A képen szöveg látható

Automatikusan generált leírás

Java alkalmazások előadás beadandó

Badinszky Dániel

HLISZU

Git User: badinszky97

Holyba Attila Patrik

FLX1D8

Git User: Beantech0

*Github:* <https://github.com/badinszky97/java_CRUD>

**2024.11.10.**

Tartalom

[Fejlesztési és tesztelési környezet 3](#_Toc183815184)

[Feladat - Adatbázis 3](#_Toc183815185)

[Választott adatbázis 3](#_Toc183815186)

[Felhasználói felület 4](#_Toc183815187)

[CRUD: 5](#_Toc183815188)

[Olvas almenü 5](#_Toc183815189)

[Olvas2 almenü 5](#_Toc183815190)

[Ír almenü 6](#_Toc183815191)

[Módosít almenü 6](#_Toc183815192)

[Töröl almenü 6](#_Toc183815193)

[Feladat - Párhuzamos programozás 7](#_Toc183815194)

[Feladat – SoapKliens 8](#_Toc183815195)

[Menüsor 8](#_Toc183815196)

[Letöltés 8](#_Toc183815197)

[Feladat - Forex 9](#_Toc183815198)

[Számlainformációk 9](#_Toc183815199)

[Aktuális árak 10](#_Toc183815200)

[Historikus árak 10](#_Toc183815201)

[Pozíció nyitás 10](#_Toc183815202)

[Pozíció zárás 11](#_Toc183815203)

[Nyitott pozíciók 11](#_Toc183815204)

# Fejlesztési és tesztelési környezet

Az alkalmazás Ubuntu 22.04 LTS verzión lett készítve.

IntelliJ IDEA 2024.2.3 Ultimate Edition

Java verzió: OpenJDK ’17.0.12’ 2024-07-16

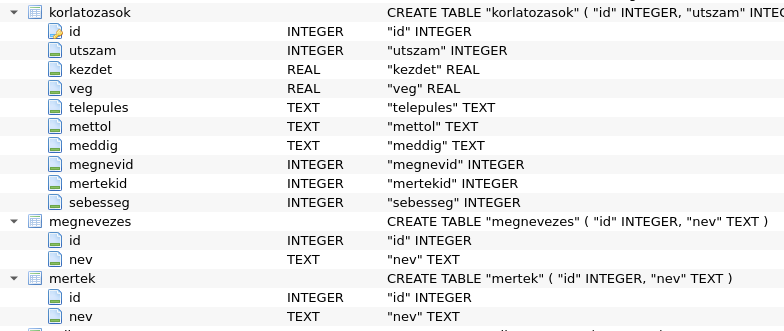
A jar fájl a PDF alapján lett elkészítve és a tesztkörnyezeten tesztelve.

# Feladat - Adatbázis

## Választott adatbázis

A választott adatbázis: Forgalomkorlátozások.

Ez az adatbázis magyarországi utakat tartalmaz és azok lezárásainak adatait. Összesen 3 tábla található benne.



A JavaFX alkalmazás a Hibernate csomagon keresztül egy SQLite fájlt használ adatbázis kiszolgálóként. Az adatbázis linkje [itt](https://github.com/badinszky97/java_CRUD/blob/master/hibernate.db) található.

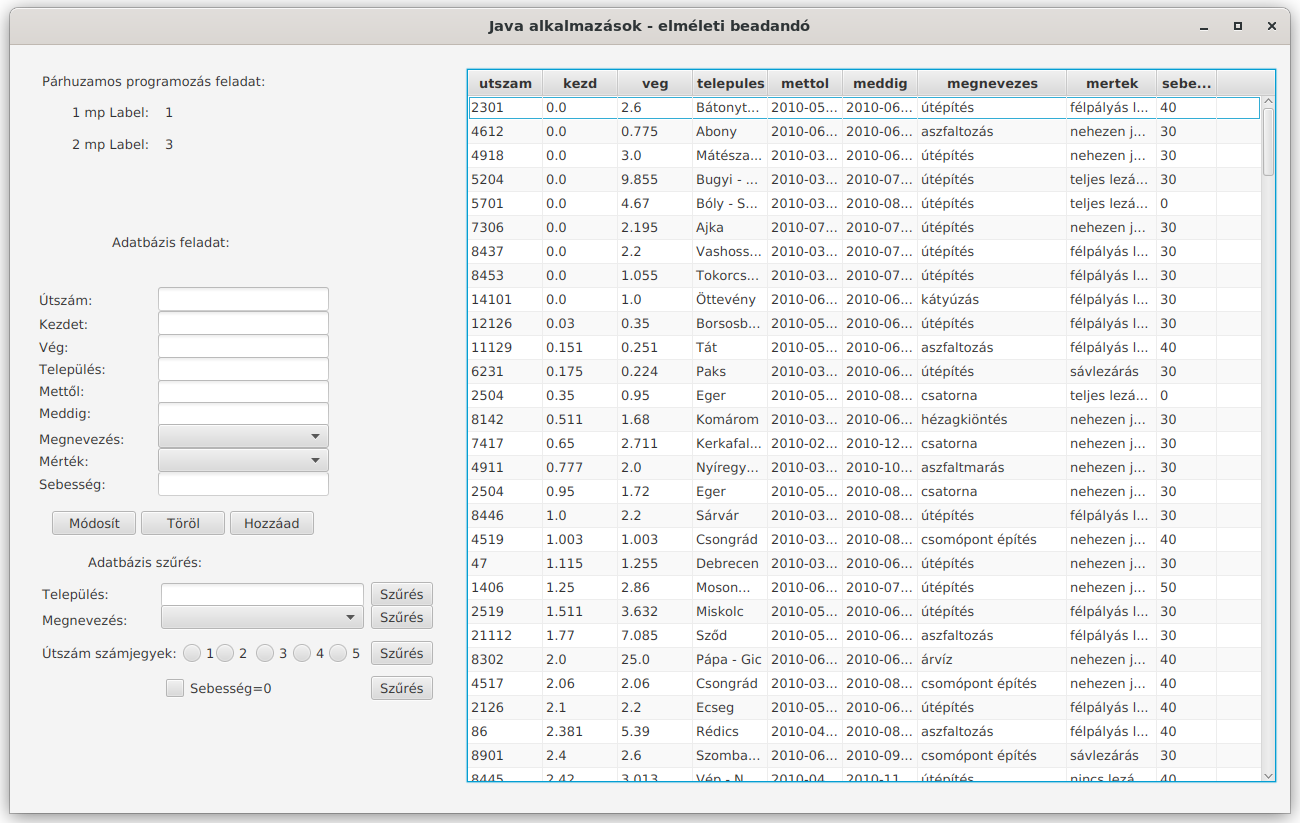
A Hibernate használatóhoz létre Kellett hozni mind a 3 táblához kapcsolódóan a Java osztályokat. Ezek “Korlatozas”, “Mertek” és “Megnevezes” nevet kaptak. A Korlátozás osztály van alapvetően használva és ennek van két példányosított változója ami a két másik osztályhoz kapcsolódik. Ebben az esetben több az egyhez kapcsolat áll fent a táblák között mind a két esetben. Ez a forráskódban jelölve is van:

@ManyToOne

@JoinColumn(name = "megnevid")

private Megnevezes megnevezes;

## Felhasználói felület



Párhuzamos programozás rész

Adatbázis feladat rész

Adatbázis feladat GUI rész

A felhasználói felület elkészítéséhez A Scenebuilder applikáció volt használva. Az összes itt látható vezérlő a GUIController java osztályban van definiálva és használva.

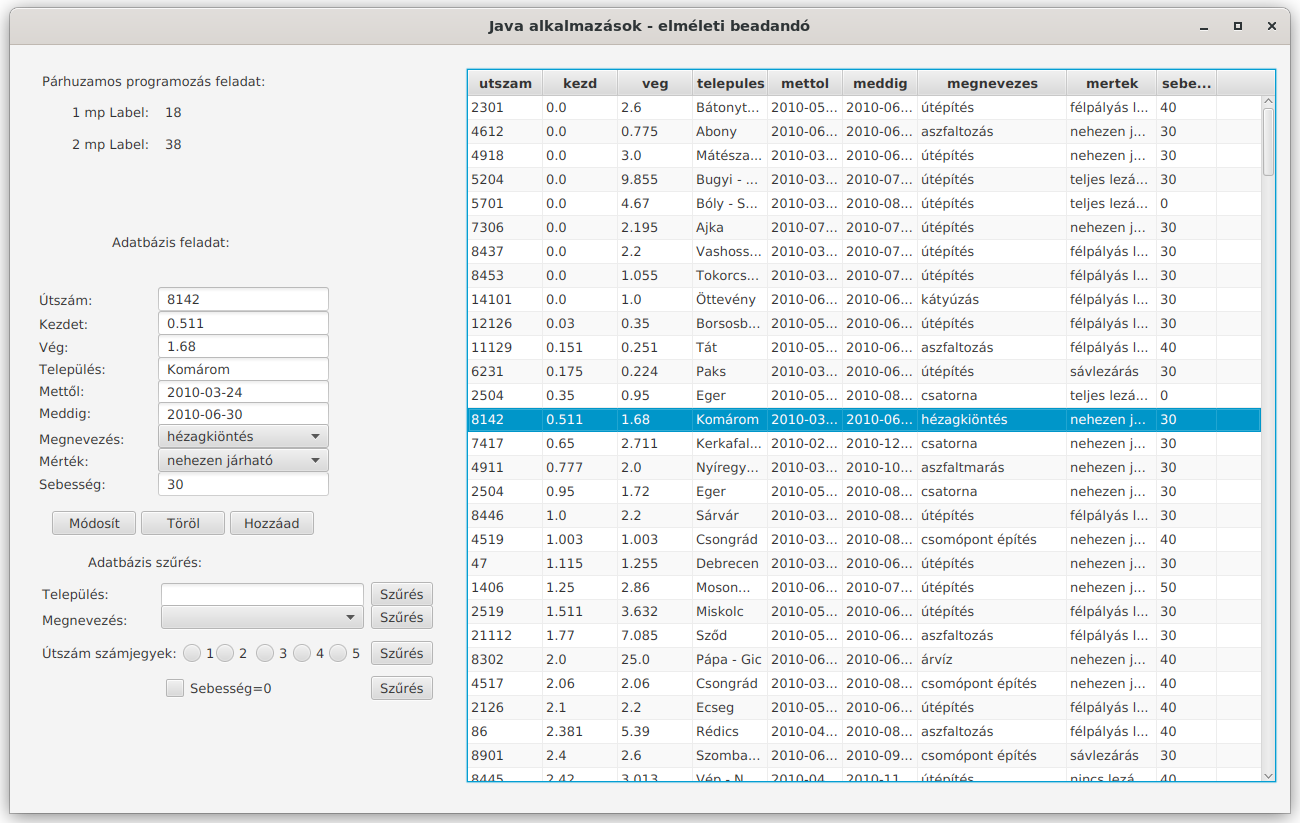
A felhasználói felületet egyszerűre terveztem. A két alapvető feladatot látja el ami a beadandó kiírásáában volt:

- Két labelen különböző időintervallumokkal egy egy szöveg változik, ezzel megvalósítva, a párhuzamos programozást.

- Egy SQLite adatbázishoz csatlakozva egy GUI felületen megvalósítja a CRUD modellt.

## CRUD:

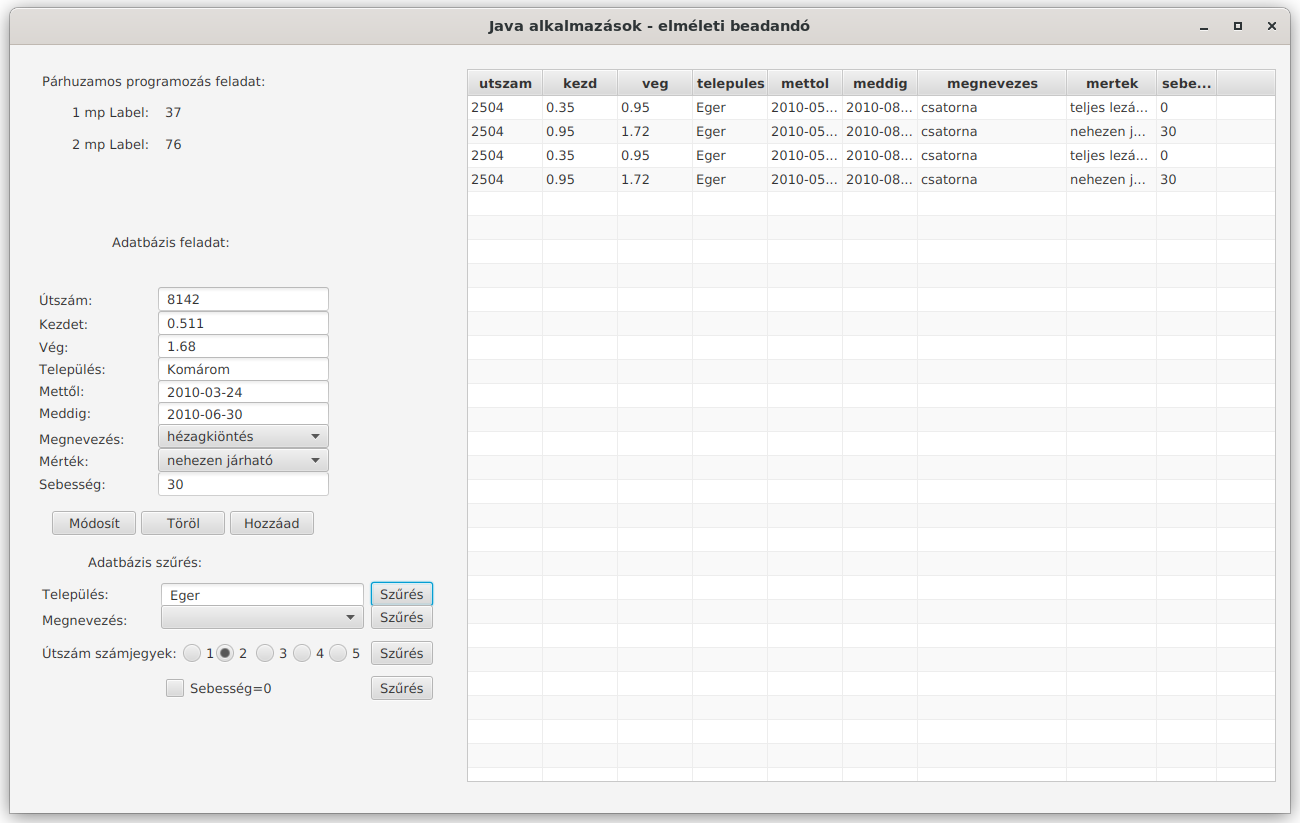
### Olvas almenü

A program elindulásakor a teljes adatbázist betölti a program és megjeleníti a jobb oldali mezőben. Itt lehet sorba rendezni bármelyik oszlop szerint.

A táblázat megjelenítéséhez nincs szükség segéd listákra. Úgy lett kialakítva, hogy képes fogadni a Korlátozások osztály egy-egy példányát és natívan megjeleníteni. A Korlátozások osztály tartalmaz 2 másik osztály-példányt is magában, mert a Megnevezés és Mérték elemek külön táblából olvashatóak. Ahhoz, hogy ez ne okozzon problémát, a toString metódus módosítva lett mind a két osztályban.

@Override  
public String toString() {  
 return this.nev;  
}

### Olvas2 almenü

A szűrés feladathoz külön rész let kialakítva a kezelőfelületen.

A feladat kiírásban szerepeltetett vezérlők itt találhatóak.

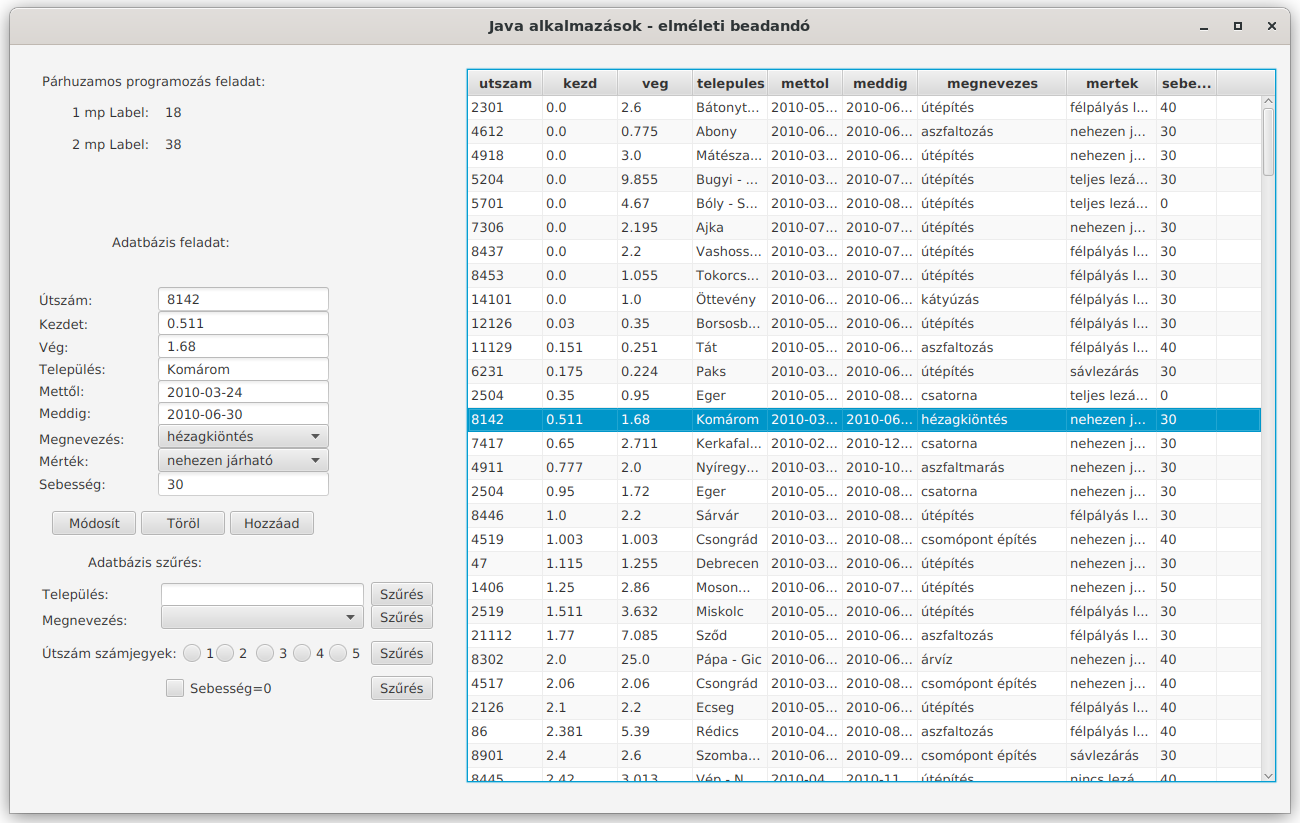
Lehetőséget ad Település név szerinti szűrésre. Ehhez a mezőbe pontos nevet kell megadni. Maszkolási lehetőség nincs implementálva. A Szűrés gombra kattintva a jobb oldali lista frissülni fog, immáron csak az arra a településre vonatkozó rekordok fognak megjelenni**. Ha a mező üres akkor szűretlen lista jelenik meg job oldalt, ezzel lehet törölni a szűrést.**

Lehetőség van Megnevezés alapján is szűrni. Ehhez az alkalmazás az adatbázisból felolvasott mezőket jeleníti meg. Nem fixen beégetett lista van alkalmazva. A megnevezés kiválasztása és a gombra kattintás után a jobb oldali lista frissül.

A radiobutton feladatot az útszám számjegyeire implementáltam. Ki lehet választani, hogy hány számjegyből álljon az útszám. A radiobutton kiválasztása után a gombra kattintva a jobb oldali lista frissül.

A checkbox mezővel arra lehet szűrni, hogy csak azok a rekordok jelenjenek meg, ahol a sebességnek 0 van megadva. A gombra kattintva a lista frissül.

### Ír almenü

A kezelőfelület bal oldalán van egy form ami az adatok manipulálására szolgál.

Ha hozzá szeretnénk adni az adatbázishoz, akkor ki kell tölteni a formot és rákattintani a “Hozzáad” gombra. Ekkor a jobb oldali táblázat frissül és már tartalmazza is az új elemet. Az elem azonnal commit-álásra kerül az adatbázisfájlban. Amennyiben előtte szűrés volt aktív az inaktívvá válik és szűretlen lista jelenik meg.

### Módosít almenü

Egy elemre kattinta a jobb oldali táblázatban kijelölésre kerül az adott elem. Az adatai átmásolódnak a baloldani formba, ahol szabadon szerkeszthetjük. Szerkesztés után a módosít gombra kattinva az elem módosul az adatbázisfájlban.

### Töröl almenü

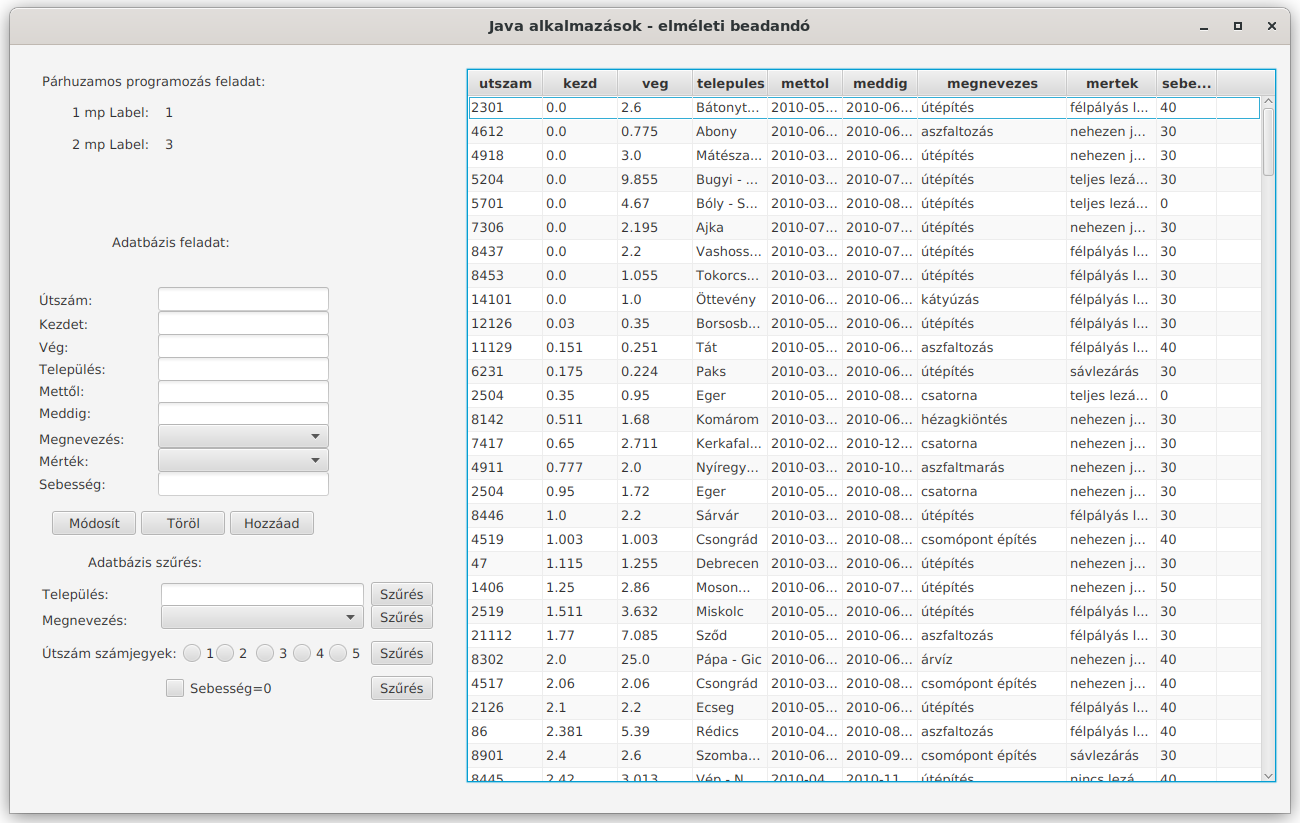
Egy elemre kattintva az kijelölődik. Ezek után a Törlés gombra kattintva az elem véglegesen törlődik az adatbázisból.

# Feladat - Párhuzamos programozás

A felület bal felső sarkában található a párhuzamos programozásra szánt rész.

Itt két label található. Az egyik 1, a másik 2 mp-enként növeli az értékét eggyel, így egy-egy számlálót létrehozva. Ennek megvalósításához a TimeLine objektumot használtam ami kezeli a szálak közti versenyhezetet és képes megoldani a szálak közti adatátvitelt.

*/\*\**  
 *\* Párhuzamos programozás feladat objektumok*  
 *\*/*  
Timeline timelinemp = new Timeline(  
 new KeyFrame(Duration.*seconds*(1),  
 e -> {  
 mpLabel.setText(mp.toString());  
 mp++;  
 }));  
  
Timeline timelineduplamp = new Timeline(  
 new KeyFrame(Duration.*seconds*(2),  
 e -> {  
 duplampLabel.setText(duplamp.toString());  
 duplamp++;  
 }));

Az időzítők rögtön a program indulásakor elindulnak, így nem igényelnek semmi gombot az induláshoz. Teljesen automatizált.

# Feladat – SoapKliens

## Menüsor

A szükséges formok sokasága miatt a Soap és a Forex menűk külön aloldalakra kerültek.

Ennek megoldására MenuBar lett használva, az átláthatóság miatt pedig az fxml fájlok egymásba lettek ágyazva. A fő a “main\_layout.fxml” ebben van a menuBar és ez kommunikál a Controllerrel.

A menu választáskor a controllerben levő loadView függvény tölti be a main\_layout-ban

<VBox fx:id="contentArea"/>

Vbox mezőbe a menüben megadott fxml fájl alapján.

## Letöltés

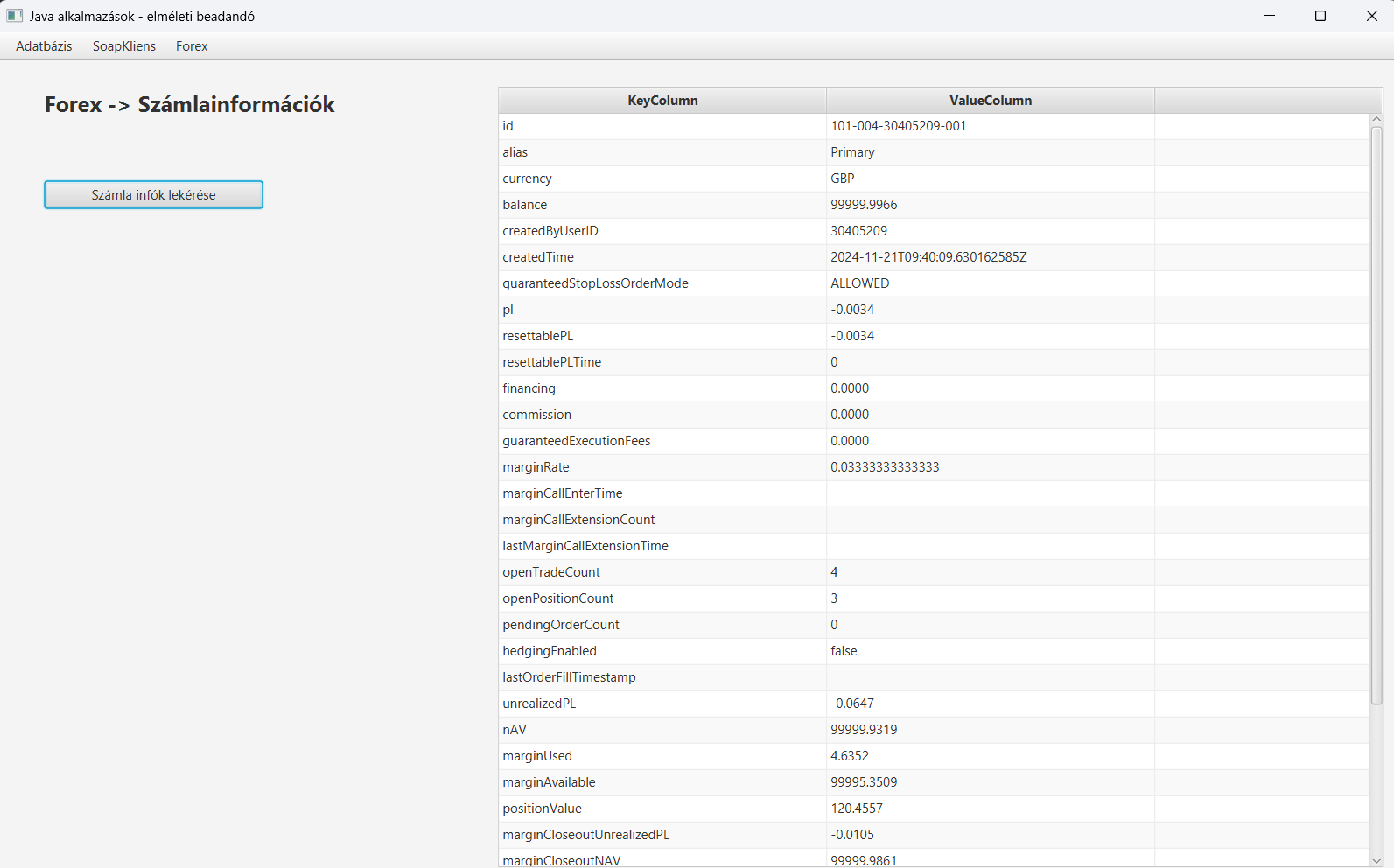
# Feladat - Forex

Az Oanda API feladatnál a pom.xml fájlban az alábbi módon lett behúzva a dependency.

<dependency>  
 <groupId>com.oanda.v20</groupId>  
 <artifactId>v20</artifactId>  
 <version>3.0.25</version>  
</dependency>

Ezenkívül több oanda osztályt is szükséges volt engedélyezni a module-info.java fájlban.

## Számlainformációk



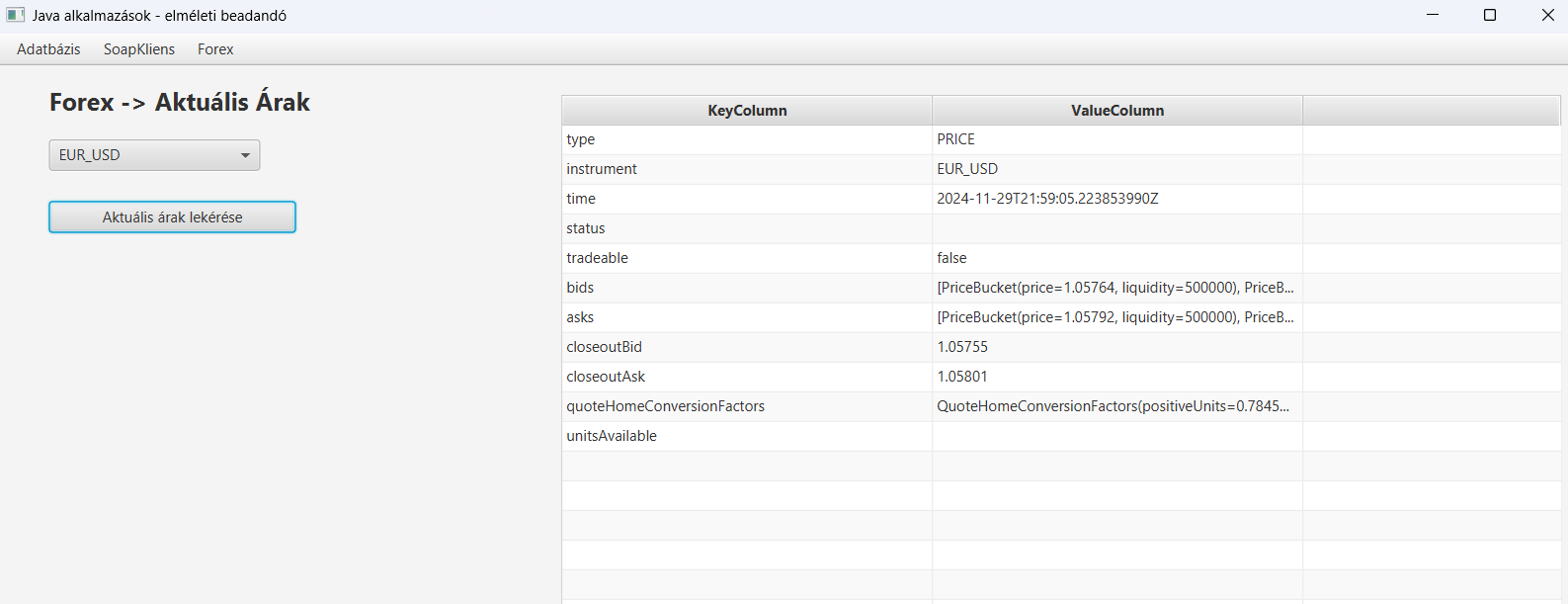
A számlainformációk lekérése a gombra kattintás után történik meg a “forexGetSzamlaInfo” függvényben.

Az oandas feladathoz létrehoztam egy oandaConfig osztályt az átláthatóság miatt, ezt a Controller elején példányosítom, ebben a osztályban a konstruktor létre is hozza a contextet automatikusan.  
Készítettem egy accountInformation függvényt ami a getDeclaredFields()-el kiszedi az összes adatot amit a summary lekérdezés visszaad. Ezt egy 2 elemű listát tartalmazó tömbbe teszem és ezt adom vissza a controllernek.

accountDetailTbl.setItems(FXCollections.*observableArrayList*

Függvény betölti a visszaadott listát a táblázatba.

## Aktuális árak



Az Aktuális árak feladat hasonló táblázatba tölti az adatokat mint a Számlainformáció.

A lenyiló menű tartalmát a loadView résznél feltöltöm opciókkal.

A gombra kattintva lekéri a kiválasztott értéknek megfelelő adatot.

Ez a rész az OandaConfig osztályon belül a “getActualPrice” függvény oldja meg.

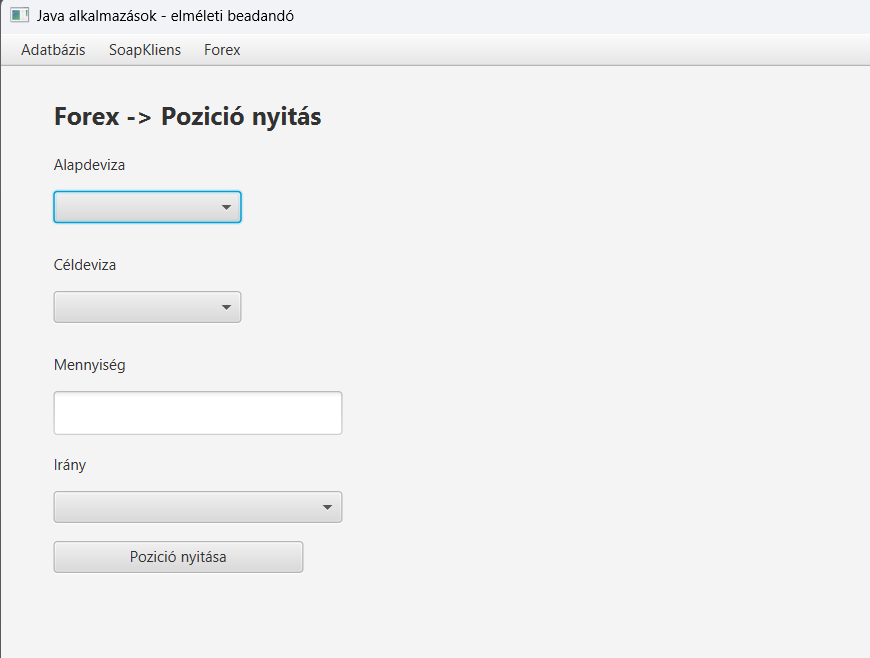
Itt is az adatokat egy listaként adom vissza 2 a Key-Value párokat soronként jelenítem meg.

Ha nincs kiválasztva elem a lenyiló mezőben akkor alertként hibát dob fel.

## Historikus árak

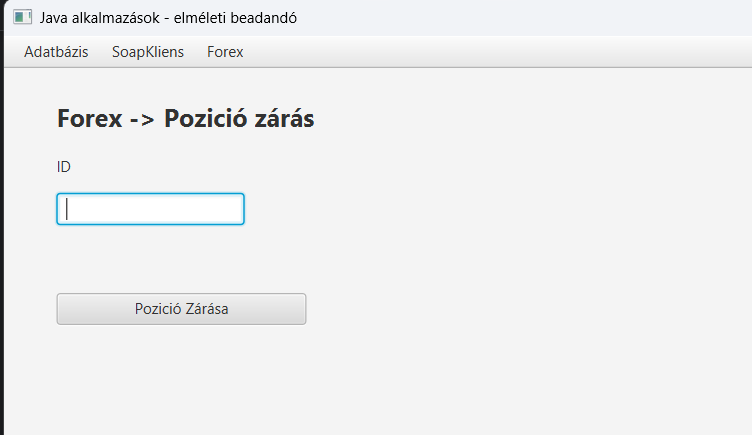
Ez a rész még nincs kész. (Erre most jöttem rá a dokumentációnál)

## Pozíció nyitás



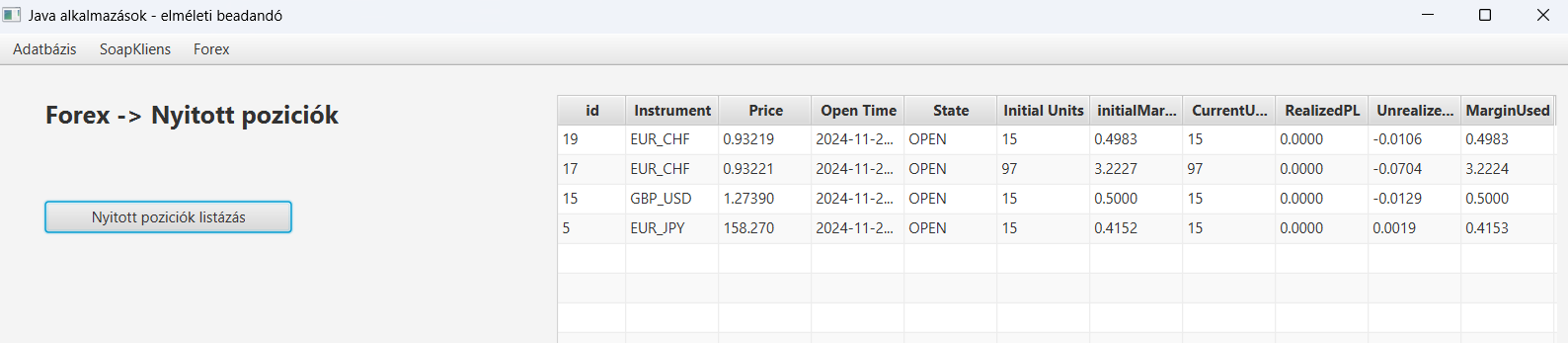
Egy form jön fel a Pozíció nyitás gombot választva ahol a formon minden mezőt kötelező megadni és azt is ellenörzi hogy a mennyiség cask szám lehet. A controllerben az oandaOpenPos függvény felel ezért a részért, ha pedig minden rendben akkor átadja a kiválasztott értékpárt és az értéket a oandaConfig.setOpenPosition függvénynek ahol pedig a request létrehozása után a setOrder hivással kiadom a pozíció nyítást.

## Pozíció zárás



A pozíció nyitáshoz hasonlóan egy form jön fel ahol csak az ID-t lehet megadni, ez a nyitott pozicíó azonosítója amit a Nyított pozíciókban lehet megnézni. A zárás a Controller “oandaClosePos” függvényén keresztül az “oandaConfig” osztályban a “setClosePosition” függvényben történik meg.

## Nyitott pozíciók



A nyított pozíciók menüben a gombra kattintásra lekéri a nyított pozíciókat a rendszer és a PropertyValueFactory fxml mező alapján betölti a táblázatba a kapott adatokat.

Ezt a függvény is szintén az OandaConfig osztályban van és az FXCollections.observableArrayList függvénnyel kerül betöltésre a táblázat elemei. Ami egy Trade listát ad vissza a getTrades() függvény használatakor.