



Neumann János Egyetem  
Műszaki és Informatikai  
Kar

# JAVA Alkalmazások

## Elméleti beadandó

Hoki Attila – F2XVKV  
Koltai Armand - GYBASN

# Tartalomjegyzék

<b>1.</b>	<b>GITHUB REPOSITORY .....</b>	<b>2</b>
<b>2.</b>	<b>FELHASZNÁLÓI FELÜLET .....</b>	<b>3</b>
2.1.	KEZDŐNÉZET ÉS MENÜSOR .....	3
2.2.	ADATBÁZIS FELADAT.....	4
2.2.1.	<i>Olvas részfeladat.....</i>	<i>4</i>
2.2.2.	<i>Olvas2 részfeladat.....</i>	<i>4</i>
2.2.3.	<i>Ír részfeladat .....</i>	<i>5</i>
2.2.4.	<i>Módosít részfeladat .....</i>	<i>6</i>
2.2.5.	<i>Töröl részfeladat .....</i>	<i>6</i>
2.3.	SOAP KLIENS (MNB) FELADAT .....	6
2.3.1.	<i>Letöltés részfeladat.....</i>	<i>6</i>
2.3.2.	<i>Letöltés2 részfeladat.....</i>	<i>7</i>
2.3.3.	<i>Grafikon részfeladat.....</i>	<i>8</i>
2.4.	PÁRHUZAMOS FELADAT .....	8
<b>3.</b>	<b>ÜZLETI LOGIKA.....</b>	<b>8</b>
3.1.	ADATBÁZISTERVEZÉS .....	8
3.2.	ADATBÁZISOK FELADAT.....	9
3.2.1.	<i>Olvas részfeladat.....</i>	<i>9</i>
3.2.2.	<i>Olvas2 részfeladat.....</i>	<i>11</i>
3.2.3.	<i>Ír részfeladat .....</i>	<i>12</i>
3.2.4.	<i>Módosít részfeladat .....</i>	<i>14</i>
3.2.5.	<i>Töröl részfeladat .....</i>	<i>15</i>
3.3.	SOAP FELADAT.....	16
3.3.1.	<i>Letöltés részfeladat.....</i>	<i>16</i>
3.3.2.	<i>Letöltés2 részfeladat.....</i>	<i>19</i>
3.3.3.	<i>Grafikon részfeladat.....</i>	<i>20</i>
3.4.	PÁRHUZAMOS FELADAT .....	22

# 1. GitHub Repository

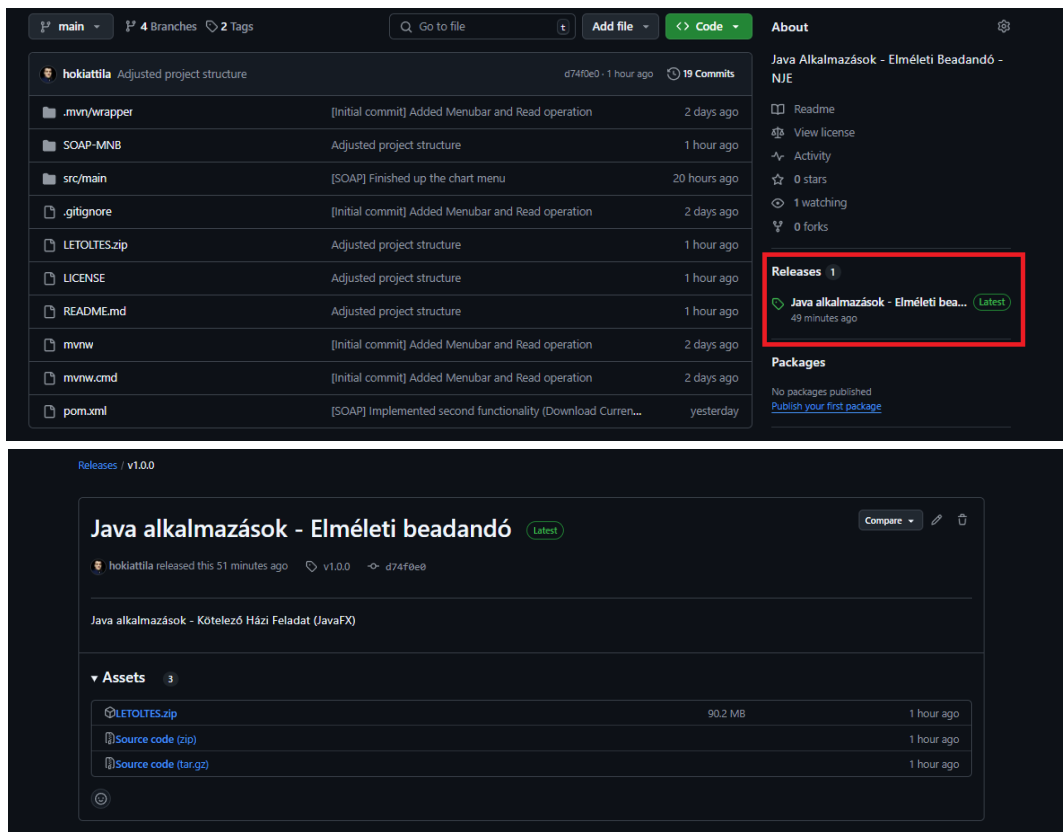
A feladat megoldása a következő GitHub tárolóban érhető el: [java\\_beadando\\_nje](#)

Teszteléshez a **LETOLTES.ZIP**-ben csomagolt állományokra van csak szükség. Az archívum elérhető a projekt gyökérkönyvtárában:



hokiattila Adjusted project structure	d74f0e0 · 52 minutes ago	19 Commits
<code>.mvn/wrapper</code>	[Initial commit] Added Menubar and Read operation	2 days ago
<code>SOAP-MNB</code>	Adjusted project structure	52 minutes ago
<code>src/main</code>	[SOAP] Finished up the chart menu	20 hours ago
<code>.gitignore</code>	[Initial commit] Added Menubar and Read operation	2 days ago
<code>LETOLTES.zip</code>	Adjusted project structure	52 minutes ago
<code>LICENSE</code>	Adjusted project structure	52 minutes ago
<code>README.md</code>	Adjusted project structure	52 minutes ago
<code>mvnw</code>	[Initial commit] Added Menubar and Read operation	2 days ago
<code>mvnw.cmd</code>	[Initial commit] Added Menubar and Read operation	2 days ago
<code>pom.xml</code>	[SOAP] Implemented second functionality (Download Curren...	yesterday

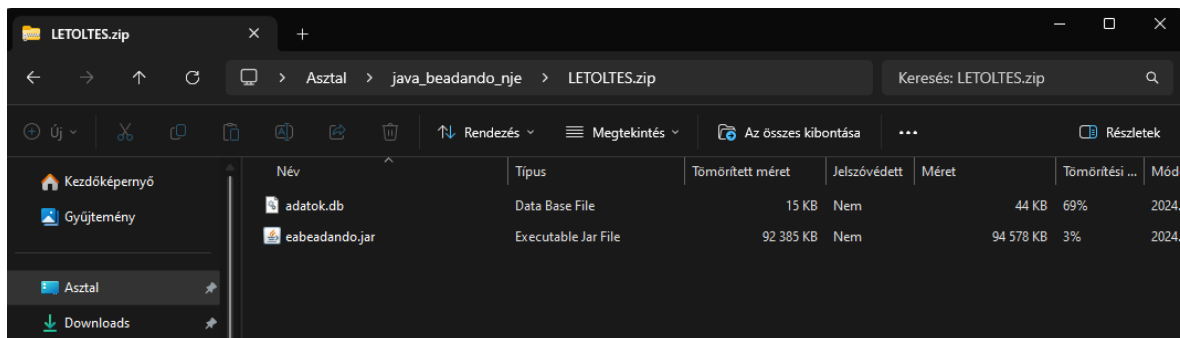
Ha nincs szükségünk a forráskódra és csak a végeredményt szeretnénk futtatni akkor nem kell az egész kódbázist letölteni, oldalt a *Releases* menüpontban is megtaláljuk a könyvtárat:



The first screenshot shows the main repository page for 'hokiattila Adjusted project structure'. The 'Releases' section on the right sidebar is highlighted with a red box, showing 'Java alkalmazások - Elméleti bea...' as the latest release, 49 minutes ago.

The second screenshot shows the 'Releases / v1.0.0' page. It displays the release details for 'Java alkalmazások - Elméleti beadandó' (Latest). Under the 'Assets' section, there are three items: 'LETOLTES.zip' (90.2 MB, 1 hour ago), 'Source code (zip)' (1 hour ago), and 'Source code (tar.gz)' (1 hour ago).

A LETOLTES.ZIP-ben két állományt találunk, a megoldás JAR fájl, valamint az adatok. *SQLite* adatbázis állományt:

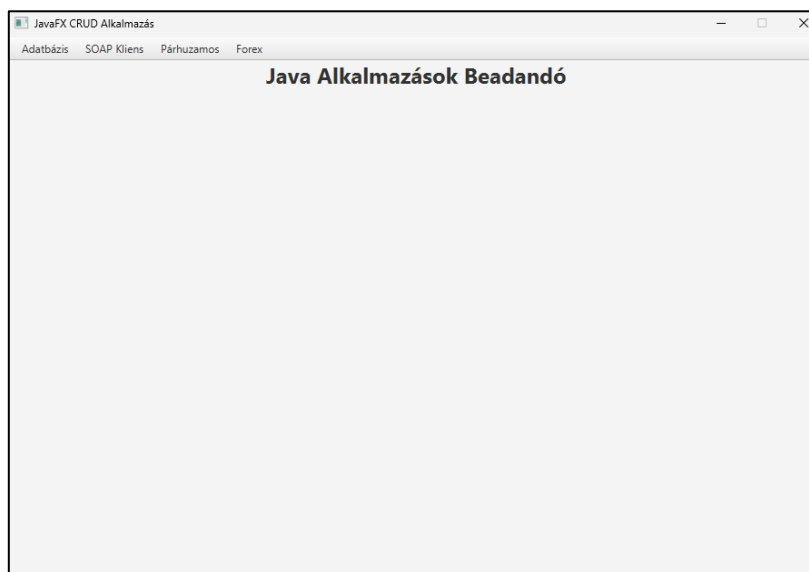


A futtatást megelőzően az **adatok.db** állományt a **c:\adatok** könyvtárba kell helyezni, a program a feladat leírásának megfelelően itt fogja keresni a betöltendő adatokat.

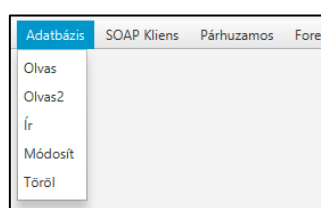
## 2. Felhasználói felület

### 2.1. Kezdőnézet és menüsor

Ha elindítjuk a JAR fájlt akkor a következő nézet töltődik be:



Alapértelmezetten az alkalmazás főoldala egy üres nézetből áll, amiben csak a „*Java Alkalmazások Beadandó*” főcím és a menüsor található. A feladatok megoldásait a menüsoron keresztül az egyes almenükre kattintva érhetjük el:

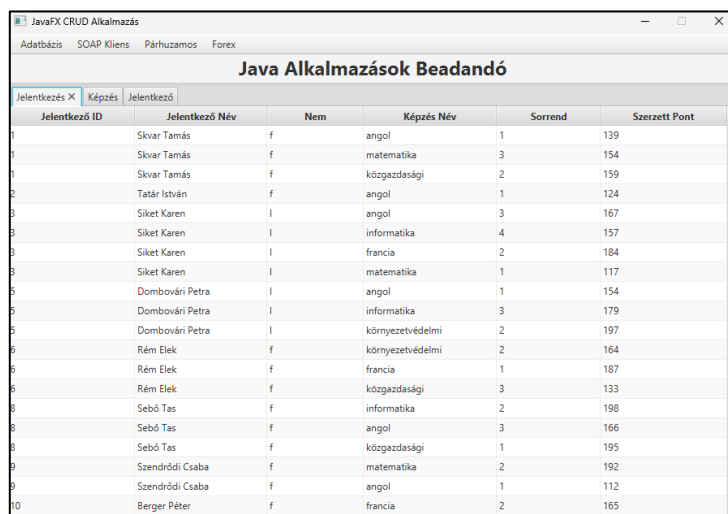


A menüsor nézettől függetlenül mindig elérhető. A menüpontok mind egy-egy fő feladat megoldását tartalmazzák (5 pontos), melyen belül az almenük egy-egy részfeladatot valósítanak meg.

## 2.2. Adatbázis feladat

### 2.2.1. Olvas részfeladat

Ha az adatbázis menüponton belül az Olvas almenüre kattintunk akkor a program betölti az adatok.db adatbázisban található táblákat:



The screenshot shows the 'Adatbázis' menu selected in the 'JavaFX CRUD Alkalmazás' window. The 'Jelentkező' tab is active, displaying a table of applicants with the following data:

Jelentkező ID	Jelentkező Név	Nem	Képzés Név	Sorrend	Szerzett Pont
1	Skvar Tamás	f	angol	1	139
1	Skvar Tamás	f	matematika	3	154
1	Skvar Tamás	f	közgazdasági	2	159
2	Tatár István	f	angol	1	124
3	Siket Karen	l	angol	3	167
3	Siket Karen	l	informatika	4	157
3	Siket Karen	l	francia	2	184
3	Siket Karen	l	matematika	1	117
5	Dombóvári Petra	l	angol	1	154
5	Dombóvári Petra	l	informatika	3	179
5	Dombóvári Petra	l	környezetvédelmi	2	197
6	Rém Elek	f	környezetvédelmi	2	164
6	Rém Elek	f	francia	1	187
6	Rém Elek	f	közgazdasági	3	133
8	Sebő Tas	f	informatika	2	198
8	Sebő Tas	f	angol	3	166
8	Sebő Tas	f	közgazdasági	1	195
9	Szendrői Csaba	f	matematika	2	192
9	Szendrői Csaba	f	angol	1	112
10	Berger Péter	f	francia	2	165

A nézeten belüli tabulátorokkal tudjuk váltani, hogy mely tábla adataira vagyunk kíváncsiak. Az adatbázis mindhárom táblája betölthető:

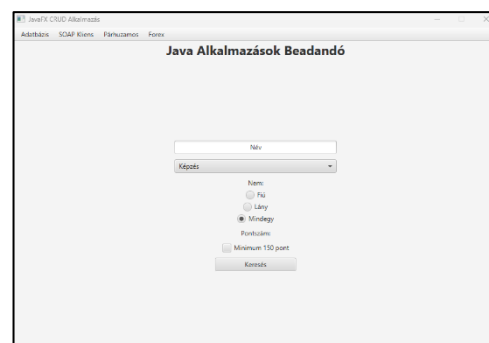


The screenshot shows the 'Adatbázis' menu selected in the 'JavaFX CRUD Alkalmazás' window. The 'Képzés' tab is active, displaying a table of training programs with the following data:

Képzés ID	Képzés Név	Képzés limit	Képzés minimum
1	francia	16	140
2	angol	30	150
3	matematika	16	160
4	informatika	16	150
5	környezetvédelmi	16	130
6	közgazdasági	30	130

### 2.2.2. Olvas2 részfeladat

A Olvas2 almenüre kattintva egy *form* töltődik be, ahol megadhatjuk a keresési feltételeket, szűrhetünk a jelentkező nevére, kiválaszthatunk képzést, kérhetünk le adatokat nem szerint, valamint egy minimum 150 pontos feltételt is beállíthatunk.



The screenshot shows the 'Adatbázis' menu selected in the 'JavaFX CRUD Alkalmazás' window. The 'Olvas2' sub-menu is active, displaying a search form with the following fields and options:

- Név:
- Képzés:
- Nem: ☐ Fé ☐ Nő
- Mindegy: ☒ Nem
- Minimum 150 pont: ☐
- Keresés:

A keresés gombra kattintva új nézet töltődik be az eredményekkel. Példa a „*Péter*” nevű jelentkezők szűrésére:

JavaFX CRUD Alkalmazás

Adatbázis SOAP Kliens Párhuzamos Forex

**Java Alkalmazások Beadandó**

Jelentkező ID	Jelentkező ...	Nem	Képzés Név	Sorrend	Szerzett Pont
10	Berger Péter	f	francia	2	165
10	Berger Péter	f	matematika	1	123
31	Zsolnai Péter	f	francia	2	152
31	Zsolnai Péter	f	informatika	1	186
125	Kiss-Szabó ...	f	közgazdasági	1	166
183	Péter Tamás	f	francia	1	181
183	Péter Tamás	f	informatika	2	132
231	Müller Péter	f	informatika	1	187
242	Sütő Péter	f	francia	1	186
374	Késmárky P...	f	angol	2	173
374	Késmárky P...	f	közgazdasági	1	151
374	Késmárky P...	f	matematika	3	143

Új keresés

Itt az új keresés gombra kattintva, visszatérhetünk az űrlapra.

### 2.2.3. Ír részfeladat

Az ír részfeladatban lehetőségünk van mind három táblába adatokat feltölteni, a tabulátorokra kattintva válthatunk az egyes táblákhoz tartozó űrlapok között:

JavaFX CRUD Alkalmazás

Adatbázis SOAP Kliens Párhuzamos Forex

**Java Alkalmazások Beadandó**

Jelentkező X Képzés X Jelentkező

Jelentkező neve

Nem

Jelentkező hozzáadása

Képzés neve

Felvételek száma

Minimális pontszám

Képzés hozzáadása

A hozzáadás gombra kattintva az adatok mentésre kerülnek. Példa *Gipsz Jakab* nevű jelentkező beszúrására:

JavaFX CRUD Alkalmazás

Adatbázis SOAP Kliens Párhuzamos Forex

**Java Alkalmazások Beadandó**

Jelentkező X Képzés X Jelentkező

Gipsz Jakab

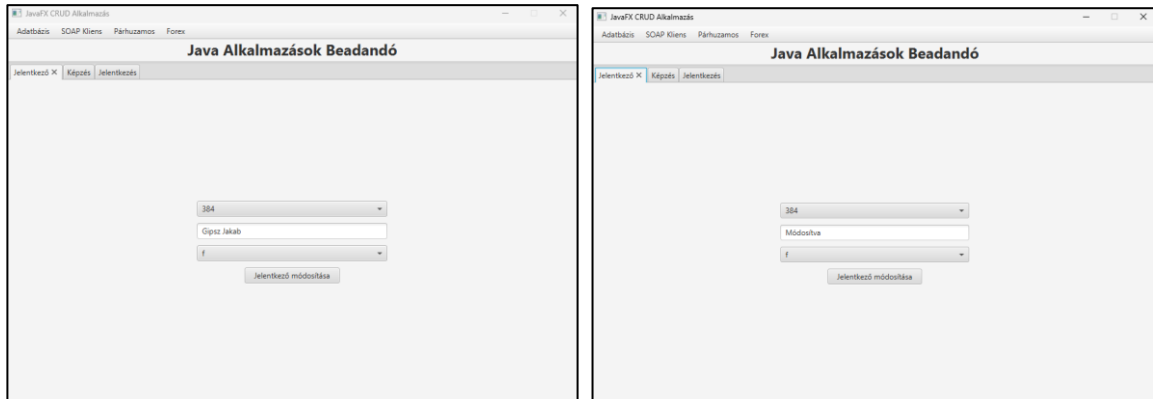
Fü

Jelentkező hozzáadása

380	Németh Edit	I
381	Polgár István	f
382	Molnár Mónika	I
383	Szabó Boglárka Gyöngyi	I
384	Gipsz Jakab	f

### 2.2.4. Módosít részfeladat

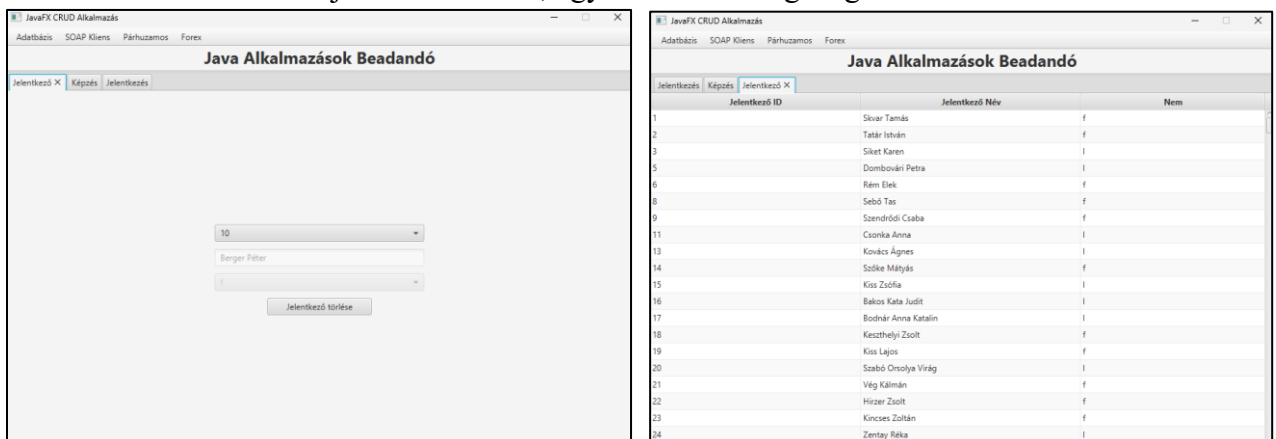
A módosít részfeladat megoldása hasonló az ír feladathoz, azzal a különbséggel, hogy a felhasználó egy *Combo*x segítségével kiválaszthatja a módosítandó rekordot. Példaként módosítom az előzőekben beszúrt *Gipsz Jakab* jelentkező nevét „Módosítva” -ra.



381	Polgár István	f
382	Molnár Mónika	l
383	Szabó Boglárka Gyöngyi	l
384	Módosítva	f

### 2.2.5. Töröl részfeladat

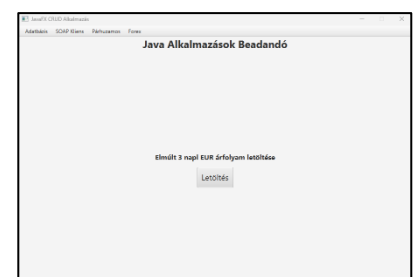
A törlés nézetében annyi a különbség, hogy itt, ha bár látja a felhasználó a kiválasztott rekordok attribútumait, viszont azok le vannak tiltva, így nem módosíthatók. Egyedül a törlendő rekord azonosítója választható ki, egy *Combobox* segítségével:



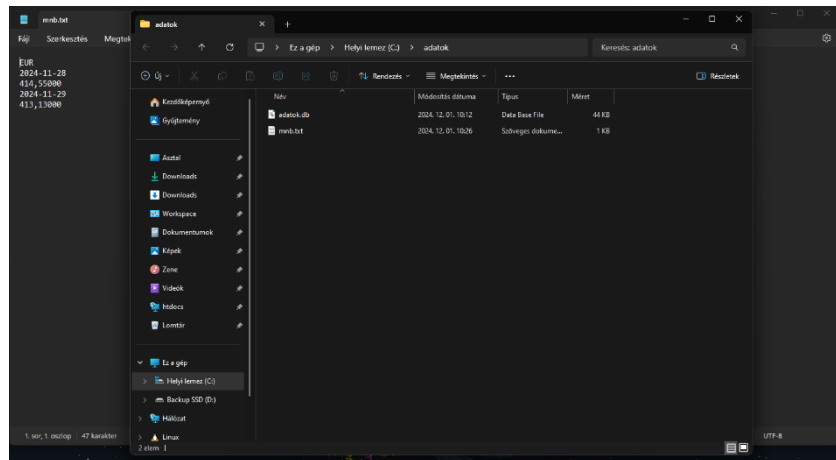
## 2.3. SOAP Kliens (MNB) feladat

### 2.3.1. Letöltés részfeladat

A letöltés részfeladatban egy egyszerű nézetet kapunk, ahol egy gomb segítségével lekérhetjük az elmúlt 2 nap euró árfolyamának változását.

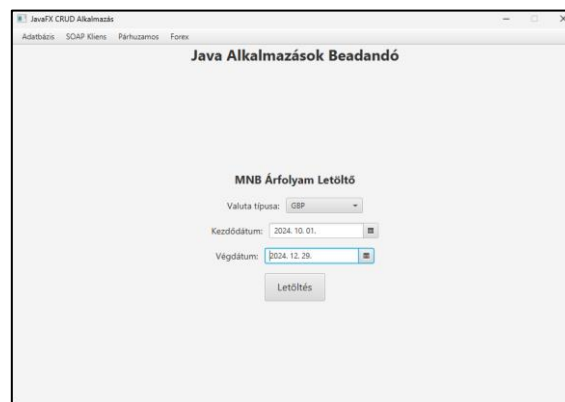


A gomb megnyomását követően az alkalmazás letölti az adatokat a **c:\adatok** mappába, **mnb.txt** néven. Sikeres futást követően a rendszer automatikusan meg is nyitja a mappát és az txt fájlt is:

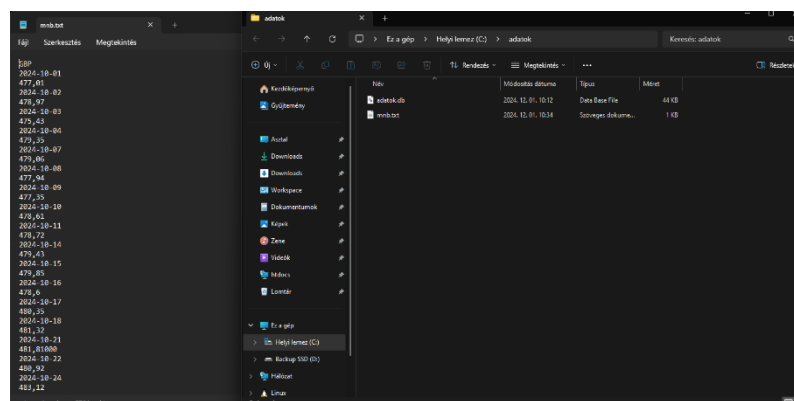


### 2.3.2. Letöltés2 részfeladat

A Letöltés 2-höz tartozó nézet először egy űrlapot tölt be, ahol kiválaszthatjuk a deviza típusát egy *ComboBox* segítségével, valamint kezdő és végdátumot állíthatunk be *DatePicker*-ek segítségével:



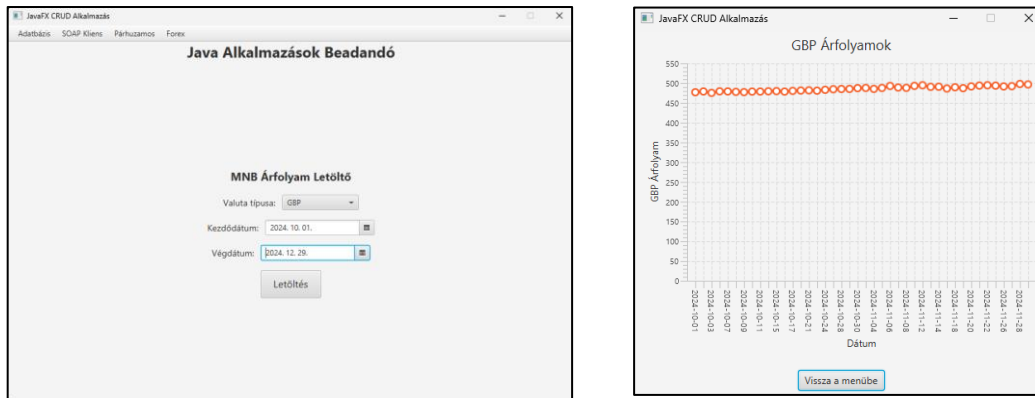
Ezt követően a rendszer az előzőekben említettekhez hasonlóan letölti az adatokat és automatikusan megnyitja a text állományt, valamint az azt tartalmazó mappát:





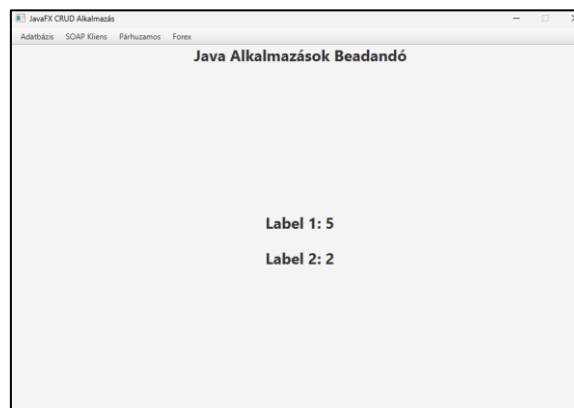
### 2.3.3. Grafikon részfeladat

A grafikon feladat ugyan azt az űrlapot használja, mint a *Letöltés2* azzal a különbséggel, hogy itt már nem csak az adatok letöltése valósul meg, hanem JavaFX chart segítségével kirajzolja annak a grafikonját:



### 2.4. Párhuzamos feladat

A párhuzamos feladatban két címkehez rendelt integer érték növelését láthatjuk, eltérő időközönként a két külön szálon futó (párhuzamosított) működésnek köszönhetően:



## 3. Üzleti logika

### 3.1. Adatbázis-tervezés

A feladathoz a *Középiskolai felvételi* adatbázist választottuk, így ennek az adatforrásnak a txt állományait kellett feldolgoznunk. Választásunk az *SQLite* által kínált lokális relációs adatbázis-kezelőre esett, hiszen a feladat leírása alapján alkalmazásunk nem használhat külső segédszolgáltatást (pl. XAMPP vagy MongoDB server). A *jelentkező* és *képzés* entitás táblákkal dolgoztunk, a köztük fennálló kapcsolatot pedig a *jelentkezés* tábla megvalósításával oldottuk meg.

Az adatbázist létrehozó SQL szkript:

```
CREATE TABLE jelentkezo (  
    id INTEGER PRIMARY KEY,  
    nev TEXT NOT NULL,  
    nem TEXT NOT NULL  
);  
  
CREATE TABLE kepzes (  
    id INTEGER PRIMARY KEY,  
    nev TEXT NOT NULL,  
    felveheto INTEGER NOT NULL,  
    minimum INTEGER NOT NULL  
);  
  
CREATE TABLE jelentkezes (  
    jelentkezoId INTEGER NOT NULL,  
    kepzesId INTEGER NOT NULL,  
    sorrend INTEGER NOT NULL,  
    szerzett INTEGER NOT NULL,  
    FOREIGN KEY (jelentkezoId) REFERENCES jelentkezo(id),  
    FOREIGN KEY (kepzesId) REFERENCES kepzes(id)  
);
```

A feladat leírásnak megfelelően az adatbázis a **c:\adatok** mappában került elhelyezésre, az adatkezelésért felelős DAO is itt fogja keresni.

## 3.2. Adatbázisok feladat

### 3.2.1. Olvas részfeladat

Feladathoz tartozó view: **read-view.fxml**

Ez a kód három különböző táblázatot (TableView-t) tartalmaz egy TabPane elemben. A TabPane több különálló fületet jelenít meg, így az adatokat három különböző kategóriába sorolja, amik között az almenüben váltani lehet.

```
<StackPane xmlns="http://javafx.com/javafx/21" xmlns:fx="http://javafx.com/fxml/1"  
fx:controller="nje.ea.eabeadando.controllers.ReadController">  
    <TabPane fx:id="tabPane" prefWidth="600" prefHeight="400">  
        <Tab text="Jelentkezés">  
            <TableView fx:id="recordTable" VBox.vgrow="ALWAYS">  
                <columns>  
                    <!--Jelentkezés oszlopok-->  
                </columns>  
            </TableView>  
        </Tab>  
        <Tab text="Képzés">  
            <TableView fx:id="kepzesTable" VBox.vgrow="ALWAYS">  
                <columns>  
                    <!--Képzés oszlopok-->  
                </columns>  
            </TableView>  
        </Tab>  
        <Tab text="Jelentkező">  
            <TableView fx:id="jelentkezoTable" VBox.vgrow="ALWAYS">  
                <columns>  
                    <!--Jelentkező oszlopok-->  
                </columns>  
            </TableView>  
        </Tab>  
    </TabPane>  
</StackPane>
```

Feladathoz tartozó controller: `ReadController.java`

A view-ban használt elemek először is inicializálva lettek a controllerben.

```
public class ReadController {

    @FXML
    private TableView<Record> recordTable;

    @FXML
    private TableView<kepzesRecord> kepzesTable;

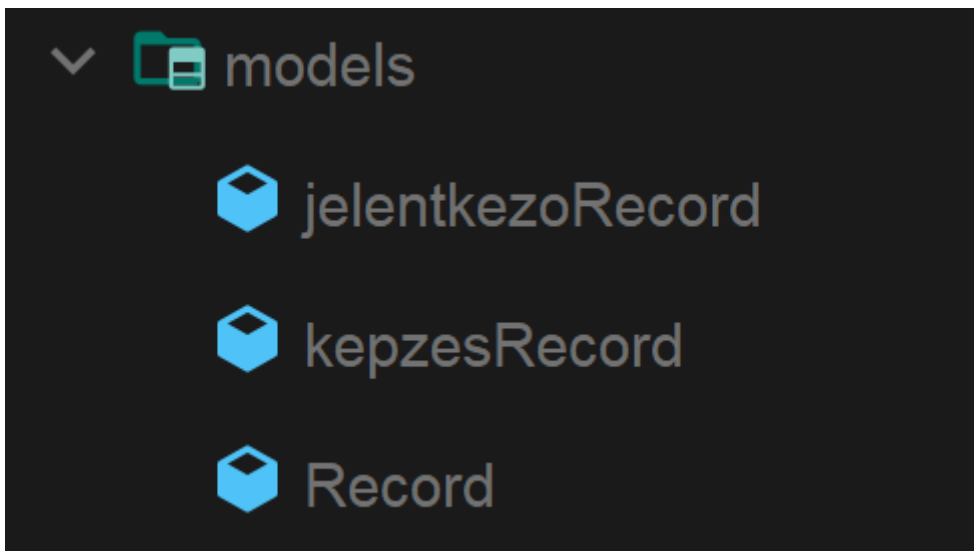
    @FXML
    private TableView<jelentkezoRecord> jelentkezoTable;

    @FXML
    private TableColumn<Record, Integer> jelentkezoIdColumn;

    @FXML
    private TableColumn<Record, String> jelentkezoNevColumn;

    ...
}
```

Mindhárom táblának létrehoztunk egy-egy osztályt, amikben az adatok tárolva vannak.



A táblák szintén a controllerben kerültek feltöltésre értékekkel. A példa kedvéért a „Jelentkező” táblát mutatom be az alábbi kódrészben:

```
@FXML
public void initialize() {
    jelentkezoTable.setColumnResizePolicy(TableView.CONSTRAINED_RESIZE_POLICY);
    VBox.setVgrow(jelentkezoTable, Priority.ALWAYS);

    jelentkezoId.setCellValueFactory(new PropertyValueFactory<>("id"));
    jelentkezoNev.setCellValueFactory(new PropertyValueFactory<>("nev"));
    jelentkezoNem.setCellValueFactory(new PropertyValueFactory<>("nem"));

    refreshTable();
}

public void refreshTable() {
    List<jelentkezoRecord> jelentkezoRecords = recordDAO.getAllJelentkezoRecords();
    jelentkezoTable.getItems().setAll(jelentkezoRecords);
}
}
```

A valós kódban mindhárom tábla, és a hozzá szükséges lekérdezések szerepelnek, de ugyan azt a logikát követik.

Az adatbázisban való lekérdezést pedig a recordDAO fájlban végezzük el:

```
public List<jelentkezoRecord> getAllJelentkezoRecords() {
    List<jelentkezoRecord> jelentkezoRecord = new ArrayList<>();

    String query = "SELECT * FROM jelentkezo ORDER BY id ASC";

    try (Connection conn = DriverManager.getConnection(DB_URL);
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(query)) {

        while (rs.next()) {
            int id = rs.getInt("id");
            String nev = rs.getString("nev");
            String nem = rs.getString("nem");
            System.out.println(nev + " " + nem + " " + id);

            jelentkezoRecord.add(new jelentkezoRecord(id, nev, nem));
        }
    } catch (Exception e) {
        e.printStackTrace();
    }

    return jelentkezoRecord;
}
```

Itt végezzük el a „SELECT \*” műveletet az adott táblára vonatkozóan, jelen esetben a „jelentkezo” táblára. A bekért adatokat egy listában adjuk vissza a controllernek, hogy a táblázatba be tudja tölteni az értékeket.

### 3.2.2. Olvas2 részfeladat

Ennek a feladatnak az elkészítése két nagyobb részből állt. Az egyik a mezők megjelenítése amin a szűrőket be tudjuk állítani és ezeket az adatokat kezeljük, a másik pedig az adatokkal való műveletek elvégzése, amikor is a szűrőknek megfelelően kilistázzuk a megfelelő adatokat.

Feladathoz tartozó view: `read2-view.fxml`

```
<StackPane xmlns:fx="http://javafx.com/fxml"
fx:controller="nje.ea.eabeadando.controllers.Read2Controller">

    <!-- Keresési Form -->
    <BorderPane fx:id="searchPane">
        <center>
            <VBox spacing="10" alignment="CENTER">
                <!-- Szűrési lehetőségek ... -->
                <Button text="Keresés" maxWidth="150" onAction="#handleFilterAction"/>
            </VBox>
        </center>
    </BorderPane>

    <!-- Táblázat nézet -->
    <VBox fx:id="tablePane" spacing="10" alignment="CENTER" visible="false">
        <TableView fx:id="recordTable" VBox.vgrow="ALWAYS">
            <columns>
                <!-- ... Oszlopok ... -->
            </columns>
        </TableView>
    </VBox>
</StackPane>
```

```

        </columns>
    </TableView>
    <Button text="Új keresés" onAction="#handleNewSearchAction"/>
</VBox>

</StackPane>

```

Feladathoz tartozó controller: **Read2Controller.java**

A két állapot közötti váltást az alábbi kódrészlet kezeli:

```

private void showSearchPane() {
    searchPane.setVisible(true);
    tablePane.setVisible(false);
}

private void showTablePane() {
    searchPane.setVisible(false);
    tablePane.setVisible(true);
}

```

A Szűrési folyamat a következőképp történik, a `handleFilterAction` függvényen belül:

```

// Adatok szűrése
List<Record> allRecords = recordDAO.getAllRecords();
List<Record> filteredRecords = allRecords.stream()
    // Név szűrése (tartalmazza a keresett szöveget)
    .filter(record -> nevFilter.isEmpty() ||
record.getJelentkezoNev().toLowerCase().contains(nevFilter))
    // Képzés szűrése (egyezik a kiválasztott képzéssel)
    .filter(record -> kepzesFilter == null ||
record.getKepzesNev().equalsIgnoreCase(kepzesFilter))
    // Nem szűrés (fiú -> f, line -> l, mindegy -> nem szűrünk)
    .filter(record -> genderFilter == null ||
        (genderFilter.equals("f") &&
record.getJelentkezoNem().equalsIgnoreCase("f")) ||
        (genderFilter.equals("l") &&
record.getJelentkezoNem().equalsIgnoreCase("l")) ||
        genderFilter == null)
    // Pontszám szűrése (min. 150 pont)
    .filter(record -> !minPontSzures || record.getSzerzettPont() >= 150)
    .collect(Collectors.toList());

// Táblázat feltöltése
ObservableList<Record> observableList =
FXCollections.observableArrayList(filteredRecords);
recordTable.setItems(observableList);

// Nézet váltása táblázatra
showTablePane();

```

A `recordDAO.getAllRecords()` a nevéhez hűen egy olyan függvény, amely a jelentkezők táblából minden adatot lekér. Jelen esetben ezzel dolgozunk.

### 3.2.3. Ír részfeladat

A jelentkező és képzés tábla esetében egyszerű hozzáadás műveletekről beszélünk. A szöveges mezőkbe beírjuk az értékeket, a felsorolt értékek közül kiválasztunk valamit a ComboBox-ban, rákattintunk a hozzáadásra, és meg is vagyunk. A „Jelentkezés” tábla

esetében viszont dinamikusan kell betöltenünk az adatokat az adatbázisból, hiszen csak olyan képzés és jelentkező ID-t választhatunk ki, ami ezekben a táblákban már létezik.

Példaként így történik egy jelentkező hozzáadása:

Feladathoz tartozó view: `ir-view.fxml`

```
<StackPane xmlns="http://javafx.com/fxml" xmlns:fx="http://javafx.com/fxml"
fx:controller="nje.ea.eabeadando.controllers.IrController">
    <TabPane fx:id="tabPane" prefWidth="600" prefHeight="400">
        <!-- Jelentkező Tab -->
        <Tab text="Jelentkező">
            <VBox spacing="10" alignment="CENTER">
                <TextField fx:id="nameField" promptText="Jelentkező neve" maxWidth="300"/>
                <ComboBox fx:id="genderComboBox" promptText="Nem" maxWidth="300">
                    <items>
                        <FXCollections fx:factory="observableArrayList">
                            <String fx:value="Fiú"/>
                            <String fx:value="Lány"/>
                        </FXCollections>
                    </items>
                </ComboBox>
                <Button text="Jelentkező hozzáadása" onAction="#handleAddJelentkezo"
maxWidth="150"/>
            </VBox>
        </Tab>
    </TabPane>
</StackPane>
```

Feladathoz tartozó controller: `IrController.java`

A Controllerben a hozzáadást ez a függvény kezeli:

```
@FXML
private void handleAddJelentkezes() {
    // Jelentkezés adatainak lekérése
    String applicantName =
recordDAO.getApplicantNameById(Integer.parseInt(applicantComboBox.getValue()));
    String courseName =
recordDAO.getKepzesNameById(Integer.parseInt(courseComboBox.getValue()));
    String rankText = rankField.getText();
    String scoreText = scoreField.getText();

    // Ellenőrizzük, hogy minden adatot megadtak-e
    if (applicantName == null || applicantName.isEmpty() || courseName == null ||
courseName.isEmpty() || rankText == null || rankText.isEmpty() || scoreText == null ||
scoreText.isEmpty()) {
        System.out.println("Kérlek töltsd ki az összes mezőt!");
        return;
    }

    try {
        int rank = Integer.parseInt(rankText);
        int score = Integer.parseInt(scoreText);

        // Jelentkező ID és Képzés ID lekérése
        int applicantId = recordDAO.getApplicantIdByName(applicantName);
        int courseId = recordDAO.getCourseIdByName(courseName);

        // Hozzáadjuk a jelentkezést az adatbázishoz
        recordDAO.addJelentkezes(applicantId, courseId, rank, score);
        System.out.println("Jelentkezés hozzáadva: ID: " + applicantId + " Kurzus: " +
courseId + " Rangsor: " + rank + " Pontszám: " + score);
        // Mezők tisztítása
        applicantComboBox.getSelectionModel().clearSelection();
        courseComboBox.getSelectionModel().clearSelection();
        rankField.clear();
    }
```

```

        scoreField.clear();

    } catch (NumberFormatException e) {
        System.out.println("Kérem, válassza ki a helyes számformátumot a sorrend és szerzett pont mezőknél!");
    }
}

```

### 3.2.4. Módosít részfeladat

Az módosítást igyekeztünk minél inkább felhasználó baráttra tervezni, mindezt úgy, hogy a feladat követelményének is megfeleljen.

A legfelső legördülő menü azért felel, hogy kiválasszuk azt az ID-t, amihez tartozó adatokkal dolgozni szeretnénk. Ez megfelel lényegében első lépésben egy „get by ID” műveletnek, a lentebbi field-ekbe betöltjük azokat az adatokat, amik a rekordhoz tartoznak.

Az alábbi kódban a „képzés” táblán fogom ezt bemutatni, mivel ismét mindhárom táblán lehetséges a módosítás műveletet elvégezni. Fontos megjegyezni, hogy a „jelentkezes” táblában ez kifejezetten bonyolultabb megoldást igényelt, mivel összetett kulcsról beszélünk, magyarul egyszerű „Id” alapján való betöltés nem létezik.

Feladathoz tartozó view: **modosit-view.fxml**

```

<!-- Képzés Tab -->
<Tab text="Képzés">
    <VBox spacing="10" alignment="CENTER">
        <ComboBox fx:id="kepzesComboBox" promptText="Képzés" maxWidth="300"
onAction="#handleKepzesSelection"/>
        <TextField fx:id="courseNameField" promptText="Képzés neve" maxWidth="300"/>
        <TextField fx:id="courseCapacityField" promptText="Felvehető létszám"
maxWidth="300"/>
        <TextField fx:id="courseMinScoreField" promptText="Minimum pontszám"
maxWidth="300"/>
        <Button text="Képzés módosítása" onAction="#handleKepzesModify"
maxWidth="150"/>
    </VBox>
</Tab>

```

Feladathoz tartozó controller: **ModositController.java**

Először is betöltjük az értékeket a mezőkre. Ez akkor történik meg, amikor a felhasználó a ComboBox-ból kiválaszt egy értéket.

```

private void loadKepzesData(int id) {
    int courseId = id;
    if (courseId != -1) {
        courseNameField.setText(recordDAO.getCourseNameById(id));
        courseCapacityField.setText(recordDAO.getCourseCapacityById(id));
        courseMinScoreField.setText(recordDAO.getCourseMinscoreById(id));
    }
}

```

Ahogy láthatjuk, az értékek betöltése adatbázis lekérdezéseken keresztül történik a recordDAO függvényeit meghívva. Például a `getCourseNameById` függvény a

következőképp néz ki a recordDAO fájlban. Paraméterül megadjuk az ID-t, és return-öljük a nevet után megtörtént a lekérdezés:

```
public String getCourseNameById(int courseId) {
    String courseName = "";
    String query = "SELECT nev FROM kepzés WHERE id = ?";

    try (Connection conn = DriverManager.getConnection(DB_URL);
        PreparedStatement stmt = conn.prepareStatement(query)) {

        stmt.setInt(1, courseId); // Az ID-t állítjuk be
        ResultSet rs = stmt.executeQuery();

        if (rs.next()) {
            courseName = rs.getString("nev"); // A nev oszlopot kérjük le
        }
    } catch (Exception e) {
        e.printStackTrace();
    }

    return courseName;
}
```

Miután a betöltött értékeket átírtuk, és a módosítás gombra kattintunk, a következő függvény hajtja végre a változtatásokat:

```
@FXML
private void handleKepzesModify() {
    int id = Integer.parseInt(kepzésComboBox.getValue());
    String courseName = courseNameField.getText();
    String capacity = courseCapacityField.getText();
    String minScore = courseMinScoreField.getText();
    if (id != -1) {
        boolean success = recordDAO.updateKepzes(id, courseName,
Integer.parseInt(capacity), Integer.parseInt(minScore));
        if (success) {
            System.out.println("Képzés módosítva: " + courseName + ", " + capacity + ", "
+ minScore);
        }
    }
}
```

### 3.2.5. Töröl részfeladat

A törlés nagyon hasonlóan működik a módosításhoz. Kiválasztjuk az ID-t, amivel után az adatokat megtekintésre betöltjük a felhasználónak. Módosításra viszont nem adunk lehetőséget, az értékeket „disable” taggel látjuk el.

Feladathoz tartozó view: `torol-view.fxml`

```
<Tab text="Képzés">
    <VBox spacing="10" alignment="CENTER">
        <ComboBox fx:id="kepzésComboBox" promptText="Képzés" maxWidth="300"
onAction="#handleKepzesSelection"/>
        <TextField fx:id="courseNameField" promptText="Képzés neve" maxWidth="300"
disable="true"/>
        <TextField fx:id="courseCapacityField" promptText="Felvehető létszám"
maxWidth="300" disable="true"/>
        <TextField fx:id="courseMinScoreField" promptText="Minimum pontszám"
maxWidth="300" disable="true"/>
        <Button text="Képzés törlése" onAction="#handleKepzesDelete" maxWidth="150" />
    </VBox>
</Tab>
```



Feladathoz tartozó controller: **TorolController.java**

A gombra kattintva nem módosítást, hanem „delete by id” sql parancsot futtatunk.

```
@FXML
private void handleKepzesDelete() {
    int id = Integer.parseInt(kepzesComboBox.getValue());
    String courseName = courseNameField.getText();
    String capacity = courseCapacityField.getText();
    String minScore = courseMinScoreField.getText();
    if (id != -1) {
        boolean success = recordDAO.deleteKepzes(id);
        if (success) {
            System.out.println("Képzés törölve: " + courseName + ", " + capacity + ", " +
minScore);
        }
    }
}
```

### 3.3. SOAP feladat

#### 3.3.1. Letöltés részfeladat

Mivel az MNB Soap kliens nem működik jól a JavaFX legújabb verziójával, így ezt a feladatot egy külön konzolos Java alkalmazás segítségével írtuk meg, az előadáson elhangzottak mentén. Az alkalmazást ezután JAR fájlként csomagoltuk, a JavaFX alkalmazásunk pedig ezt a JAR fájlt használja fel a funkció megvalósítása során. Ez a segédprogram parancssori paraméterek mentén kéri le a várt adatokat az MNB API-tól és menti el azokat az **mnb.txt** fájlba a **c:\adatok** mappában. Amennyiben nem kap parancssori paramétereket, konzolról kéri be azokat.

A Main metódusban valósul meg a SOAP adatfolyam és az adatmentés:

```
// Az MNB árfolyam szolgáltatás implementációjának példánya
MNBArfolyamServiceSoapImpl impl = new MNBArfolyamServiceSoapImpl();
// SOAP interfész lekérése
MNBArfolyamServiceSoap service = impl.getCustomBindingMNBArfolyamServiceSoap();

// Az eredmények tárolására egy StringBuilder objektumot használunk
StringBuilder output = new StringBuilder();

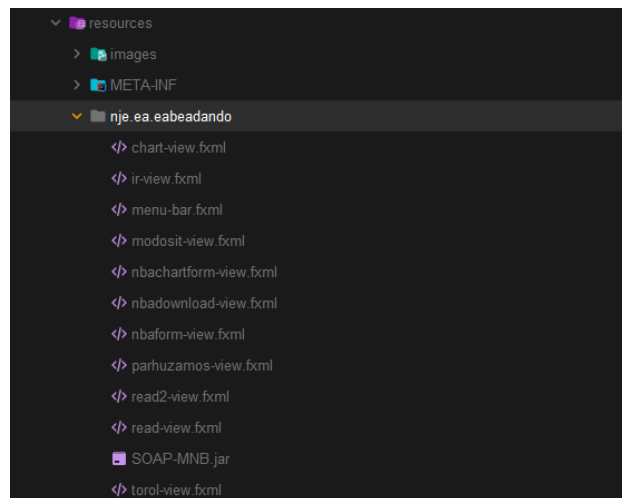
try {
    // Árfolyamok lekérése a megadott paraméterek alapján
    String exchangeRates = service.getExchangeRates(startDate, endDate, currency);

    // Az adatokat hozzáadjuk a StringBuilderhez
    output.append(currency).append("\n").append(parseExchangeRates(exchangeRates)).append("\n");

    // A fájl elérési útja
    String filePath = "c:\\adatok\\mnb.txt";

    // Az adatokat kiírjuk a fájlba
    Files.write(Paths.get(filePath), output.toString().getBytes(),
StandardOpenOption.CREATE, StandardOpenOption.TRUNCATE_EXISTING);
}
```

A JavaFX alkalmazásban ez a JAR a *resources* mappának a része, ezen az útvonalon kerül meghívásra:



Mivel a *Letöltés2* és *grafikon* részfeladatok is megkövetelik ezt az alapfunkcionalitást, így a JAR-t kezelő állományt kiszerveztük egy statikus metódusba, ezzel elkerülve a kódismétlést és a logikai redundanciát.

Ez a funkció az **MNB** osztály *getData* statikus metódusában került implementálásra, itt a *processBuilder()* segítségével használjuk a JAR-t:

```
ProcessBuilder processBuilder = new ProcessBuilder("java", "-jar", jarPath, currency,
startDate, endDate);

// Parancs futtatása
Process process = processBuilder.start();
int exitCode = process.waitFor(); // Várakozás a folyamat befejezésére

if (exitCode != 0) {
    throw new RuntimeException("SOAP-MNB.jar futtatása sikertelen! Hibakód: " +
exitCode);
}
```

Ez a megoldás IntelliJ környezetben teljes értékűen működik, azonban mikor futtatható verziót állítottuk össze rádöbbenünk, hogy a becsomagolt JAR állományon belül ez a konzol applikáció JAR nem érhető el és nem futtatható, így a projekt build verzióját módosítanunk kellett, mégpedig úgy, hogy ez a függőség ideiglenesen betöltésre kerüljön futási időben.

Tehát a build verzióban az osztály az alábbi módon hívja meg ez a függőséget:

```
try {
    // Az erőforrás elérési útja a JAR-ban
    String jarResourcePath = "/nje/ea/eabeadando/SOAP-MNB.jar";
    File tempJarFile = File.createTempFile("SOAP-MNB", ".jar");
    tempJarFile.deleteOnExit(); // Ideiglenes fájl automatikus törlése a program
    kilépésekor

    // A JAR-ból másoljuk ki a másik JAR fájlt egy ideiglenes fájlba
```

```

try (InputStream inputStream = MNB.class.getResourceAsStream(jarResourcePath);
    FileOutputStream outputStream = new FileOutputStream(tempJarFile)) {
    if (inputStream == null) {
        throw new IOException("Nem található az erőforrás: " + jarResourcePath);
    }
    byte[] buffer = new byte[1024];
    int bytesRead;
    while ((bytesRead = inputStream.read(buffer)) != -1) {
        outputStream.write(buffer, 0, bytesRead);
    }
}

```

```

// ProcessBuilder a másik JAR futtatására
ProcessBuilder processBuilder = new ProcessBuilder("java", "-jar",
tempJarFile.getAbsolutePath(), currency, startDate, endDate);

// Parancs futtatása
Process process = processBuilder.start();
int exitCode = process.waitFor(); // Várakozás a folyamat befejezésére

```

Ezzel a statikus osztállyal kommunikál az *NBADownloadController*, ami köztes réteget képez a megjelenítésért felelős *nbadownload-view.fxml* és az *NBA* osztály között. Mivel a *Letoltes1* feladatban még nem kérünk be felhasználtól adatokat a lekéréshez, így a *controller* logikája is leegyszerűsödik, figyeli a gomb lenyomását és előre meghatározott értékekkel lekéri az adatokat a kientstől.

```

public void onDownloadClicked(ActionEvent actionEvent) {
    try {
        // 1. Mai és három nappal ezelőtti dátum kiszámítása
        LocalDate today = LocalDate.now();
        LocalDate threeDaysAgo = today.minusDays(3);

        // Dátumok formázása
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");
        String todayStr = today.format(formatter);
        String threeDaysAgoStr = threeDaysAgo.format(formatter);

        // Kérés indítás
        MNB.getData("EUR", threeDaysAgoStr, todayStr, true);
    }
}
...

```

A *nbadownload-view.fxml* view szerkezete igencsak egyszerű, egy kiírásból és egy nyomógombból áll:

```

<VBox xmlns="http://javafx.com/javafx" xmlns:fx="http://javafx.com/fxml"
fx:controller="nje.ea.eabeadando.controllers.NBADownloadController"
    alignment="CENTER" spacing="10" style="-fx-padding: 20;">
    <Label text="Elmúlt 2 napi EUR árfolyam letöltése" style="-fx-font-size: 16px; -fx-text-
fill: #333; -fx-font-weight: bold;" />
    <Button text="Letöltés" onAction="#onDownloadClicked" style="-fx-font-size: 18px; -fx-
padding: 10;" />
</VBox>

```

### 3.3.2. Letöltés2 részfeladat

A *Letöltés2* részfeladatban, ha bár a funkcionalitás megegyező, már a felhasználói bemenet alapján kell az adatok lekérnünk így a *view* is komplexebb lesz, valamint a *controller*-ben is több FXML elemet kell regisztrálnunk és validációs esetet kell kezelnünk.

Az űrlap megjelenítéséért a *nbachartform-view.fxml* felel:

```
<VBox xmlns="http://javafx.com/javafx" xmlns:fx="http://javafx.com/fxml"
fx:controller="nje.ea.eabeadando.controllers.NBACHartFormController"
    alignment="CENTER" spacing="15" style="-fx-padding: 20;">
    <Label text="MNB Árfolyam Megjelenítő" style="-fx-font-size: 18px; -fx-font-weight: bold;"
/>
    <HBox alignment="CENTER" spacing="10">
        <Label text="Valuta típusa:" style="-fx-font-size: 14px;" />
        <ComboBox fx:id="currencyComboBox" promptText="Válassz devizát" />
    </HBox>
    <HBox alignment="CENTER" spacing="10">
        <Label text="Kezdődátum:" style="-fx-font-size: 14px;" />
        <DatePicker fx:id="startDatePicker" />
    </HBox>
    <HBox alignment="CENTER" spacing="10">
        <Label text="Végdátum:" style="-fx-font-size: 14px;" />
        <DatePicker fx:id="endDatePicker" />
    </HBox>
    <Button text="Letöltés" onAction="#onDownloadClicked" style="-fx-font-size: 16px; -fx-
padding: 10 20;" />
</VBox>
```

A hozzátartozó *NBACHartFormController* inicializáljuk a *Combobox*-t, veszünk át adatokat és végzünk validációt:

```
public class NBAFormController {
    @FXML
    private ComboBox<String> currencyComboBox;

    @FXML
    private DatePicker startDatePicker;

    @FXML
    private DatePicker endDatePicker;

    @FXML
    public void initialize() {
        currencyComboBox.getItems().addAll("EUR", "USD", "GBP", "CHF", "JPY");
    }

    public void onDownloadClicked(ActionEvent actionEvent) {
        try {
            // 1. Inputok validálása
            String currency = currencyComboBox.getValue();
            if (currency == null || currency.isEmpty()) {
                throw new IllegalArgumentException("Kérjük, válasszon egy valutát a
listából!");
            }

            LocalDate startDate = startDatePicker.getValue();
            LocalDate endDate = endDatePicker.getValue();
            if (startDate == null) {
                throw new IllegalArgumentException("Kérjük, válasszon egy kezdődátumot!");
            }
            if (endDate == null) {
                throw new IllegalArgumentException("Kérjük, válasszon egy végdátumot!");
            }
        }
    }
}
```

```

        if (startDate.isAfter(endDate)) {
            throw new IllegalArgumentException("A kezdődátum nem lehet későbbi, mint a
végdátum!");
        }

        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");
        String startDateStr = startDate.format(formatter);
        String endDateStr = endDate.format(formatter);

        MNB.getData(currency, startDateStr, endDateStr, true);

        ...

```

Ha a kapott bemenet formátuma helyes, akkor az előző feladathoz hasonlóan lekérjük az adatokat.

Ellenkező esetben értesítjük a felhasználót:

```

    } catch (IllegalArgumentException e) {
        // Felhasználói hibák kezelése
        showAlert(Alert.AlertType.WARNING, "Figyelem!", e.getMessage());
    } catch (Exception e) {
        // Egyéb hibák kezelése
        showAlert(Alert.AlertType.ERROR, "Hiba", "Valami hiba történt: " +
e.getMessage());
    }
}

```

### 3.3.3. Grafikon részfeladat

A grafikon részfeladat is támaszkodik az előző két funkcióra, azonban itt már nem csak az adatok letöltése a feladat, hanem az árfolyamot ki is kell rajzoltatnunk JavaFX Chart segítségével. A megvalósítás az árfolyam adatainak lekéréséig megegyezik az előző pontokban leírtakkal, ugyan azt az űrlapot használjuk és ugyan úgy az **MNB** osztályon keresztül kérjük és töltjük le az adatokat, azonban ez után a felhasználót átirányítjuk a *chart-view.fxml* oldalra, ahol a grafikont kirajzoljuk.

A *chart-view.fxml* oldal a következőképp került felépítésre:

```

<BorderPane xmlns="http://javafx.com/javafx" xmlns:fx="http://javafx.com/fxml"
fx:controller="nje.ea.eabeadando.controllers.ChartController">
    <center>
        <LineChart fx:id="lineChart" title="Árfolyamok"
            legendVisible="false">
            <xAxis>
                <CategoryAxis fx:id="xAxis" />
            </xAxis>
            <yAxis>
                <NumberAxis fx:id="yAxis" label="Árfolyam" />
            </yAxis>
        </LineChart>
    </center>
    <bottom>
        <HBox alignment="CENTER" spacing="10" style="-fx-padding: 10;">
            <Button text="Vissza a menübe" onAction="#onMenuRedirectClicked" />
        </HBox>
    </bottom>

```

```
</BorderPane>
```

A view-hoz tartozó *ChartController*-ben regisztráltuk a vezérlőket és definiáltuk a forrás állományt:

```
public class ChartController {

    @FXML
    private LineChart<String, Number> lineChart;

    @FXML
    private CategoryAxis xAxis;

    @FXML
    private NumberAxis yAxis;

    public void initialize() {
        // Az mnb.txt fájl útvonala
        String filePath = "c:\\adatok\\mnb.txt";

        // Ellenőrizzük, hogy létezik-e a fájl
        File file = new File(filePath);
        if (file.exists()) {
            // Fájl létezik: dolgozzuk fel
            processFile(filePath);
        } else {
            // Fájl nem létezik: kérjünk adatokat az MNB API-tól
            requestDataFromMNB();
        }
    }
}
```

Ha az állomány létezik (MNB.txt) akkor beolvassuk és feldolgozzuk az adatokat majd hozzáadjuk a Chart-hoz:

```
private void processFile(String filePath) {
    try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
        // Az első sor a valuta típusa
        String currencyType = br.readLine();
        xAxis.setLabel("Dátum");
        yAxis.setLabel(currencyType + " Árfolyam");
        lineChart.setTitle(currencyType + " Árfolyamok"); // A grafikon címének beállítása

        // Új széria létrehozása
        XYChart.Series<String, Number> series = new XYChart.Series<>();
        series.setName(currencyType + " Árfolyamok");

        // Az adatok feldolgozása
        String dateLine;
        while ((dateLine = br.readLine()) != null) {
            String valueLine = br.readLine(); // Az érték az aktuális dátum után
            if (valueLine != null) {
                String date = dateLine.trim();
                double value = Double.parseDouble(valueLine.trim().replace(",", "."));
                // Adatok hozzáadása a szériához
                series.getData().add(new XYChart.Data<>(date, value));
            }
        }

        // Széria hozzáadása a charthoz
        lineChart.getData().add(series);
    }
}
```

Ha nem létezik a forrás állományunk akkor lekérjük az elmúlt három nap EUR árfolyamát:

```
private void requestDataFromMNB() {
    try {
        // Mai és három nappal ezelőtti dátum számítása
        LocalDate today = LocalDate.now();
        LocalDate threeDaysAgo = today.minusDays(3);

        // Dátumok formázása
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");
        String todayStr = today.format(formatter);
        String threeDaysAgoStr = threeDaysAgo.format(formatter);

        // API hívás
        MNB.getData("EUR", threeDaysAgoStr, todayStr, false);
    } catch (Exception e) {
        e.printStackTrace();
        System.err.println("Hiba az MNB API meghívása közben!");
    }
}
```

### 3.4. Párhuzamos feladat

Feladathoz tartozó controller: **ParhuzamosController.java**

A párhuzamos feladat megvalósítása 2 időzítő létrehozásával történt, a „Timeline” használatával. Az egyik 1 másodpercenként inkrementálta a változó értékét, a másik pedig 2 másodpercenként. Ezek egymástól függetlenül működtek.

```
@FXML
public void initialize() {
    // Időzítő a label1 frissítésére 1 másodpercenként
    Timeline timeline1 = new Timeline(
        new KeyFrame(Duration.seconds(1), event -> updateLabel1())
    );
    timeline1.setCycleCount(Timeline.INDEFINITE);
    timeline1.play();

    // Időzítő a label2 frissítésére 2 másodpercenként
    Timeline timeline2 = new Timeline(
        new KeyFrame(Duration.seconds(2), event -> updateLabel2())
    );
    timeline2.setCycleCount(Timeline.INDEFINITE);
    timeline2.play();
}

// label1 szövegének frissítése
private void updateLabel1() {
    label1.setText("Label 1: " + counter1++);
}

// label2 szövegének frissítése
private void updateLabel2() {
    label2.setText("Label 2: " + counter2++);
}
}
```

Feladathoz tartozó view: **ParhuzamosController.java**

A viewban pedig nagyon egyszerűen létrehoztunk 2 label-t, amit kisebb formázással megnöveltünk, és vastagítottunk:

```
<Label fx:id="label1" text="Label 1: ">
  <font>
    <Font name="System Bold" size="24" />
  </font>
  <alignment>CENTER</alignment>
</Label>
<Label fx:id="label2" text="Label 2: ">
  <font>
    <Font name="System Bold" size="24" />
  </font>
  <alignment>CENTER</alignment>
</Label>
```