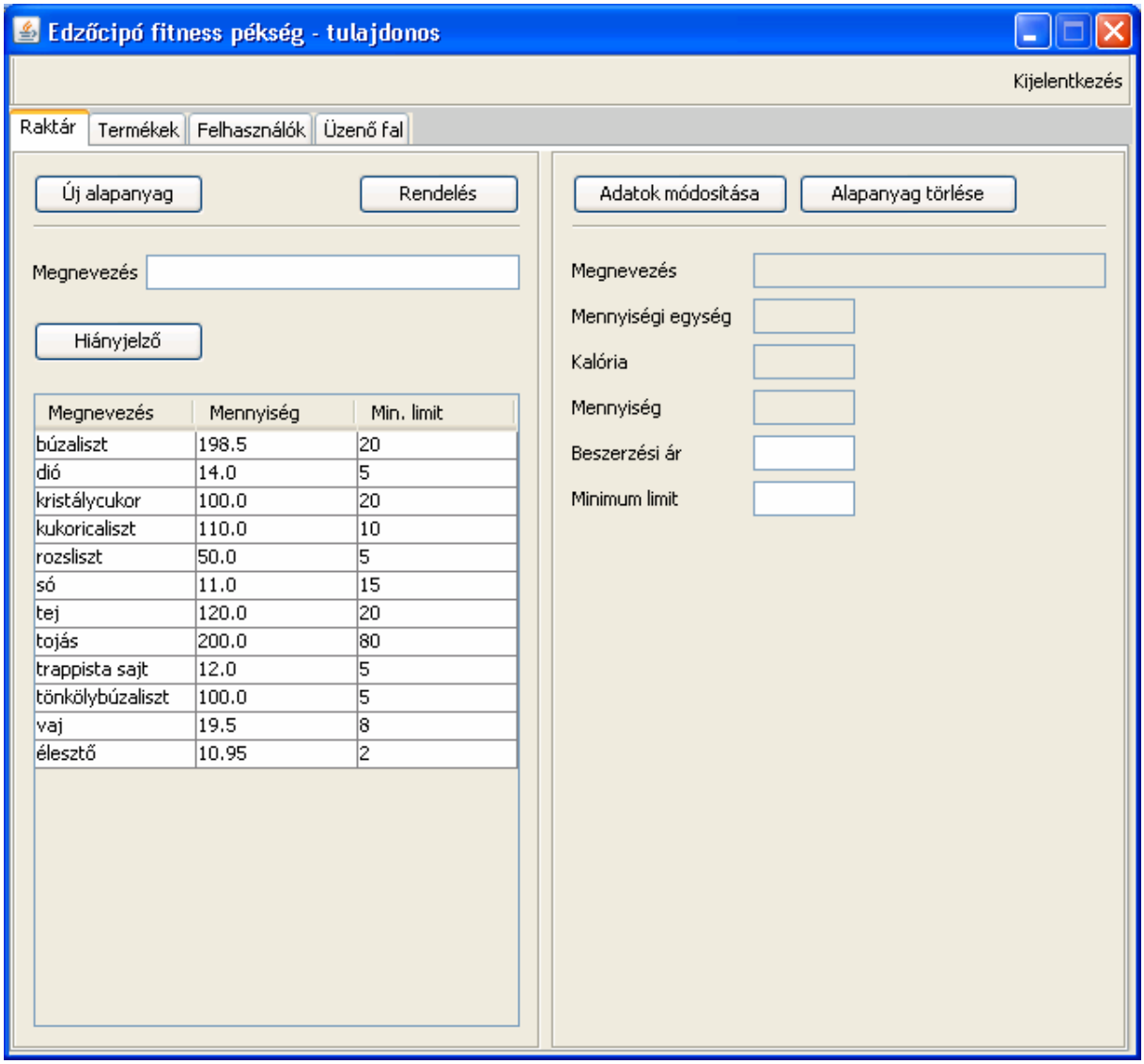


16. ábra - Üzenőfal

Az új üzenet gombra kattintva létrehozhatunk egy új üzenetet. Minden üzenet elé beíródik a létrehozójának a neve és a létrehozás időpontja. Minden felhasználó láthatja az összes üzenetet.

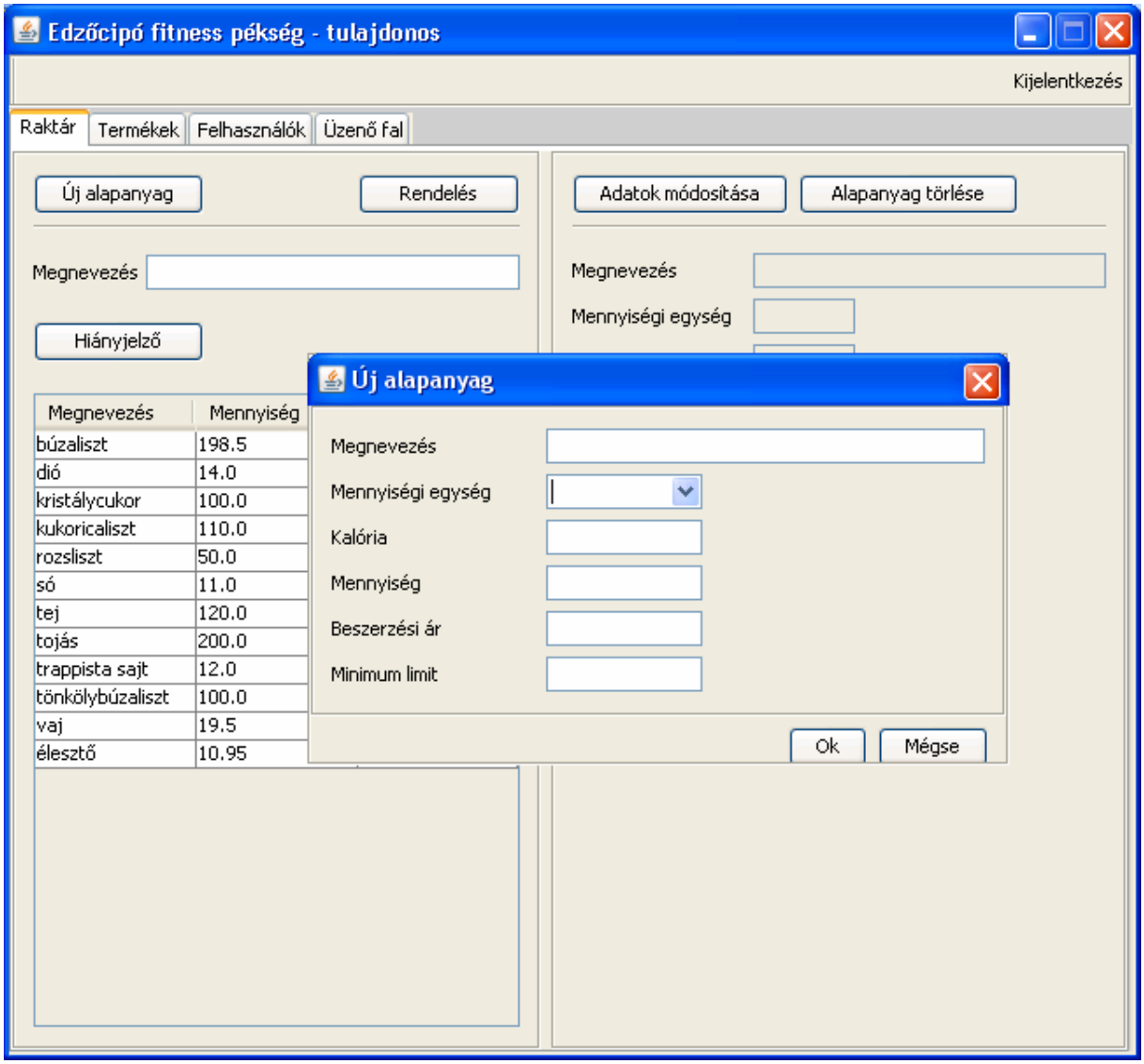
3.3.4. Raktárkezelés

Alapanyagok felvétele, módosítása, törlése a Raktár fülön tehető meg.



17. ábra - Alapanyagok listázása (Raktár)

Bármelyik felhasználóval belépve elérhetjük ezt a funkciót. Bal oldalon láthatjuk felsorolva az összes raktáron lévő alapanyagot és az elérhető mennyiséget az adott alapanyagból. Ezen a fülön is elképzelhető, hogy egyszerre sok alapanyagot kezelünk, ezért itt is van egy, a felhasználóknál már bemutatott, szűrési lehetőség. A Megnevezés melletti mezőbe, ha elkezdünk gépelni, akkor az arra illeszkedő alapanyagokat listázza csak ki. Ha egy új alapanyagot akarunk felvenni, akkor ezt az Új alapanyag gombra kattintva tehetjük meg.



18. ábra - Új alapanyag létrehozása

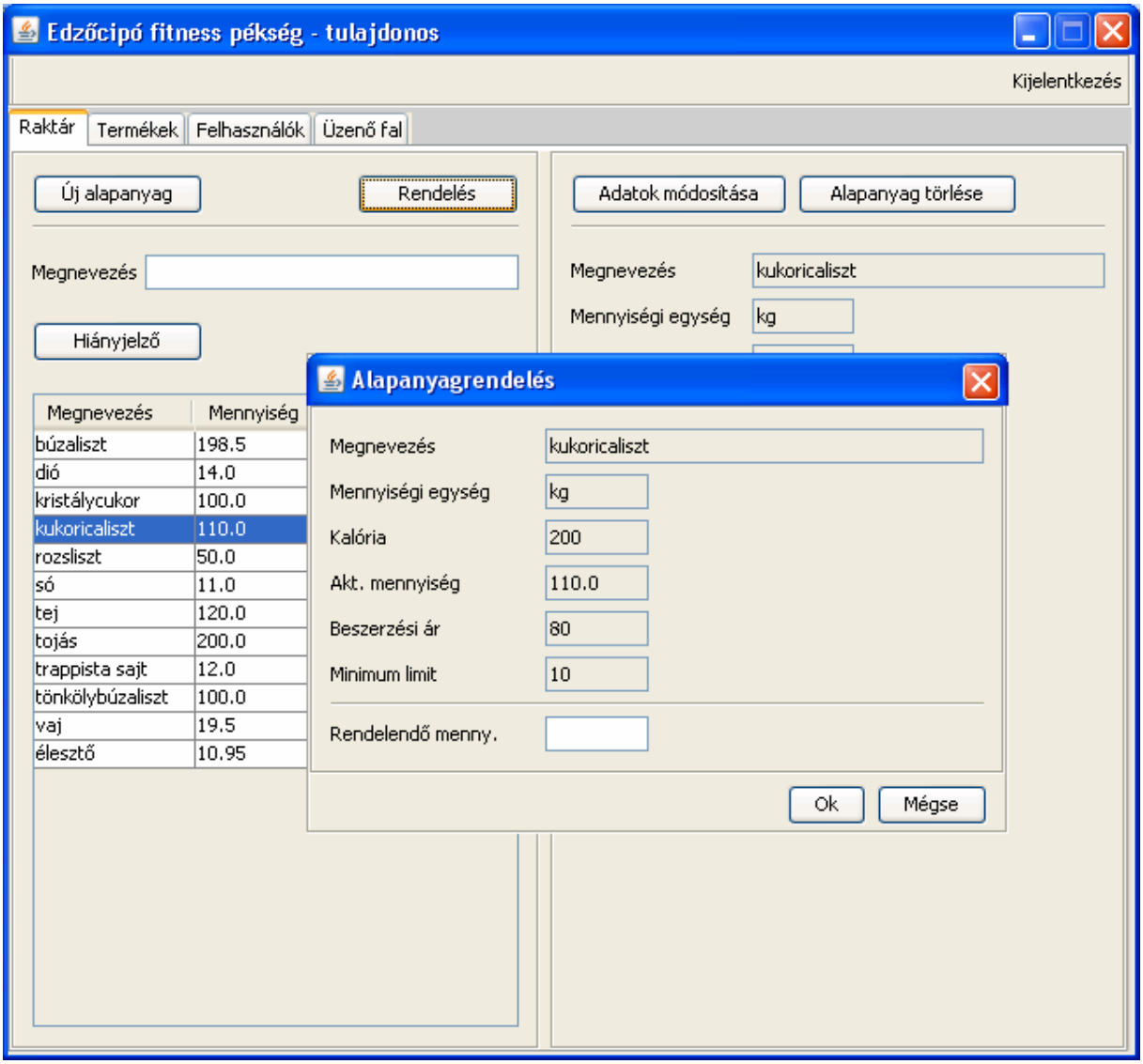
Itt megadhatjuk az új alapanyag megnevezését, kiválaszthatjuk a mennyiség egységét (db, kg, liter). Megadhatjuk a kalóriát, a mennyiséget, a beszerzési árat és a minimum limitet.

Ha arra vagyunk kíváncsiak, hogy melyek azok az alapanyagok, melyekből már a meghatározott limitnél kevesebb van a raktáron, azokat kilistázhatjuk a Hiányjelző gombra kattintva.

Ha egy alapanyagnak módosul a beszerzési ára, vagy meg akarjuk változtatni a minimum limitét, akkor azt úgy tehetjük meg, hogy rákattintunk a kiválasztott alapanyagra, ekkor a jobb oldalon lévő panelba betöltődik az alapanyag összes adata, melyen már módosíthatjuk a beszerzési árat és minimum limitet. Az Adatok módosítása gombra kattintva a módosítások véglegesítődnek.

Egy alapanyagot az Alapanyag törlése funkcióval távolíthatunk el a listából, ha előtte kiválasztottuk a törlendőt.

Ha egy alapanyagból már kevés van, vagy csak esedékes a raktáron lévő mennyiség pótlása, akkor ezt a rendelés funkcióval tehetjük meg. A lenti listából kiválasztjuk az alapanyagot, majd a rendelés gombra kattintva előugrik egy ablak, ahol megadhatjuk, hogy mekkora mennyiséget szeretnénk rendelni.

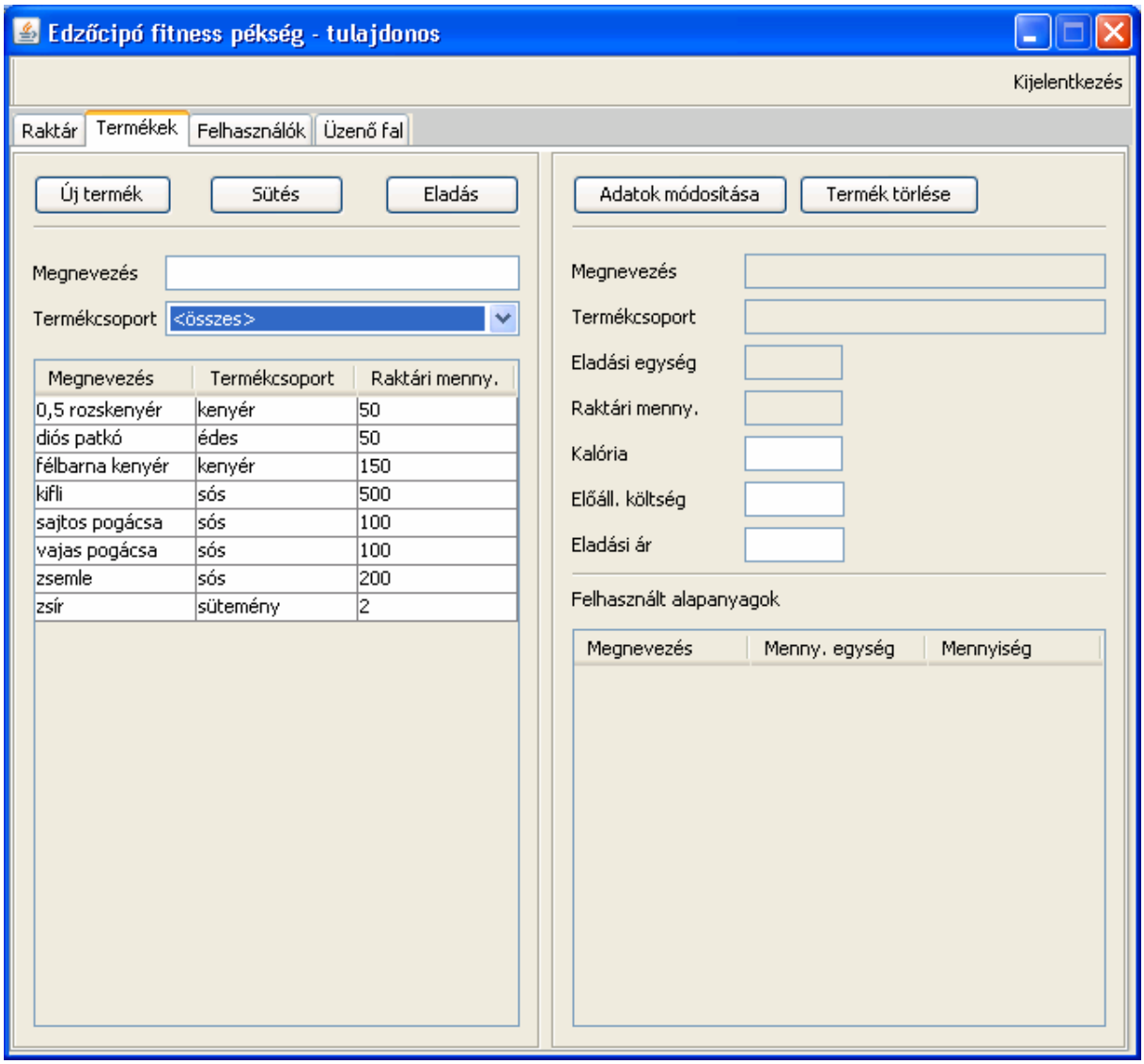


19. ábra - Alapanyag rendelése

Ekkor a rendelt mennyiséggel megnövekszik a raktárban lévő alapanyag mennyisége.

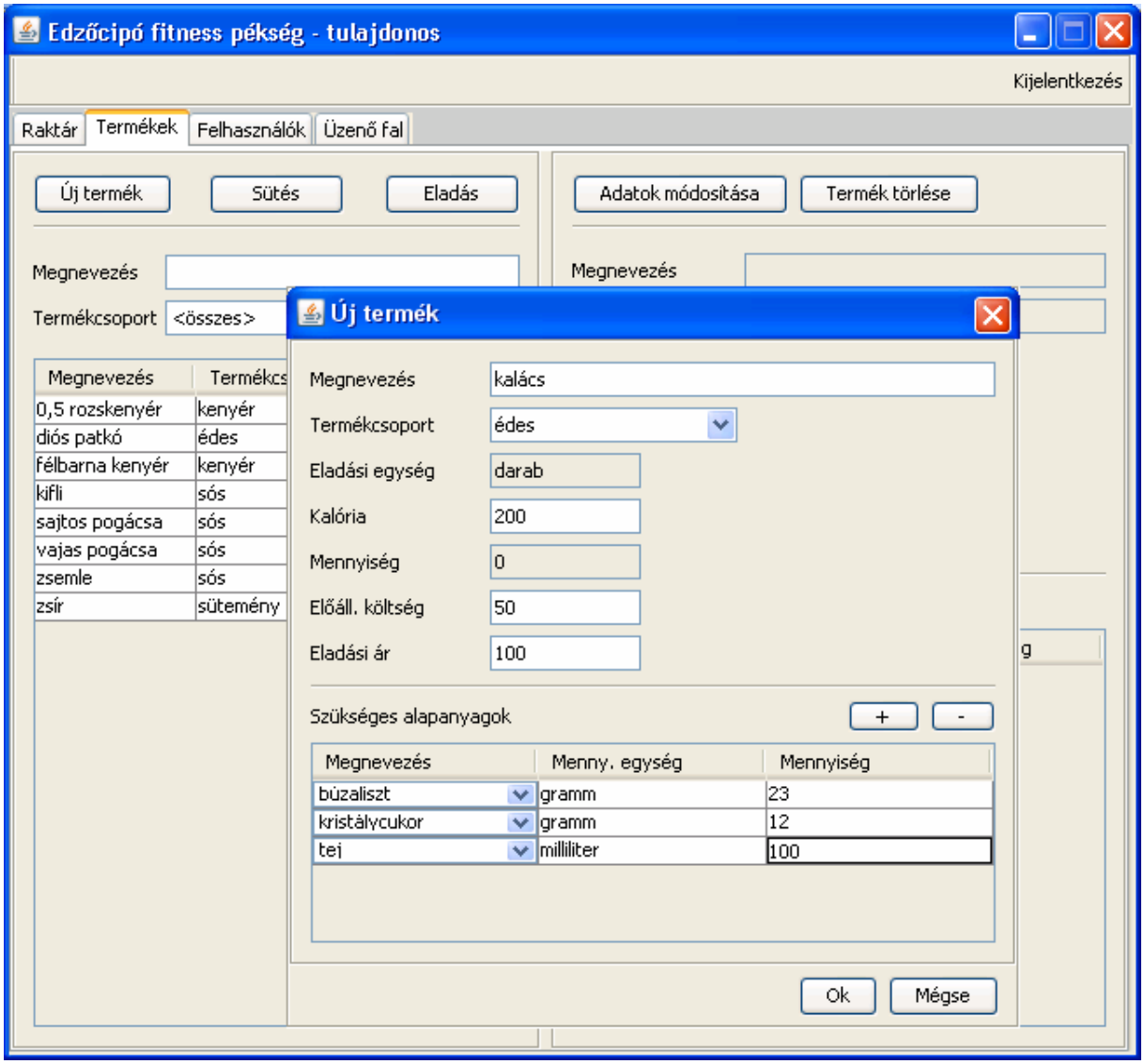
3.3.5. Termékek kezelése

Ezen a fülön találhatóak azok a termékek, amelyeket a pékség előállít. Az egyes termékeket különböző Termékcsoportokba sorolhatjuk. Ezen a fülön is elvégezhetünk szűrést a Megnevezésre vonatkozóan. Továbbá a Termékcsoportot kiválasztva kilistázhatjuk csak az adott termékcsoportba tartozó termékeket is.



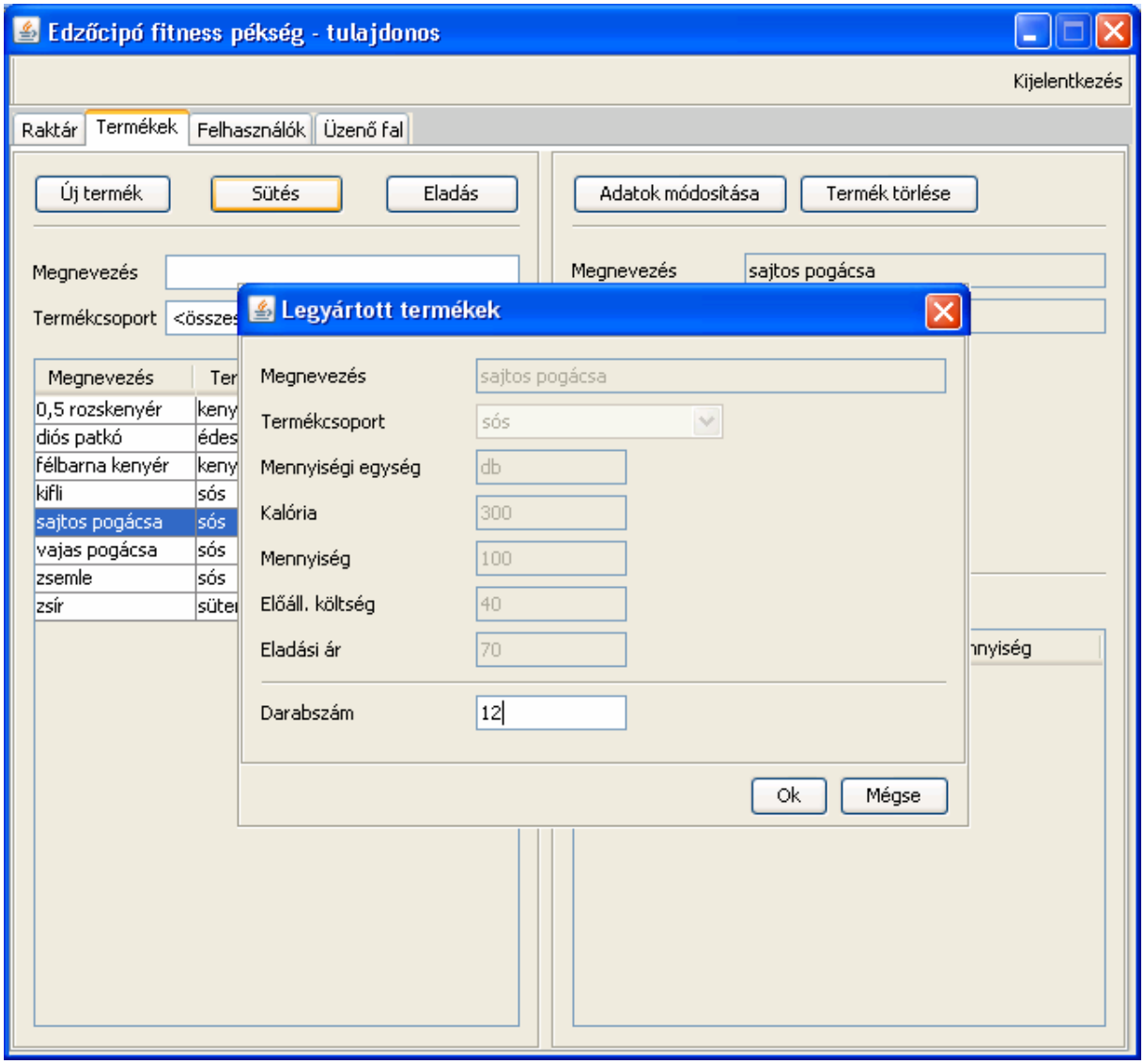
20. ábra - Termékek listázása

Létrehozhatunk új terméket az Új termék gombra kattintva. Ekkor egy felugró ablakban megadhatjuk az új termék megnevezését, azt, hogy melyik termékcsoportba tartozik. Az elkészített termék kalóriáját. Az előállítási költséget, az eladási árat. Továbbá megadhatóak a szükséges alapanyagok és azokhoz a szükséges mennyiség.



21. ábra - Új termék létrehozása

A termékek listájában láthatjuk, hogy aktuálisan mekkora mennyiség áll rendelkezésre a raktárban az adott termékből. Ha egy terméket akarunk készíteni, akkor válasszuk ki a készítendő terméket, majd a Sütés gombra kell kattintani. Ekkor egy ablakban megadhatjuk, hogy hány darabot készítünk az adott termékből.



22. ábra - Termék előállítása (Sütés)

Ha az Ok-ra kattintunk, akkor raktáron lévő mennyiség az adott termékből növekszik a megadott mennyiséggel.

Az eladás ehhez nagyon hasonlóan történik. Ekkor a raktáron lévő mennyiség csökken a megadott mennyiséggel.

Az egyes termékek esetén módosíthatjuk a Kalóriát, az Előállítási költséget, az Eladási árat és a termék hozzávalóinak mennyiségét. Továbbá törölhetjük is a kiválasztott terméket.

4. Összefoglalás

A szakdolgozatom írása során mélyebb betekintést nyerhettem a Java nyelv lehetőségeibe és eszközeibe. Megtapasztaltam egy konkrét tervezési, fejlesztési folyamatot az elejétől a végéig. A fejlesztés során sokszor módosítottam a terveket, mert olyan akadályokba ütköztem, melyeket a terv készítésénél még nem láttam tisztán. Így utólag sokkal jobban átlátom a programot, a követelményrendszert és a funkciókat. Valószínűleg ha most kezdeném, akkor egy sokkal jobban megtervezett és kialakított szoftver fejlesztenék ki.

A fejlesztői környezetet is sokkal jobban megismertem. A funkcióinak nagy részét kipróbáltam és alkalmaztam. Programozóként elhelyezkedve ez a tudásom is előnyömre válhat, mivel a cégek egy részénél ugyanezt a fejlesztő rendszert használják.

Ennek az alkalmazásnak a fejlesztése azért is volt igen hasznos számomra, mivel egyetlen program kifejlesztésében nagyon sok egyetemi tantárgyban tanult elméleti tudást integrálhattam a gyakorlatban. Az Adatbázisrendszerektől kezdve, a Programozás 1 és 2-n keresztül a Rendszerfejlesztés technológiáig.

A közeljövőben tervezem elkészíteni a programnak egy kibővített verzióját, melyre a szakdolgozatom leadási határideje miatt már nem jutott idő, amit az állásinterjúk során referenciaként mutathatnék be.

5. Irodalomjegyzék

Vég Csaba, Juhász István: Java-start!, Logos 2000, Debrecen, 1999.

Nyékyné Gaizler Judit: Java 2, Útikalauz programozóknak 1.3, ELTE, Budapest, 2001.

Vég Csaba: Alkalmazásfejlesztés a Unified Modeling Language szabványos jelöléseivel, Logos 2000, Debrecen, 1999.

Eric J. Naiburg, Robert A. Maksimchuk: UML földi halandóknak, Kiskapu Kft, Budapest, 2006

Ian Sommerville: Szoftverrendszerek fejlesztése, Panem Könyvkiadó Kft, Budapest, 2002.

Kende Mária, Nagy István: ORACLE pédatár, Panem Könyvkiadó Kft, Budapest, 2005.

http://www.netbeans.org

http://java.sun.com/javase/6/docs/api

http://hsqldb.org/web/hsqlDocsFrame.html

6. Melléklet

6.1 Forgatókönyv

A felhasználó bejelentkezik a rendszerbe:

- Elindítja az alkalmazást
- A megjelenő párbeszéd ablakban beírja a felhasználói azonosítóját és a jelszavát
    - A bejelentkezés sikeres
    - A bejelentkezés sikertelen
        - Újra megpróbálja
        - Kilép az alkalmazásból

A rendszer funkciói csak a bejelentkezés után érhetők el.

A felhasználó megtekinti az alapanyagok listáját:

- Bejelentkezik a rendszerbe
- Kiválasztja a ***Raktár***-t
- Az ablak bal oldalán megjelenik az alapanyagok listája táblázatos formában az alapanyagok fontosabb adataival.

A felhasználó megtekinti egy alapanyag részletes adatait:

- Megtekinti az alapanyagok listáját
- Kiválaszt egy alapanyagot.

A felhasználó új alapanyagot hoz létre:

- Megtekinti az alapanyagok listáját
- Kiválasztja az ***Új alapanyag***-t
- Megjelenik egy új ablak, ahol megadja az új alapanyag adatait

A felhasználó módosítja egy alapanyag adatait:

- Megtekinti egy alapanyag részletes adatait
- Kiválaszt egy alapanyagot
- A módosítandó adatokat átírja
- Kiválasztja az ***Adatok módosítása***-t

A felhasználó töröl egy alapanyagot:

- Megtekinti egy alapanyag részletes adatait
- Kiválasztja a törlendő alapanyagot
- Kiválasztja az ***Alapanyag törlése***-t.

A felhasználó megtekinti, hogy mely alapanyagok hiányosak:

- Megtekinti az alapanyagok listáját
- Kiválasztja a ***Hiányjelző***-t

A felhasználó beszerez egy alapanyagot:

- Megtekinti az alapanyagok listáját
- Kiválaszt egy alapanyagot
- Az új ablakban megadja a beérkezett mennyiséget

A felhasználó megtekinti a termékek listáját:

- Bejelentkezik a rendszerbe
- Kiválasztja a ***Termékek***-t
- Az ablak bal oldalán megjelenik a termékek listája táblázatos formában, jobb oldalon pedig a kiválasztott terméke részletes adatai

A felhasználó új terméket hoz létre:

- Megtekinti a termékek listáját
- Kiválasztja az ***Új termék***-t
- Megjelenik egy új ablak, ahol megadja az új termék adatait
  - Megadja az új termék előállításához szükséges alapanyagokat és azok mennyiségét

A felhasználó megtekinti egy termék részletes adatait:

- Megtekinti a termékek listáját
- Kiválaszt egy terméket
- Az ablak jobb oldalán megjelennek a kiválasztott termék részletes adatai.

A felhasználó módosítja egy termék adatait:

- Megtekinti egy termék részletes adatait
- Kiválasztja a módosítandó terméket
- A jobb oldali ablakban a módosítandó adatokat átírja
- Kiválasztja az ***Adatok módosítása***-t

A felhasználó töröl egy terméket:

- Megtekinti egy termék részletes adatait
- Kiválkasztja a terméket
- Kiválasztja a ***Termék törlése***-t.

A felhasználó előállít egy terméket:

- Megtekinti egy termék részletes adatait
- Kiválkasztja a terméket
- Kiválasztja a ***Sütés***-t.
- Megadja az előállítani kívánt darabszámot

A felhasználó elad egy terméket:

- Megtekinti egy termék részletes adatait
- Kiválasztja a terméket
- Kiválasztja az ***Eladás***-t.
- Megadja az eladni kívánt darabszámot

A felhasználó megtekinti az üzenő falat:

- Bejelentkezik a rendszerbe
- Kiválasztja az ***Üzenő fal***-t

A felhasználó ír az üzenő falra:

- Megtekinti az üzenő falat
- Kiválasztja az ***Új üzenet***-t.

Az adminisztrátor megtekinti a felhasználók listáját:

- Bejelentkezik a rendszerbe
- Kiválasztja a ***Felhasználók***-t
- Az ablak bal oldalán megjelenik a felhasználók listája táblázatos formában a felhasználók fontosabb adataival.

Az adminisztrátor megtekinti egy felhasználó részletes adatait:

- Megtekinti a felhasználók listáját
- Kiválaszt egy felhasználót
- Az ablak jobb oldalán megjelennek a felhasználó részletes adatai.

Az adminisztrátor regisztrál egy új felhasználót:

- Megtekinti a felhasználók listáját
- Kiválasztja az ***Új felhasználó***-t
- Megjelenik egy új ablak, ahol megadja az új felhasználó adatait

Az adminisztrátor módosítja egy felhasználó adatait:

- Megtekinti egy felhasználó részletes adatait
- A módosítandó adatokat átírja
- Kiválasztja az ***Adatok módosítása***-t

Az adminisztrátor töröl egy felhasználót:

- Megtekinti egy felhasználó részletes adatait
- Kiválasztja az ***Felhasználó törlése***-t.