

Password Manager Project

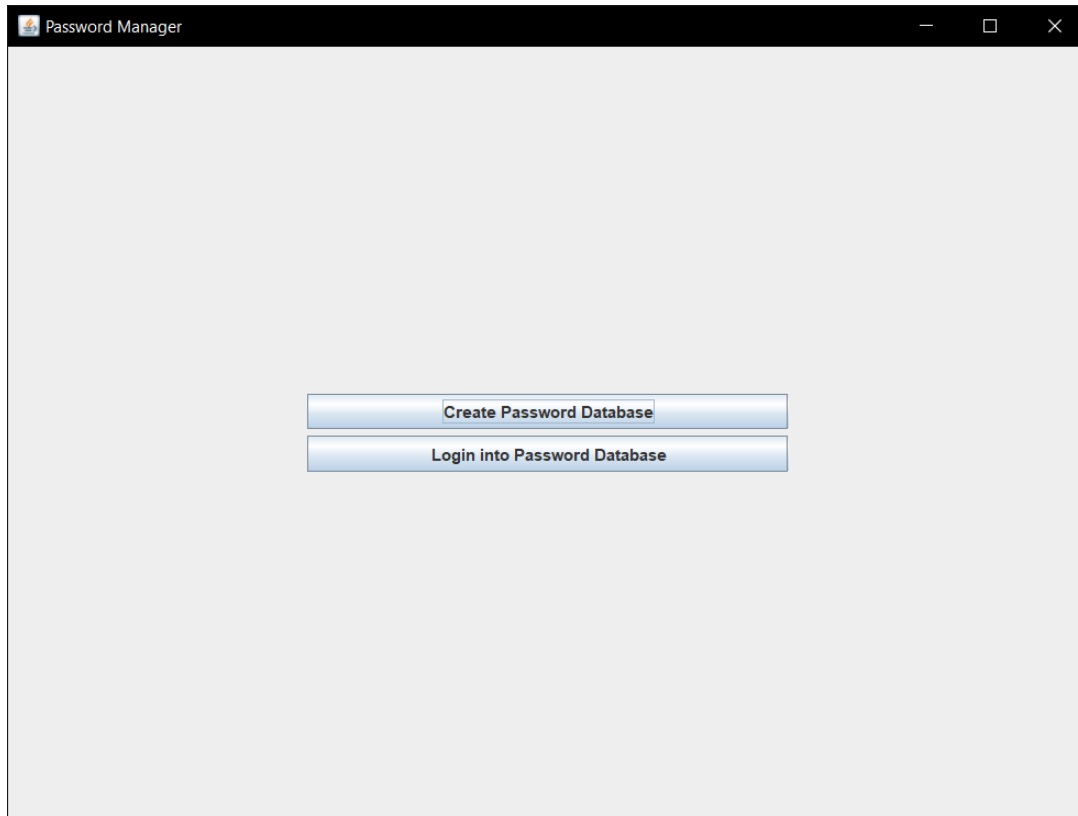
Sipos Szilárd

- Általános leírás:

A projekt ötletem onnan jött, hogy ma napságban a digitális biztonság egy utógondolat a legtöbb embernek, emiatt újra használnak jelszavakat, amik személyes információkat tartalmaz, mint a születésnapjuk, valamilyen fontos dátum az életükben, kedvenc állatuk neve és stb. Röviden nem biztonságos, személyes információt tartalmazó jelszavakat használnak fel újra és újra minden weboldalnál, ha véletlenül egy adat szivárgás történik egy cégnél, ahol nem tartás be a standard személyes információ, jelszó tárolását akkor a jelszót meglehet kapni nyers szöveggént, ahol próbálkozhatnak mindenféle különböző weboldalra bejelentkezni. Ha a jelszó személyes információkon alapszik akkor, ha valaki eléggé ismeri a célzott személyt akkor sok próbálkozással lehet könnyedén megkaphassák a jelszót. A fenti okok miatt hasznos egy jelszó kezelő program, ami a lokálisan tartalmazza a jelszavakat titkosítva egy mesterjelszóval. A program így megkönnyebbíti az emberek életét, hogy tudják sikeresen gyakorolni a helyes digitális biztonságot. Más ihletem az volt, hogy érdekltek, hogy hogyan is működnek a háttérben az ilyen programok és örvendek, hogy ilyen témával tudtam foglalkozni, mert érdekes dolgokat tudtam tanulmányozni. A projektnek a technikai részeit majd lennebb lehet megkapni.

- Felhasználói Utasítás:

Amikor először indítsuk el a programot megjelenik a főmenü, amin két gomb lesz.



“Create Password Database” gombbal létrehozzunk egy jelszó adatbázist, amiben meg kell adjuk a fájl nevét és annak a mester jelszavát. Figyelembe kell venni, hogy a jelszónak vannak feltételei. A jelszó hossza 3-24 karakter lehet, kell tartalmazzon egy nagybetűt, egy kisbetűt, egy számot, és egy speciális karaktert, mint "@?;" stb. A fájlt létre lesz hozva egy Data nevű mappába és a fájl formátuma .db típusú lesz. Ha véletlenül nem megfelelő a fájl név akkor a Create gomb alatt kilesz írva a hiba, hogy rossz a jelszó vagy fájl neve. A Back gombbal pedig visszalépünk a főmenübe.

Back

Enter the name for your file

Please enter the master password to access the file

Create

Password Manager

Back

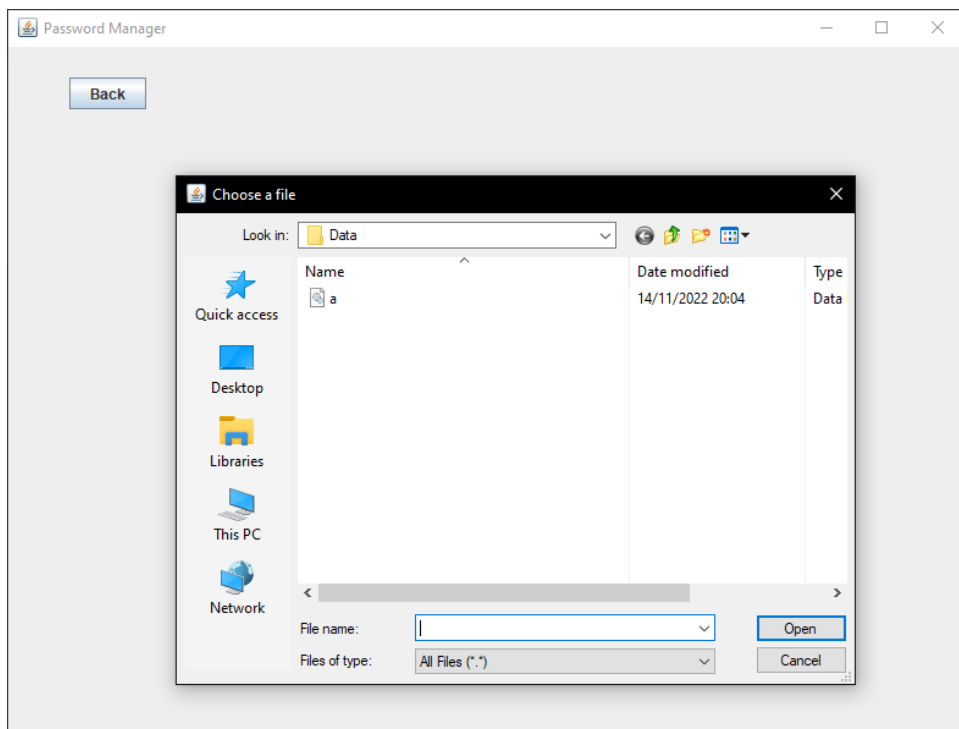
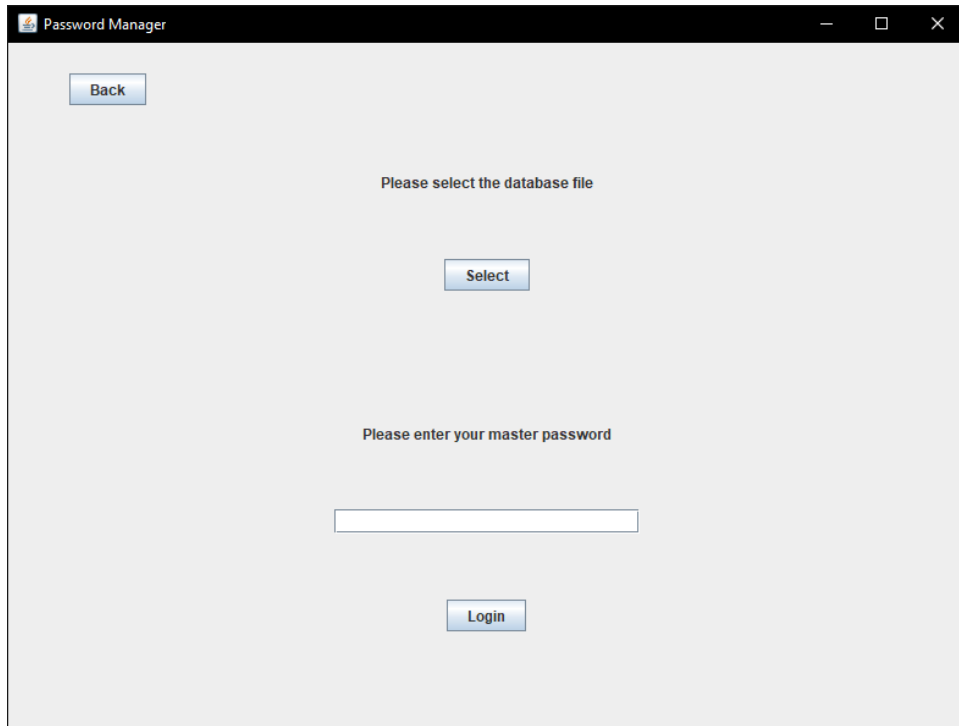
Enter the name for your file

Please enter the master password to access the file

Create

File name invalid

“Login into Password Database” gombbal pedig be lehet jelentkezni egy bizonyos adatbázisba. Lesz egy Select gombbal, amivel kilehet a fájlt, amiben beszeretnénk jelentkezni. Ha megnyomjuk elő fog ugrani egy dialog, ami a Data mappa tartalmát mutassa meg ahonnan ki kell válasszuk majd a fájlt. Majd azután a jelszó mezőbe be kell írni a jelszót és megnyomni a login gombot. Ha nem válasz ki egy fájlt vagy rossz jelszót ír be a Login gomb alatt kiírja. A fájl csak a Data mappában lehet másképp nem fogja megkapni a program. A Back gombbal pedig visszalépünk a főmenübe.



Password Manager

Back

Please select the database file

Select

a.db

Please enter your master password

....

Login

Wrong password!

Ha létrehozunk egy adatbázist vagy sikeresen bejelentkezzünk egybe azután a jelszó kezelőbe fog dobni a bizonyos fájl tartalmaival. Ahogy láthassuk több gomb jelenik meg és a fájl tartalmát mutassa meg. Ha egy sor tartalmát szeretnénk kimásolni akkor rá kell klikkelni egy sorba CTRL + C és a clipboardba be lesz másolva a sor tartalma.

Password Manager

Back Save Set Master Password

Entry Internet Email General

Username	Password
lacika123	GmDwNqASKPVk1bHv{zw{q<@l
laszlo	Laszlo@E123

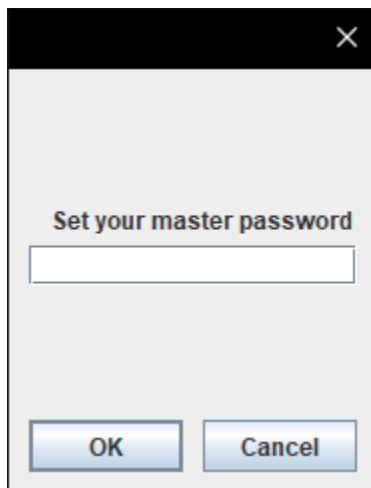
Add Edit Delete

Copy username Copy Password

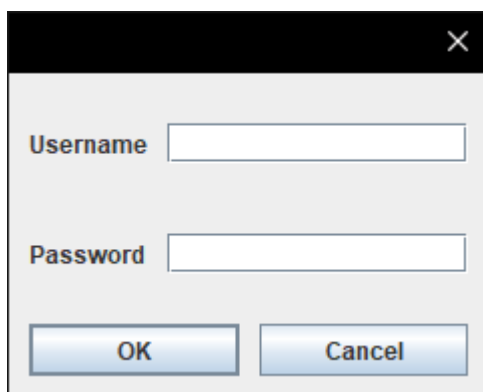
Back gomb visszadob minket a főmenübe és lementi a fájlt is nekünk.

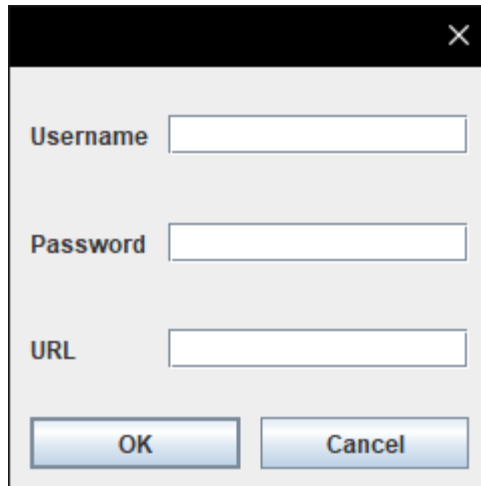
Save gombbal pedig manuálisan lementhessük a fájlt.

Set Master Password egy új mester jelszót állíthatunk be a fájlnak, ahol egy újabb dialog jelenik meg ahol be kell írjuk az új jelszót. A jelszónak ugyanaz a feltétele van, mint ahol létrehoztuk az adatbázist. A textfield ugyanúgy meg fog jeleni egy hiba szöveg, ha a jelszó nem megfelelő. Az OK gombbal létrehozza az adatbázist az új jelszóval a Cancel gombbal pedig nem viszi végbe.

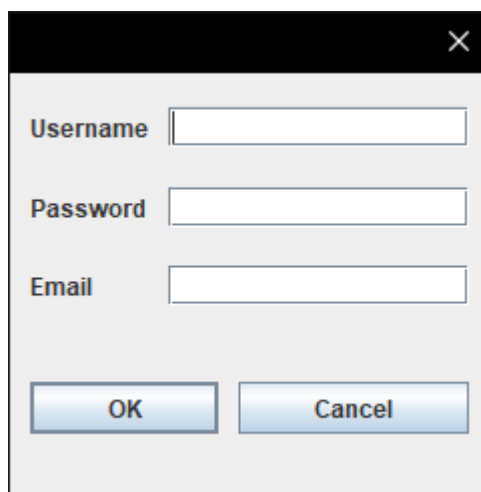
A dialog box titled "Set your master password" with a close button (X) in the top right corner. It contains a single text input field and two buttons at the bottom: "OK" and "Cancel".

Az Add gombbal pedig hozzáadunk egy újabb bejegyzést a csoporthoz. 4 féle csoport van Entry ahol a felhasználónevet és a jelszavát adhassuk meg, Internet, ahol a felhasználó nevet, jelszavát meg egy linket lehet megadni, Email felhasználónevet, jelszót és egy bizonyos email címet és General, ahol felhasználónevet, jelszavát meg egy szöveget lehet hozzáadni. Egy bizonyos feltétel, ami link, email és notes mezőbe kell figyelembe venni, hogy nem lehet szóközt bele tenni. Ha jelszó mezőnél nem fogunk írni semmit akkor a program generálni fog egy 24 karakter hosszúságú jelszót.

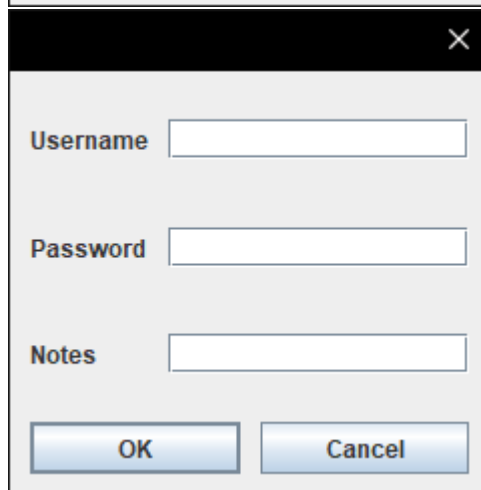
A dialog box with a close button (X) in the top right corner. It contains two labeled text input fields: "Username" and "Password". At the bottom, there are two buttons: "OK" and "Cancel".



A dialog box with a black title bar containing a close button (X). The main area has a light gray background. It contains three text input fields: 'Username', 'Password', and 'URL'. At the bottom, there are two buttons: 'OK' and 'Cancel'.



A dialog box with a black title bar containing a close button (X). The main area has a light gray background. It contains three text input fields: 'Username', 'Password', and 'Email'. At the bottom, there are two buttons: 'OK' and 'Cancel'.



A dialog box with a black title bar containing a close button (X). The main area has a light gray background. It contains three text input fields: 'Username', 'Password', and 'Notes'. At the bottom, there are two buttons: 'OK' and 'Cancel'.

Edit gomb ugyanúgy működik, mint az Add gomb azzal a különbséggel, hogy egy létező bejegyzést lehet módosítani. Ahhoz, hogy tudjunk módosítani egy bejegyzést rá kell klikkelni egyszer és megnyomni az Edit gombot a dialogban a sor mezői belesznek írva a dialog mezőbe.

- Érdekesebb kódrészlet:

Most két érdekesebb kódrészletet fogok bemutatni projektemből.

Az első dolog az, hogy hogyan mentem le a bejegyzéseket a fájlban a ThreadSaveFile osztály segítségével.

```
public class ThreadSaveFile extends Thread {  
  
    8 usages  
    private PasswordManagerModel model;  
  
    2 usages  
    private boolean isExit;  
  
    1 usage  
    public ThreadSaveFile(PasswordManagerModel model, boolean isExit) {  
        this.model = model;  
        this.isExit = isExit;  
    }  
  
    @Override  
    public void run() {  
        saveFileRunnable();  
    }  
  
    1 usage  
    private void saveFileRunnable() {  
        try {  
            FileHandler fileHandler = model.getFileHandler();  
            FileHandler temp = new FileHandler( filePaths: "temp.db");  
            temp.deleteFile();  
            temp.write(fileHandler.read( lineNumber: 1));  
            for (int groupIndex = 0; groupIndex < IDs.values().length; ++groupIndex) {  
                for (int entryIndex = 0; entryIndex < model.getSize(groupIndex); ++entryIndex) {  
                    Entry entry = model.getEntry(entryIndex, groupIndex);  
                    String writeline = model.getInstanceEnc().encrypt(entry.toString());  
                    temp.write(writeline);  
                }  
            }  
            fileHandler.deleteFile();  
            temp.rename(fileHandler.getFilePath());  
        } catch (FilePathIsNullException e) {  
            throw new RuntimeException(e);  
        } finally {  
            if (isExit) {  
                reset();  
            }  
        }  
    }  
}
```

```
    }  
}  
  
1 usage  
private void reset() {  
    model.getInstanceEnc().reset();  
    model.clearEntries();  
    model.getFileHandler().resetFilePath();  
}
```

ThreadSaveFile osztály örököli a Thread osztályt emiatt, ha véletlenül sok bejegyzésünk a felhasználó nem kell megvárja ameddig az összes bejegyzést lementi és továbbá is tud bejegyzéseket hozzáadni, módosítani, kimásolni és kitörölni. Egyedüli eset, ahol meg kell várni a lementést mikor visszalépünk a főmenübe mivel, ha véletlenül nincs minden lementve és a felhasználó próbál bejelentkezni akkor egy hibás adatbázisba fog belépni. Ez a PasswordMenuFormba van jelen amikor megnyomjuk a Back gombot úgy van leellenőrizve, hogy nézi, hogy ha a thread fut még és ha nem akkor visszavisz a főmenübe.

```
},
backButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        saveFile(isExit: true);
        resetTables();
        while (saveThread.isAlive()) {
            System.out.println("Still saving");
        }
    }
});
```

Most jöjjön az, hogy hogyan is menti le a fájlt. Mikor meghívjuk a saveFile metódust először megnézi, ha a saveThread, ami egy referencia ThreadSaveFile példányra, ha nem null akkor megnézi, hogy ha az a Thread fut még és ha nem akkor létrehoz egy újabb példányt. Szükséges egy újabb példány mivel a start metódus csak egyszer használható meglehetne hívni a run metódust, de az nem lenne multithreaded mivel a main threaden futna le.

```
private void saveFile(boolean isExit) {
    if (saveThread != null) {
        while (saveThread.isAlive()) {
        }
    }
    saveThread = new ThreadSaveFile(model, isExit);
    saveThread.start();
}
```

Miután a saveThreadet startoltuk akkor lefuttassa a run metódust. ThreadSaveFile run metódusában egy más metódust hívunk meg ami a saveFileRunnable.

```

private void saveFileRunnable() {
    try {
        FileHandler fileHandler = model.getFileHandler();
        FileHandler temp = new FileHandler( filePath: "temp.db");
        temp.deleteFile();
        temp.write(fileHandler.read( lineNumber: 1));
        for (int groupIndex = 0; groupIndex < IDs.values().length; ++groupIndex) {
            for (int entryIndex = 0; entryIndex < model.getSize(groupIndex); ++entryIndex) {
                Entry entry = model.getEntry(entryIndex, groupIndex);
                String writeLine = model.getInstanceEnc().encrypt(entry.toString());
                temp.write(writeLine);
            }
        }
        fileHandler.deleteFile();
        temp.rename(fileHandler.getFilePath());
    } catch (FilePathIsNullException e) {
        throw new RuntimeException(e);
    } finally {
        if (isExit) {
            reset();
        }
    }
}

```

Ahogy láthassuk metódusban megjelenik a try-catch-finally mivel a FileHandler van egy sajátos exceptionje ami a FilePathIsNullException amit csak akkor kap el, ha a fileHandlerben lévő referenciába a nincs beállítva egy fájl útvonal. A metódusban két FileHandler referenciánk van fileHandler ami a bizonyos adatbázis fájljal foglalkozik, meg a temp amiben az információkat fogjuk írni. Első sorban lesz mindig a mester jelszó titkosítva azt átmásoljuk fileHandler fájlból a temp fájlba. Azután beírjuk az összes Entry példányt toStringjét titkosítva. Ezt úgy csináljuk meg hogy a modellben van egy ArrayListben ArrayList ami a következő módon van elrendezve ahány különböző csoportunk van annyi listánk van és egy ID alapján vannak indexelve. Egy bizonyos arraylisten belül annak a csoportnak az entitásai kaphassuk meg például, ha az entriesnek ami arraylistben arraylist típusú, ha 0-dik ArrayListet szedjük elő akkor a 0-dik indexű csoportnak az entitieseit kapjuk meg benne. IDs.values().length megmondja hány csoportunk van és egy for ciklussal bejárjuk az összes csoportot IDs belül automatikusan van megadva a csoportnak az ID-je és 0-tól kezdődik. EntryIndexel,for-al pedig bejárjuk a bizonyos csoport listáját, ahol megkapjuk az entryket. Mindegyik bejegyzésnek Entry az alapsztálya és ha újabb csoportot hozzunk létre a kódban akkor az kell legyen az ősoosztálya. Itt be van mutatva a sokalakúság van bebizonyítva mivel mikor az entry.toString() metódust hívjuk meg akkor nem mindig ugyanazt kapjuk vissza, ha az entry a Entry gyerekosztályának a példánya akkor mást fog visszatéríteni, ami a gyerekosztályban található. Titkosítás meg úgy történik, hogy meghívjuk a modellnek az Encryption típusú példányát és meghívjuk az encrypt metódust, ami a mester jelszó alapján fogja titkosítani aztán ezt write metódussal beírjuk a fájlba. Itt nem kell megfigyelni, hogy a sor ismétlődik-e vagy már nem kellene létezzen mert a temp fájl mindig üres és a modell adatait írjuk bele. Miután beírtuk az összes az összes bejegyzést a modellnek a fájlt kitöröljük és a temp fájlt átevezzük arra a fájlnevére, ami a model FileHandler példánya foglalkozik. Ha az isExit igaz amikor akkor van, ha a PasswordMenuFormba visszalép a főmenüre mikor le van nyomva a Back gomb akkor a modellt egy tiszta állapotba hozza. Megszabadul a mostani mester jelszótól kitisztítsa a bejegyzéseket és a FileHandler referenciának a fájl útvonalát lenulláza. Így működik a mentés.

```

    }
}

1 usage
private void reset() {
    model.getInstanceEnc().reset();
    model.clearEntries();
    model.getFileHandler().resetFilePath();
}

```

Második kódrészletben bemutatom, hogyan készítsük elő PasswordMenuFormba a tabbedPanet.

```

private void createUIComponents() {
    tables = new ArrayList<JTable>();
    tabbedPane = new JTabbedPane(SwingConstants.TOP);
    for (IDs ID_value : IDs.values()) {
        DefaultTableModel model = new DefaultTableModel(ID_value.getColumns(), rowCount: 0) {
            @Override
            public boolean isCellEditable(int row, int column) {
                return false;
            }
        };
        JTable table = new JTable(model);
        table.setFillViewportHeight(true);
        table.getTableHeader().setResizingAllowed(false);
        table.setRowSelectionAllowed(true);
        tables.add(ID_value.getID(), table);
        tabbedPane.addTab(ID_value.getName(), new JScrollPane(table));
    }
}

```

Tables meg a tabbedPane attribútumok a tables egy referencia egy ArrayListre amiben lefogjuk menteni a csoportok tábláit és a tabbedPanebe megtalálható egy referencia egy új JTabbedPane példányra. Ezután egy for ciklussal megkapjuk az összes IDs értéket és egyenként létrehozuk a táblákat csoportonként. Mindegyik table egy referenciát fog tartalmazni egy JTable példányt, ami DefaultTableModel példányon fog alapozni, amiben megfelelő oszlopok lesznek létrehozva ID_value.getColumns metódussal és a cellák nem módosíthatóak meg. Beállításokat fogunk elvégezni, mint hogy csak sorokat lehet kiválasztani meg pár módosítás, hogy működjön rendesen JScrollPaneen belül. Miután megvan table hozzáadjuk a tables ArrayListbe és a csoport ID-ja alapján fogjuk indexelni. Azután a tabbedPanehez egy újabb tabként adjuk hozzá a neve ID_value.getname metódus fogja megadni és a komponens egy JScrollPane fog lenni, ami tartalmazza a táblát. Miután ez megvan és ha bejelelnkezzünk az adatbázisban akkor a LoginForm classben megtalálható lesz az initTable metódus.

```

public void initTable() {
    resetTables();
    for (IDs ID_value : IDs.values()) {
        DefaultTableModel model = (DefaultTableModel) tables.get(ID_value.getID()).getModel();
        switch (ID_value) {
            case ENTRY -> {
                addEntrytoModel(model);
                break;
            }
            case ENTRYINTERNET -> {
                addInternettoModel(model);
                break;
            }
            case ENTRYEMAIL -> {
                addEmailtoModel(model);
                break;
            }
            case ENTRYGENERAL -> {
                addGeneraltoModel(model);
                break;
            }
        }
    }
}
}

```

InitTable metódusban van egy resetTables metódus meg egy for ciklus, amivel bejárjuk a csoportokat.

```

private void resetTables() {
    for (JTable table : tables) {
        DefaultTableModel model = (DefaultTableModel) table.getModel();
        int rowIndex;
        while ((rowIndex = model.getRowCount() - 1) >= 0) {
            model.removeRow(rowIndex);
        }
    }
}

```

ResetTables metódusban fogja venni a táblákat a tables Arraylistből és egy while ciklussal hátulról kezdve kitöröli a legutolsó sort addig ameddig már nincs egy sor se a táblában.

Visszatérve az initTable metódusra ott mindegyik csoportnak, ahol csoportokként hozzáadjuk a DefaultTableModelhez a megfelelő információ, mivel model egy referencia, ami megtalálható a csoporthoz tartozó JTablehez emiatt a JTable automatikusan fogja mutatni a változásokat.

```

private void addEntrytoModel(DefaultTableModel model) {
    String[] data = new String[2];
    int groupIndex = IDs.ENTRY.getID();
    for (int indexEntry = 0; indexEntry < this.model.getSize(groupIndex); ++indexEntry) {
        Entry entry = this.model.getEntry(indexEntry, groupIndex);
        data[0] = entry.getUsername();
        data[1] = entry.getPassword();
        model.addRow(data);
    }
}

```

Csak egy ilyen metódust fogok bemutatni, mivel a többi hasonlít ehhez csak a többi csoportban lehet több mező van vagy kevesebb. Először is egy String tömböt hozzunk létre, amiben lefoglaljuk menteni a bejegyzéshez való információt. Azután a groupIndexxel amivel tudja, hogy listában kell keressük a bejegyzéseket azután egy for ciklussal bejárjuk annak a csoportnak a listáját. Megszerzi példányt a getEntry metódussal a this.model ami egy referencia egy PasswordManagerModel példányhoz. Miután megkaptuk a bejegyzést a data tömbbe betesszük a szükséges információt és hozzáadjuk a modellhez. A többi metódusok is hasonlóképpen működnek, de ahogy említettem az előbb csak lehet egy vagy több mezője lehet a DefaultTableModelnek. Így működik a tabbedPane létrehozása és beállítása.