ZÁRÓDOLGOZAT

Lautner Balázs

2018

Családi pénzügyi nyilvántartó alkalmazás

Lautner Balázs

SZÁMALK-Szalézi Szakgimnázium

esti Szoftverfejlesztő szak

KONZULENS: Kaczur Sándor

Nyilatkozat

a záródolgozat eredetiségéről

Alulírott ..... (név) /..... (anyja neve, szem.ig. szám) büntetőjogi és fegyelmi felelősségem tudatában kijelentem és aláírásommal igazolom, hogy a záródolgozat saját munkám eredménye. A felhasznált irodalmi és egyéb információs forrásokat az előírásoknak megfelelően kezeltem, a záródolgozat készítésre vonatkozó szabályokat betartottam.

Kijelentem, hogy ahol mások eredményeit, szavait vagy gondolatait idéztem, azt a záródolgozatomban minden esetben, beazonosítható módon feltüntettem, a dolgozatban található fotók és ábrák közlésével pedig mások szerzői jogait nem sértem.

Kijelentem, hogy a záródolgozatom CD/DVD adathordozón található elektronikus változata teljes egészében megegyezik a nyomtatott formával.

Hozzájárulok ahhoz, hogy az érvényben lévő jogszabályok és a Számalk-Szalézi Szakgimnázium belső szabályzata alapján az iskola saját könyvtárában megtekinthető (olvasható) legyen a záródolgozatom.

A szakdolgozat titkos/nem titkos.

Budapest, ...év .....hónap..... nap

Tanuló aláírása:.....

Tartalomjegyzék

Ábrajegyzék

1. Bevezetés

A választott témám egy PMF (Personal Finance Management = Személyes Pénzügyi Menedzsment) alkalmazás megvalósítása, vagyis egy pénzügyi tranzakciókat nyilvántartó alkalmazásé, amely segíti a személyes pénzügyek tervezését. A fejlesztés során a közös pénzügyekkel rendelkező családok igényeire fókuszáltam, ezért alkalmazásomnak az FFM – Family Finance Manager nevet adtam.

A PMF alkalmazások jellemzően úgy működnek, hogy a felhasználó regisztrálhatja a bevételt vagy kiadást jelentő pénzügyi tranzakcióit. A tranzakciókat kategóriákba rendszerezheti, és megnézheti, hogy egy adott hónapban mennyit költött a különböző kiadási kategóriákban, és mennyi bevétele volt különböző forrásokból. Az alkalmazás célja, hogy a felhasználó tudatosabban tudja követni a pénzügyeit, azáltal, hogy vizuálisan látja, hogy milyen dolgokra mennyit költ, és ezt akár hónap közben is nyomonkövetheti.

Számos ilyen típusú alkalmazás elérhető, de ezeket nem tartottam jól használhatónak. A legtöbb PFM szoftver problémája, hogy csak ezeket az egyszerű funkciókat valósítja meg, amivel alig ad hozzáadott értéket a felhasználóknak, így azok pár hónap után megunják a használatát. A viszonylag egyszerű pénzügyekkel rendelkező felhasználó, azt látja, hogy minden hónapban ugyanannyi a bevétele, amiből ugyanannyit költ el, nagyjából hasonló összetételben, ezért nem látja értelmét tovább használni a szoftvert. A bonyolultabb pénzügyekkel rendelkezőknek, akiknek havonta nagyobb arányban változnak a bevételeik vagy kiadásaik, túl kevés segítséget nyújt az ingadozások megértéséhez. Ezekből az adatokból sem trendekre nem tudnak következtetni, sem költségvetést nem tudnak összeállítani a következő időszakra, és ezért megy el tőle a kedvük.

Tovább rontja a használhatóságukat, hogy az esetek többségében mobilalkalmazásról van szó, amely a kijelző méretéből adódóan nem tud elegendő információt megjeleníteni az adatok teljeskörű értelmezéséhez.

A problémát úgy kívántam megoldani, hogy az alkalmazás nyújtson segítséget a komplexebb pénzügyek megértéséhez is. Ezt a célt szem előtt tartva a szoftverem két plusz funkciót tartalmaz a legtöbb PMF alkalmazáshoz képest.

Az első a „gyakoriság” elmentése. Ez azt jelenti, hogy a tranzakciók jellemzőjeként tárolni lehet, hogy hasonló tranzakció várhatóan milyen gyakorisággal fordul elő. Így elkülöníthetőek a havi rendszerességgel felmerülő költségek a ritkább, de nagyobb összegű tranzakcióktól, amik az időbeli változékonyságot okozzák.

A második változtatás a műszerfal jellegűen összeállított felhasználói felület. A felhasználókat ez eleinte összezavarhatja, de ha már megszokták, akkor sokkal kényelmesebben tudják elemezni a pénzügyeiket a program segítségével. A műszerfal jellegű felhasználói felület elérése érdekében asztali alkalmazásként készítettem el a programot Java nyelven.

A Java nyelv nagy előnye, hogy platformfüggetlen alkalmazás készíthető vele. Az ingyenes futtatási környezet telepítését követően Windows, Linux és macOS operációs rendszereken is használható a futtatható jar állomány elindításával. A Java kliensoldali technológiái közül a JavaFX-et használtam, amelyben XML és CSS alapú leíró nyelvekkel lehet definiálni a felhasználói felületet. A Scene Builder 2 segítségével gyorsan létre tudtam hozni a GUI-t, amely rögtön egy különálló részt alkotott az alkalmazáson belül. A leíró fájlok manuális szerkesztésével pedig könnyen testre tudtam szabni az automatikusan generált kódokat. A JavaFX másik nagy előnye az elérhető komponensei. A SplitPane, a TableView, a TreeView és a BarChart grafikus komponensek is nagyban hozzájárulnak az alkalmazás felhasználóbarát kialakításához.

Az elkészült alkalmazás a CD-mellékleten található. Az *FFM* mappában található a NetBeans projekt, ami tartalmazza a forráskódot és a program működéséhez szükséges egyéb fájlokat. A *mintaadatokkal* nevű mappában megtalálható a futtatható alkalmazás és egy adatbázis mintaadatokkal feltöltve.

2. Fejlesztői dokumentáció

2.1. Tervezés

2.1.1. Elvárt funkciók

A cél egy olyan asztali alkalmazás készítése volt, amely Windows és Linux operációs rendszeren is fut, központi szerver támogatása nélkül. Az alkalmazással tárolni lehet egy magánszemély vagy közös költségvetéssel rendelkező család szempontjából releváns tranzakciókat, amelyekből megállapítható az elért megtakarítás (vagyonnövekedés). A tranzakciókról ezenkívül különböző részleteket tárol, és ezeket olyan módon tudja megjeleníteni, ami segíti a felhasználót pénzügyi folyamatainak megértésében és a jövőre vonatkozó költségvetésének összeállításában.

A tranzakciók lehetnek bevételek vagy kiadások, amelyekről a következő részleteket tárolja az alkalmazás:

Név: rövid leírás az azonosításhoz

Összeg: egész forintban

Dátum: a tranzakció napja

Kategória: a bevételek és kiadások csoportosítására használt kategória

Gyakoriság: A tranzakció várható ismétlődése. Lehet „nincs ismétlődés”, vagy egy napban, hétben, hónapban vagy évben kifejezett időszak.

Leírás: hosszabb szöveg

A kategóriák a felhasználó által módosíthatók. A tranzakciótípusok (bevétel, kiadás) rendelkeznek egy-egy főkategóriával. Ezekhez a felhasználó fagráf struktúrában tud alkategóriákat hozzáadni tetszőleges mélységben és szélességben. A „Bevételek” és „Kiadások” főkategóriák az adott tranzakciótípushoz tartozó kategória-fagráf gyökerei.

2.1.2. Absztrakció

Az üzleti vállalkozásokkal ellentétben, ha az átlagos magánszemélyek és családok pénzügyi helyzetének változását szeretnénk elemezni, akkor elegendő a pénzmozgással együtt járó pénzügyi tranzakciókat megfigyelni. Ezenkívül feltételezhetjük, hogy a

pénzügyi tranzakció bekövetkezésének időpontja megegyezik a pénzmozgás időpontjával.

Az üzleti életben kialakult kettős könyvelés ugyanakkor a magánembereknél is használható. A fent kifejtett egyszerűsítéseket figyelembe véve elkészíthető a pénzügyi tranzakciókat bemutató számlastruktúra. A kettős könyvelés lényege, hogy mindig két számla értéke változik egyszerre. Ha egy oldalon vannak („tartozik" vagy „követel") akkor ellentétes előjellel változnak, ha különböző oldalon vannak, akkor azonos előjellel. Így biztosítható, hogy a tartozik- és követel számlák egyenlegének összege mindig egyenlő. A „tartozik" és „követel" elnevezéseket nem szó szerint kell érteni, ezek hagyományos könyvviteli elnevezések, amik csupán a két kategória elkülönítését szolgálják.

Tartozik számlák	Követel számlák
Kiadási kategóriák	Bevételi kategóriák
Pénzeszköz-számlák	

1. táblázat Számlastruktúra (Saját készítés)

Az egyszerűsítő feltételek miatt elegendő a pénzeszközök mennyiségének vagy összetételének változásával járó események nyomonkövetése. A pénzügyi tranzakcióknak ebben az esetben három típusát különböztethetjük meg:

1. Bevétel: A pénzeszközök összesített értéke nő, és egy bevételi kategória egyenlege is nő.
2. Kiadás: A pénzeszközök összesített értéke csökken, és egy kiadási kategória egyenlege nő.
3. Átvezetés: A pénzeszközök összesített értéke nem változik. Egy pénzszámla egyenlege csökken, egy másiké pedig nő.

A pénzszámlák szempontjából a bevételek és kiadások oszthatatlan egységet képviselnek. Az átvezetés azonban két részre bontható, mivel egyszerre két pénzszámlát is érint. Az egyik szempontjából pénzkiáramlás, a másik szempontjából pénzbeáramlás történt. A két rész tulajdonságai megegyeznek, kivéve az érintett pénzszámlát, és annak egyenlegére történő hatást.

<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="2">Irány</td></tr>
<tr><td>Bejövő</td><td>Kimenő</td></tr>
<tr><td rowspan="3">Partner</td><td>Külső</td><td>Bevétel</td><td>Kiadás</td></tr>
<tr><td rowspan="2">Belső</td><td colspan="2">Átvezetés</td></tr>
<tr><td>(Bejövő rész)</td><td>(Kimenő rész)</td></tr>
</table>

2. Táblázat: Tranzakciótípusok (Saját készítés)

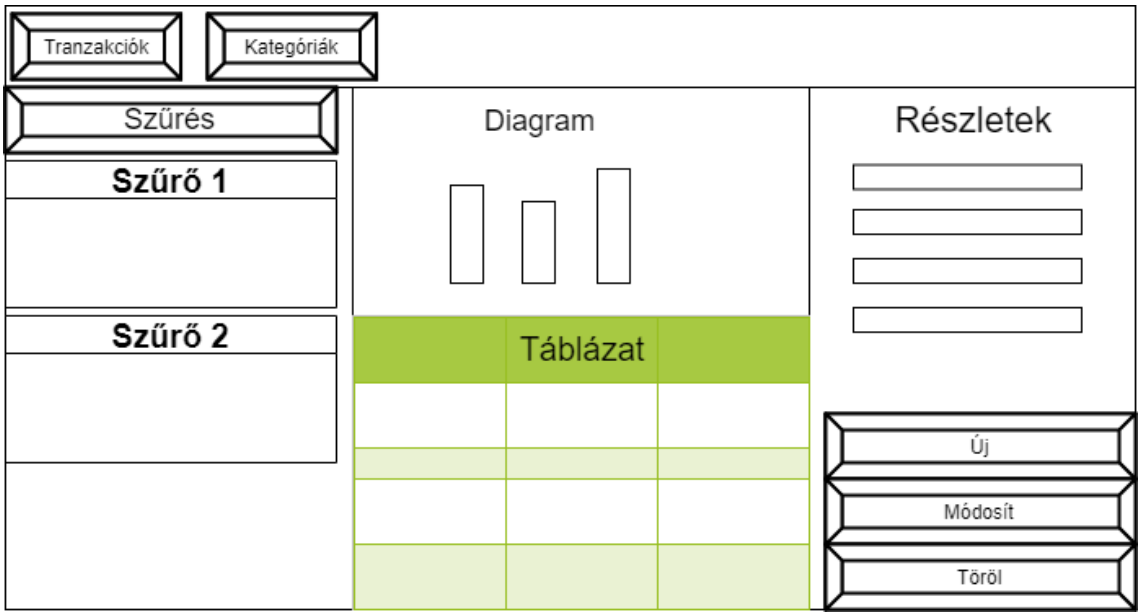
A tranzakciótípusok közül az alkalmazás jelenlegi verziója a bevételek és kiadások kezelését valósítja meg, és nincsenek benne pénzszámlák. Ezen funkciók implementálásának lehetőségéről a Továbbfejlesztési javaslatok fejezetben írok részletesebben.

Mindhárom tranzakciótípus esetén a tranzakció rendelkezik az *Elvárt funkciók* fejezetben részletezett tulajdonságokkal.

2.1.3. Felhasználói felület

Az alkalmazás felhasználói felülete két fő részből áll. A felső sávban navigációgombok találhatóak, amelyek minden nézetben látszanak, és az a feladatuk, hogy a felhasználó váltani tudjon a különböző felületek között. Az alkalmazás jelenlegi verziójában két ilyen felület van, a Tranzakciók és a Kategóriák.

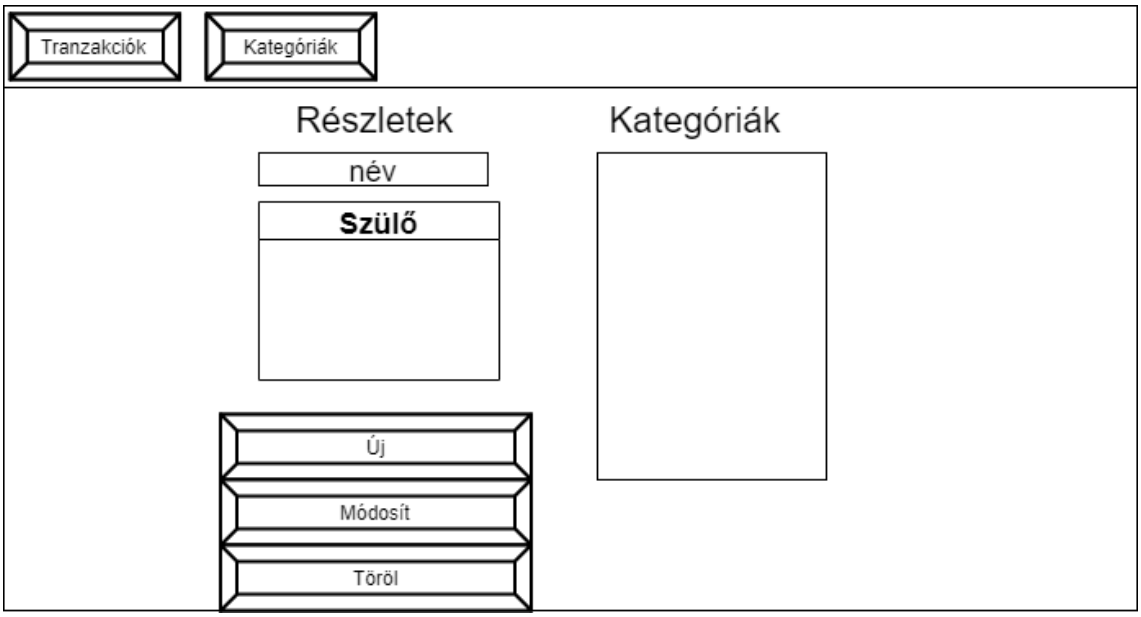
2.1.3.1. Tranzakciók felület



1. ábra A Tranzakciók felület vázlata (Saját készítés)

A Tranzakciók felület az alkalmazás nyitóképernyője, indítás után ez a lap jelenik meg. Ha már fut a program, akkor a fenti navigációsávon a *Tranzakciók* gombra kattintva érhető el. A felület négy részből áll. A négy rész mérete az elválasztóelemek arrébbhúzásával változtatható. A bal oldalon találhatóak a szűréshez használt vezérlők. A *Szűrés* feliratú gomb végrehajtja a szűrést, és frissíti a táblázatot és a diagramot. A gomb alatt lenyitható elemek vannak, amik a különböző szűrési feltételek beállítására szolgálnak. A felület középső része vízszintes irányban van felosztva. Alul található a szűrési feltételeknek megfelelő tranzakciókat megjelenítő táblázat, fölül pedig az ezeket a tranzakciókat havi csoportosításban ábrázoló diagram. A képernyő jobb oldalán a táblázatban éppen kijelölt tranzakció részletei láthatóak az űrlapon. Ezen az űrlapon vihetőek fel az új tranzakciók is a rendszerbe.

2.1.3.2. Kategóriák felület



2. ábra A Kategóriák felület vázlata (Saját készítés)

A kategóriák kezeléséhez külön felület áll rendelkezésre, amit a menüsávon található Kategóriák gombbal lehet elérni. Jobb oldalon találhatóak a már regisztrált kategóriák. A bal oldali részen jelenik meg a kiválasztott kategória részleteit megjelenítő űrlap. Legfölül található a kategória neve. Alatta egy grafikus elemen láthatóak a már regisztrált kategóriák, amelyek közül kiválasztható a kategória szülőkategóriája. Ezalatt pedig a gombok, amelyekkel a kategóriák menthetők, módosíthatók és törölhetők.

2.2. Implementáció

2.2.1. Használt nyelvek

A programutasításokat minden esetben Java nyelven írtam, a Java 8-as verziójának megfelelően. A felhasználói felület az FXML nevű leíró nyelven és a hozzá tartozó JavaFX Cascading Style Sheets (CSS) leíró nyelven készült, a JavaFX 2 szabvány szerint. Az adatbázisutasítások standard ANSI/ISO SQL nyelven történnek a JDBC-n keresztül, az Apache Derby 10.11 dokumentáció alapján.

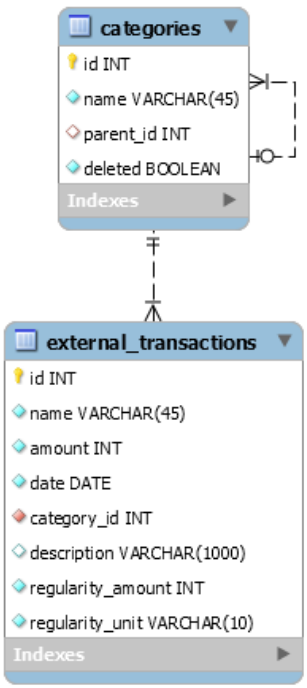
2.2.2. Fejlesztői környezet

A projekt a NetBeans IDE 8.2 fejlesztői környezetben készült. Lefordításához a Java Development Kit 8-ra van szükség. A JDK 8 a Java nyelvű programrészeket Java bájtkódra fordítja, ami futtatható a JRE 8-cal.

Az adatbázis működéséhez a Java DB könyvtárra is szükség van. Ez nem része a JDK 8-nak, ezért a futtatható állomány melletti lib könyvtárban kell elhelyezni dependenciaként. A Java DB könyvtár segítségével az alkalmazás létre tudja hozni a saját beépített adatbázisát, ami értelmezi a JDBC-n keresztül neki küldött SQL utasításokat. A Java DB az Apache Derby 10.11-es verziójú nyílt forráskódú adatbázis-kezelő rendszerét használja. Ennek a következő részei találhatóak a *lib* könyvtárban, az alkalmazás dependenciáiként:

- derby.jar
- derbyclient.jar
- derbynet.jar

2.2.3. Az adatbázis szerkezete



3. ábra Az adatbázis táblái (Saját készítés MySQL Workbench segítségével)

Mezőnév	Típus	Megszorítások	Alapértelmezett érték	Magyarázat
id	INT	NOT NULL, PRIMARY KEY	10-el induló, egyesével növelt szám.	A kategória azonosítószáma. Az első kilenc egész szám a felhasználó által nem módosítható főcsoportoknak van fenntartva.
name	VARCHAR(45)	NOT NULL	Nincs	A kategória neve.
parent_id	INT	Idegen kulcs a categories.id mezőre	NULL	A kategória szülőkategóriájának azonosítószáma.
deleted	BOOLEAN	NOT NULL	false	Jelzi, hogy a felhasználó törölte-e a kategóriát.

3. Táblázat: A categories tábla mezői (Saját készítés)

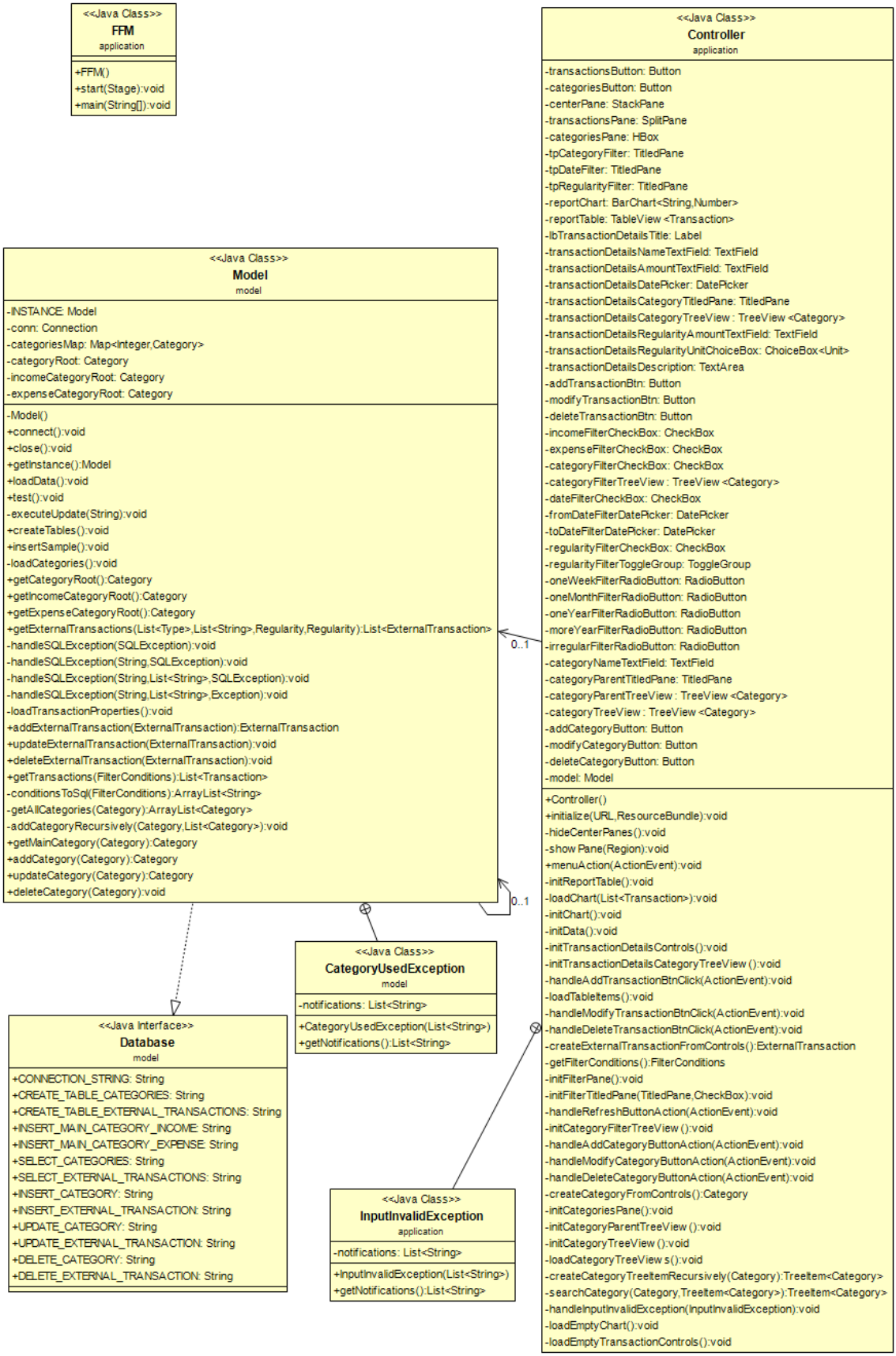
Mezőnév	Típus	Megszorítások	Alapértelmezett érték	Magyarázat
id	INT	NOT NULL, PRIMARY KEY	1-gyel induló, egyesével növelt szám. Manuális értékadás nem lehetséges.	A külső tranzakció azonosítószáma.
name	VARCHAR(45)	NOT NULL	Nincs	A külső tranzakció neve.

Mezőnév	Típus	Megszorítások	Alapértelmezett érték	Magyarázat
amount	INT	NOT NULL	Nincs	A külső tranzakció értéke forintban.
date	DATE	NOT NULL	Nincs	A külső tranzakció bekövetkezésének dátuma.
category_id	INT	NOT NULL, Idegen kulcs a categories.id mezőre	Nincs	A külső tranzakció kategóriájának azonosítószáma.
description	VARCHAR(1000)	Nincs	NULL	Megjegyzések a külső tranzakcióhoz.
regularity_ amount	INT	NOT NULL	Nincs	A külső tranzakció gyakoriságát kifejező időtartam mennyiségét jelölő szám.
regularity_ unit	VARCHAR(10)	Értékkészlet: 'FOREVER', 'DAYS', 'WEEKS', 'MONTHS', 'YEARS'	Nincs	A külső tranzakció gyakoriságát kifejező időtartam mértékegysége.

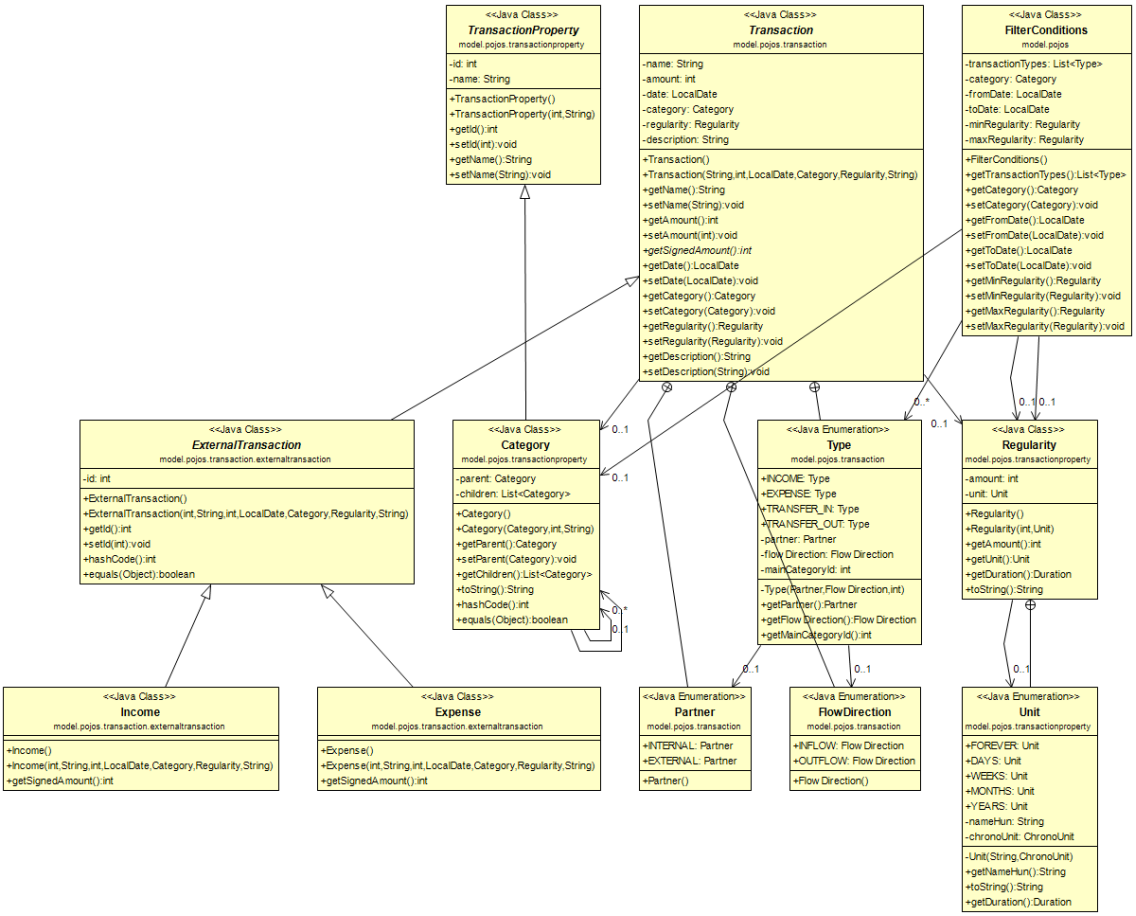
4. Táblázat: Az external_transactions tábla mezői (Saját készítés)

2.2.4. A program statikus felépítése

A program MVC szerkezeti minta szerint készült. A belépési pont az application.FFM osztályban található, amely létrehozza a Viewt és a Controllert, tehát elindítja magát az alkalmazást.



4. ábra Az alkalmazás osztályai (Saját készítés ObjectAid segítségével)



5. ábra Pojo osztályok (Saját készítés ObjectAid segítségével)

2.2.4.1. Model

A Model feladata, hogy adatokat szolgáltasson az alkalmazás aktuális állapotáról, illetve elmentse az állapotban történt változásokat. Jelen program esetében ez azt jelenti, hogy lekérdezhetők tőle a tárolt tranzakciók és a tranzakciókhoz tartozó jellemzők. A Model több részől áll, amelyek a model mappában találhatóak.

A model.Model a Modelt megtestesítő osztály. Az osztály singleton tervezési mintával készült, ami azt jelent, hogy csak egy példány létezhet belőle, amit a getInstance nevű statikus metódusával lehet megkapni.

Az adatbázis kezeléséhez szükséges állandókat, vagyis a connection stringet és az előre megírt lekérdezéseket a model.Database interface tartalmazza. A Database implementálásával a Model közvetlenül tudja használni a definiált állandókat.

A model.pojos csomag tartalmazza az alkalmazás állapotának kifejezéséhez használható osztályokat.

A model.pojos.FilterConditions osztály a felhasználói felületen található szűrőbeállítások megtestesítésére szolgál. Az a funkciója, hogy a szűrési feltételek aktuális beállításai könnyen továbbíthatóak legyenek a controller rétegből a modelnek.

A tranzakciók megtestesítéséhez több leszármazási viszonyban levő osztály is használatban van. A tranzakciók közös tulajdonságait a model.pojos.transaction.Transaction absztrakt osztály tartalmazza. Ezt terjeszti ki a model.pojos.transaction.externaltransaction.ExternalTransaction absztrakt osztály. Ez a tervezésnél bemutatott külső tranzakciót jelenti, ami az átvezetések két résztranzakciójától eltérően önállóan tárolható, ezért id azonosítóval is jelentkezik.

A bevételt és a kiadást megtestesítő Income és Expense osztályok szintén az externaltransaction csomagban találhatóak. Ezek már példányosítható osztályok. Az egyetlen eltérés közöttük, hogy a Transaction absztrakt osztály getSignedAmount nevű absztrakt metódusát, ami előjelesen adja vissza a tranzakció értékét, különbözőképpen implementálják. Mivel minden tranzakció értéke pozitív egész számként kerül tárolásra, ez a metódus tesz különbséget a pénzbeáramlások és –kiáramlások között. Ezzel a metódussal kell összegezni a tranzakciókat, ha a megtakarítást szeretnénk kiszámítani.

A tranzakciók tulajdonságait megjelenítő osztályok a model.pojos.transactionproperty csomagban találhatóak. A TransactionProperty absztrakt osztály az adatbázisban külön táblában tárolt tulajdonságok közös ősosztálya. Az alkalmazás jelenlegi verziójában ennek csak a Category osztály a leszármazottja, ami a tranzakciók kategóriáit jelenti.

A Regularity osztály a gyakoriság tulajdonságot testesíti meg. Ez két részből áll össze, egy mennyiségből és egy mértékegységből. Például ha a gyakoriság két hónap, akkor a mennyiség kettő, a mértékegység a hónap, és a kettő együtt adja meg, hogy mekkora időtávról van szó. A két rész az adatbázisban is külön kerül tárolásra minden tranzakciónál. A gyakoriság mértékegysége egy enumként lett meghatározva, ami a java.time.temporal.ChronoUnit enum néhány elemének felel meg. Ha egy tranzakció nem ismétlődik, azt a Regularity objektum úgy fejezi ki, hogy a mennyisége nulla, a mértékegysége pedig FOREVER. A többi pojotól eltérően a Regularity osztály példányai immutable objektumok, tehát tulajdonságaik már nem változtathatóak a létrehozásuk után.

2.2.4.2. View

A View réteget az application/View.fxml és az application/ViewStyle.css definiálja, a JavaFX 2 szabványnak megfelelően. A View.fxml az XML alapú FXML nyelvet használja a Scene-gráf definiálására. A Scene-gráf egy fagráf adatszerkezet, ami tartalmazza a felhasználói felület elemeit. A fa gyökérelemének egyik attribútuma a Viewhoz tartozó Controller meghatározása. A View és a Controller között két módon jön létre a kapcsolat. Egyrészt a View egyes Scene-gráf elemek esetében változónevet is megad, amivel a Controllerből hivatkozni lehet az adott elemre. Másrészt néhány vezérlőelemnél eseménykezelő metódus is meg lett adva a Viewban. Ez azt jelenti, hogy ha futás közben bekövetkezik az adott vezérlőelem által kiváltott esemény, akkor automatikusan meghívásra kerül a Controller osztály meghivatkozott metódusa.

A ViewStyle.css a JavaFX-ben meghatározott CSS típusú nyelven stílusosztályokat definiál, amiket a View.fxml a felhasználói felület egyes elemeihez rendel, így leegyszerűsítve azok jellemzőinek beállítását.

2.2.4.3. Controller

Az application/Controller.java fájlban található Controller osztály az alkalmazás vezérlő rétege. Feladata, hogy futásidőben vezérelje a View réteget, ami jellemzően a következő feladatokat jelenti:

- A program indulásakor elvégzi a felhasználói felület komponenseinek azon beállításait, amik nem a View-ban lettek definiálva. Például eseménykezelőt rendel a komponensek egyes tulajdonságaihoz, hogy az alkalmazás nyomon tudja követni azok változását. Leggyakrabban a kiválasztható elemmel rendelkező komponensek esetében definiál utasításokat arra az esetre, ha megváltozik a kijelölt elem. A kezdeti beállításokat végző initialize metódus automatikusan meghívódik a program indulásakor. Ez további init kezdetű névvel rendelkező metódusokat is meghív.
- Definiálja a View.fxml-ben hivatkozott eseménykezelő függvényeket. Ez jelenleg a vezérlőgombok által kiváltott ActionEvent típusú események kezelését jelenti. Az eseménykezelők nevei a handle szóval kezdődnek, és tartalmazzák a kezelt esemény típusát is a nevükben.

- Adatokat kérdez le a Model rétegből, azokat átalakítja a View által elvárt formátumra, és megjeleníti a felhasználói felület megfelelő komponensén. Ilyen adatlekérdezések történnek az program indulásakor is, utána pedig a felhasználó által kiváltott események kezelésekor. Ezen metódusok neve a load szóval kezdődik. Az adatok átalakítására egy példa a loadCategoryTreeViews végrehajtása során meghívott createCategoryTreeItemRecursively függvény, ami *A program dinamikus viselkedése* részben részletesebben is bemutatásra kerül.
- Validálja a felhasználói bevitelt, hogy a Model felé már csak a feldolgozáshoz megfelelő adatok legyenek továbbítva. Az esetleges beviteli hibákról a felhasználói felületen tájékoztatja a felhasználót is. A beviteli hibák kezeléséhez az InputInvalidException belső osztályt használja.

2.2.4.4. Belépési pont

Az application/FFM.java fájl tartalmazza az FFM nevű osztályt, ami a javafx.application.Application absztrakt osztály leszármazottja. Ez az osztály tartalmazza a belépési pontot, ami az Application osztály statikus launch metódusával elindítja az alkalmazást. Az indítás közben elkészül egy javafx.stage.Stage típusú objektum, ami az alkalmazás megjelenítéséért felelős ablakot testesíti meg. A launch metódus futása közben meghívódik az Application osztály start nevű absztrakt metódusa is, ami paraméterként az elkészült Stage objektumot kapja meg. Az FFM osztály feladata a start metódus implementálása. Az FFM által implementált start metódus létrehozza a felhasználói felület elemeit tartalmazó adatszerkezetet (Scene-gráfot) a View rétegben definiáltak szerint, elhelyezi azt a Stage-en, beállítja a Stage tulajdonságait, végül pedig megjeleníti a Stage-et, annak show metódusával.

2.2.5. A program dinamikus viselkedése

2.2.5.1. Az alkalmazás elindulása

A program indulásakor a View réteg létrehozza a felhasználói felületet, majd példányosítja a Controller osztályt, és átadja a vezérlést ennek a példánynak. Ekkor automatikusan lefut a Controller initialize metódusa. Ez meghívja az init kezdetű névvel rendelkező metódusokat, amik elvégzik a felhasználói felület működésének beállításait, például beállítják az eseménykezelőket egyes grafikus elemekhez. Az initData metódus

a Model osztály singleton példányát elhelyezi egy példányváltozóban. A Model a connect metódusával létrehozza az adatbáziskapcsolatot. Amennyiben az adatbázis nem létezik, létrehozza a szükséges fájlokat és táblákat.

A loadData metódus a tranzakciók későbbi példányosításakor használt objektumok kezdeti előállításáért felel. Meghívja a loadCategories metódust, ami létrehozza a kategóriák fagráf adatszerkezetét. Első lépésben egy LinkedHashMap kollekcióba betölti az adatbázis rekordjai alapján példányosított kategóriákat, de ezek kapcsolatát először nem tudja beállítani. Egy másik Map objektumba menti el, hogy melyik kategória id-hez melyik parent_id tartozik. Ezen információ alapján a kategóriák kollekcióján végigiterálva már be tudja állítani az egyes kategóriák szülőjét és gyerekeit. Az alapértelmezetten létrehozott főkategóriákat egy fiktív kategória alatt helyezi el, így létrejön a fagráf gyökere. Végül a gyökérelemet és a főkategóriákat is beállítja egy-egy példányváltozó értékének, amelyeket később el lehet kérni a Modeltől. A kategóriahierarchiák grafikus megjelenítését a Controller loadCategoryTreeViews metódusa végzi. A Controller createCategoryTreeItemRecursively függvénye a Modeltől megkapott Category objektum alapján TreeView grafikus elemeken megjeleníthető TreeItem<Category> objektumot készít. A függvény rekurzív módon dolgozik, az átadott kategóriával együtt annak alkategóriából is hasonló módon készít új objektumot, amit el is helyez az új fagráfban.

A tranzakciókat a Model getTransactions függvényével kapja meg a Controller. A felhasználói felületen található táblázat inicializálása során (initReportTable) során hívódik meg a loadTableItems metódus, ami elkéri a beállított szűrési feltételeknek megfelelő tranzakciók listáját a Modeltől, és megjeleníti a táblázatban. A loadTableItems a loadChart metódust is meghívja, ami hónaponként összesíti a megjelenített tranzakciókat, és az értékeket elhelyezi a diagramon.

2.2.5.2. Tranzakció kiválasztása

Ha a felhasználó egy új sort választ ki a táblázatban, akkor a reportTable.getSelectionModel().selectedItemProperty() objektum értesíti a változást figyelő listenereket. Jelen esetben ez egy lambda kifejezés, amit az alkalmazás elindulásakor a Controller initReportTable metódusa állított be. Az esemény bekövetkezése után lefut a lambda kifejezés. Ha a változást az okozta, hogy a kijelölés

megszűnt a táblázatban, akkor a loadEmptyTransactionControls metódussal alaphelyzetbe állítja az űrlapot, és új tranzakciót lehet hozzáadni a *Hozzáad* gombbal.

Ha tranzakció került kiválasztásra a táblázatban, akkor megnézi a tranzakció típusát (bevétel vagy kiadás), hogy ezt az űrlap tetején ki tudja jelezni a felhasználónak. Ezután a kiválasztott tranzakció adataival feltölti az űrlap mezőit. Végül a *Hozzáad* gomb feliratát *Új*-ra cseréli, ami jelzi a felhasználónak, hogy az arra kattintás után tud új tranzakciót regisztrálni.

2.2.5.3. Új tranzakció hozzáadása

Az *Új* vagy *Hozzáad* feliratú gomb lenyomásakor egy ActionEvent típusú esemény jön létre, és meghívódik a Controller osztály handleAddTransactionBtnClick metódusa.

Ha a táblázatban van kiválasztott tranzakció, akkor ez a kiválasztás megszűnik. A *Tranzakció kiválasztása* részben leírtaknak megfelelően ekkor a gomb felirata *Új*-ról *Hozzáad*-ra változik.

Ha nem volt kiválasztott tranzakció, az azt jelenti, hogy a felhasználó egy új tranzakció adatait vitte fel. A Controller createExternalTransactionFromControls függvénye összegyűjti az űrlap adatait, és egy Income vagy Expense típusú objektumot készít belőlük. A folyamat közben ellenőrzi a bevitt adatok megfelelőségét. Ha hibát talál, akkor egy *notifications* nevű figyelmeztetési listához hozzáad egy szöveges hibaüzenetet. Amikor minden ellenőrzés lefutott megvizsgálja, hogy van-e hibaüzenet a listában.

Ha van hibaüzenet, akkor egy InputInvalidException típusú kivételt dob, ami tartalmazza a figyelmeztetési listát. A handleAddTransactionBtnClick elkapja ezt a kivételt, és a handleInputInvalidException meghívásával kezeli. Ez a metódus megjelenít egy párbeszédablakot, amiben kiírja a figyelmeztetési lista üzeneteit a felhasználónak.

Ha nincs hibaüzent, akkor a createExternalTransactionFromControls elkészíti a megfelelő objektumot és visszatér vele. A handleAddTransactionBtnClick ezt továbbadja a Model addExternalTransaction metódusának, ami elmenti a tranzakciót az adatbázisba. Ezután a createExternalTransactionFromControls metódus frissíti a

táblázatban és diagramon megjelenített tranzakciókat. Ha a beállított szűrési feltételek lehetővé teszik, akkor az új tranzakció is megjelenik.

3. Tesztelés

Az alkalmazás megfelelő működését a következő funkcionális tesztekkel próbáltam ki:

3.1. A program új helyre másolása és elindítása

Várt működés: A program elindul, és megjelenik a Tranzakciók képernyő.

Működés: Megegyezik a várttal.

Eredmény: A teszt sikeres.

3.2. Új tranzakció hozzáadása a szükséges mezők kitöltése nélkül

Várt működés: A program visszajelzést ad, hogy melyik mezőket kell még kitölteni.

Működés: A vártnak megfelelő.

Eredmény: A teszt sikeres.

3.3. Új tranzakció hozzáadása az összes mező kitöltése után

Várt működés: A tranzakció megjelenik a felvitt tranzakciók között.

Működés: A vártnak megfelelő.

Eredmény: A teszt sikeres.

3.4. Tranzakció módosítása

Várt működés: A módosított tranzakció megjelenik a felvitt tranzakciók között.

Működés: A vártnak megfelelő.

Eredmény: A teszt sikeres.

3.5. Tranzakció törlése

Várt működés: A tranzakció eltűnik a táblázatból, a diagram frissül, hogy az adatok már ne tartalmazzák a törölt tranzakciót. A beviteli mezők pedig alapállapotba állnak, hogy úgy tranzakciót lehessen regisztrálni.

Működés: A vártnak megfelelő, de törlés után a diagram nem frissül, továbbra is mutatja a törölt tranzakciót.

Eredmény: A teszt sikertelen.

A hiba oka: A diagram nem frissül a táblázat változása esetén. A hiba javításra került.

3.6. Új kategória hozzáadása

Várt működés: Ha a szükséges mezők nincsenek megfelelően kitöltve, akkor a program visszajelzést ad róla, egyébként az új kategória bekerül a kategórialistába.

Működés: A vártnak megfelelő.

Eredmény: A teszt sikeres.

3.7. Kategória törlése

Várt működés: Ha a kategóriának van alkategóriája, vagy tartozik hozzá tranzakció, akkor az alkalmazás értesít róla, és nem törni a kategóriát. Egyébként a kategória eltűnik a listáról, a beviteli mezők pedig alapállapotba állnak, hogy úgy kategóriát lehessen regisztrálni.

Működés: A vártnak megfelelő.

Eredmény: A teszt sikeres.

3.8. Szűrő működése

Várt működés: A szűrők a beállításuknak megfelelően meghatározzák a megjelenített tranzakciókat. Ha egy szűrő aktív, de nincs beállítva, akkor nem befolyásolja a szűrést. A típusszűrő beállítása befolyásolja a diagram megjelenítési módját.

Működés: A vártnak megfelelő.

Eredmény: A teszt sikeres.

4. Felhasználói dokumentáció

4.1. Általános ismertető

4.1.1. A program feladata

A szoftver egy személy vagy közös pénzügyekkel rendelkező család pénzügyi tranzakcióit tárolja, és lehetővé teszi annak elemzését. A tárolt tranzakciók lehetnek bevételek vagy kiadások, ennek megfelelően bevételi és kiadási kategóriákba kell őket sorolni. A tranzakcióknak ezenkívül mindig van összegük és dátumuk (a nap, amikor a tranzakció történt). Ez nem kötelező, de a tranzakciók rendelkezhetnek névvel, gyakorisággal és leírással. A név rövid szöveg, ami segíti azonosítani a tranzakciót. A gyakoriság azt mutatja meg, ha a tranzakció várhatóan bizonyos időközönként ismétlődni fog. A leírás egy hosszabb szöveg is lehet, ami a tranzakció egyéb részleteit mutatja be.

A regisztrált tranzakciókat többféle módon is bemutatja az alkalmazás: egy táblázatban felsorolva, diagramon ábrázolva, és egy űrlapon, minden részletet megjelenítve.

Az program egyik leghasznosabb funkciója a szűrés, ami lehetővé teszi, hogy a felhasználó a tranzakciók jellemzői alapján meghatározott tételeket jelenítsen meg a táblázatban és a diagramon. Így elemezni tudja a tranzakciókat, és különböző következtetéseket tud levonni a pénzügyeit illetően.

4.1.2. A használathoz szükséges környezet

A program asztali alkalmazás, 8-as verziójú Java futtatási környezetben indítható el. Minden számítógépen és operációs rendszeren futtatható, ahol telepítve van a Java 8 futtatási környezet (Java SE Runtime Environment 8) valamelyik verziója.

4.1.3. Korlátozások

A program nem indítható el, ha a számítógépen nincs telepítve a Java SE Runtime Environment 8 valamelyik verziója.

4.1.4. A program telepítése

A program portable módon működik, vagyis nem szükséges telepíteni, elég a futtatható *FFM.jar* fájlt a kívánt könyvtárba másolni. Az *FFM.jar* mellé kell másolni a *lib* mappát

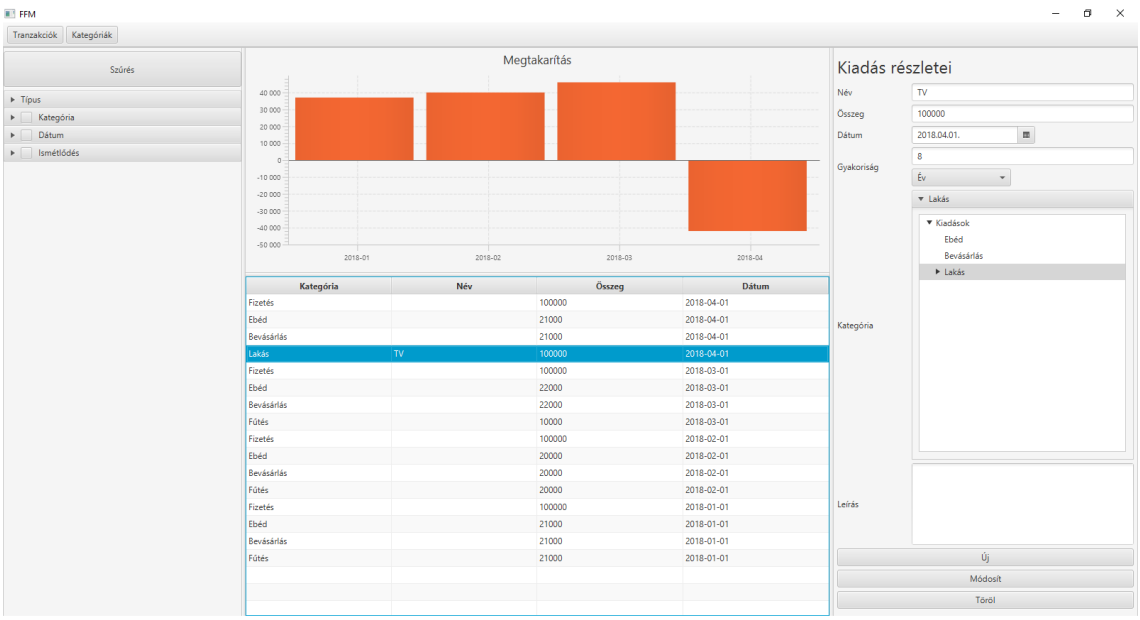
is, ami a futtatáshoz szükséges további fájlokat tartalmazza. A felhasználó által mentett adatokat a *data* mappa tartalmazza. Ha ez még nem létezik, akkor a szoftver a következő elindításkor automatikusan létrehozza. Amennyiben a *data* mappa létezik, az is új helyre másolható az *FFM.jar* futtatható állománnyal együtt, így nem vesznek el a már mentett adatok.

4.2. A program használata

4.2.1. A program futtatása

A programot az *FFM.jar* nevű fájl elindításával futtathatja. Amennyiben már vannak mentett tranzakciói, azokat a program betölti és megjeleníti a Tranzakciók képernyőn a táblázatban és a diagramon.

4.2.2. Új tranzakció regisztrálása



6. ábra A Tranzakciók felület

Ha a táblázatban ki van jelölve egy tranzakció, akkor a tranzakció részleteit mutató jobb oldali terület alján látható egy *Új* feliratú gomb. Erre kattintva megszűnik az aktuális kijelölés, a tranzakció részleteit megjelenítő űrlap alaphelyzetbe áll, alatta pedig a *Hozzáad* feliratú gomb jelenik meg.

Írja be a tranzakció részleteit az űrlap mezőibe:

Név (nem kötelező mező): Egy rövid, maximum 45 karakteres szöveget tartalmazhat, ami a tranzakció későbbi azonosítását segíti.

Összeg (kötelező mező): A tranzakció értéke forintban. Előjel nélküli egész szám.

Dátum (kötelező mező): Az a nap, amikor a tranzakció megtörtént.

Kategória (kötelező mező): A kategóriaválasztó alapesetben össze van csukva, és csak a kiválasztott kategória neve jelenik meg. Ha módosítani szeretné, kattintson a mezőre a megjelenítéshez. A kategória segítségével tudja beállítani, hogy a tranzakció bevétel vagy kiadás. A Bevétel és Kiadás főkategóriákon kívül további alkategóriák is létrehozhatók, ennek lépései is megtalálhatók a felhasználói kézikönyvben. Ha már léteznek alkategóriák, akkor a főkategória bal oldalán található kis nyíllal lehet lenyitni azok listáját. A választott kategóriát kattintással kell kiválasztani, ekkor a kategóriaválasztó terület tetején megjelenik a kiválasztott kategória neve. Erre a területre kattintva a kategóriaválasztó területe újra összecsukható.

Gyakoriság (nem kötelező mezők): A Gyakoriság mezők segítségével lehet beállítani, ha egy tranzakció a jövőben várhatóan ismétlődni fog. Ezt két mező segítségével kell megtenni. A legördülő listában válassza ki a várható ismétlődés mértékegységét (nap, hét, hónap, év), majd a legördülő lista előtti szöveges mezőbe írjon be egy pozitív egész számot, ami a mértékegységgel együtt megadja a várható ismétlődés gyakoriságát. Pl. 2 hét, 1 hónap.

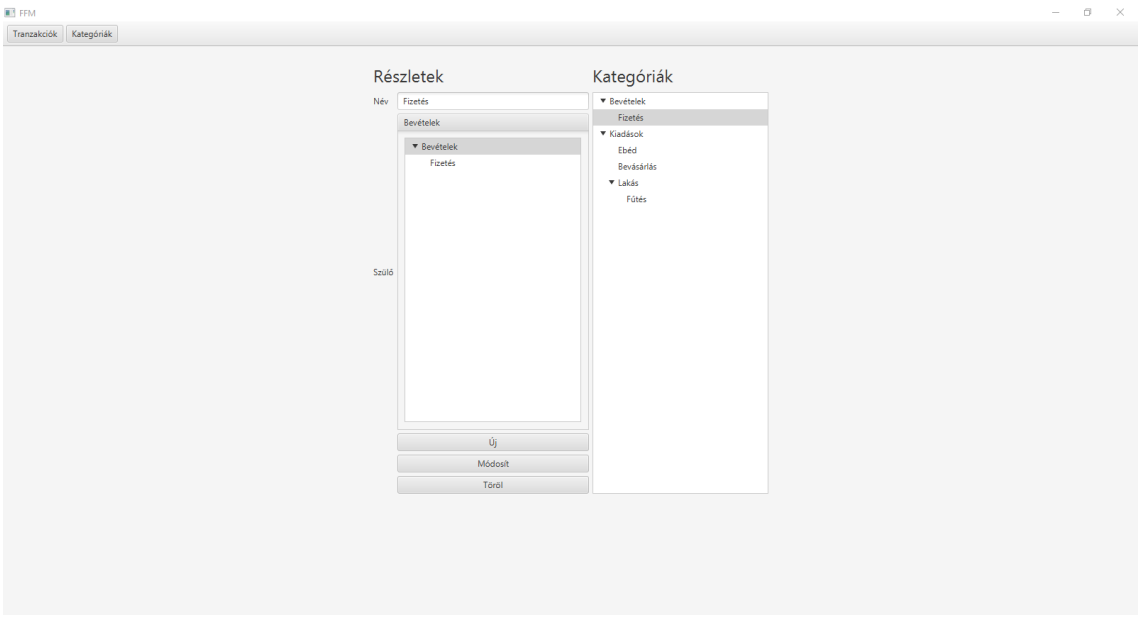
Leírás (nem kötelező mező): Itt egy hosszabb szöveget adhat meg, ami a tranzakció részleteit tartalmazza, pl. milyen részelemekből állt a tranzakció. Maximum 1000 karakter hosszú lehet.

Ezután a Hozzáad gomb megnyomásával tudja menteni a tranzakciót.

4.2.3. Tranzakciók módosítása és törlése

A *Tranzakciók* felületen a táblázatból válassza ki a módosítandó tranzakciót. A jobb oldali űrlapon megjelennek ezek adatai, amelyek átírhatóak a kívánt értékre. Végül a *Módosít* gombbal tudja menteni a változásokat. Ha a tranzakciót törölni szeretné, azt a *Töröl* gombbal tudja megtenni.

4.2.4. Új kategória regisztrálása



7. ábra A Kategóriák felület

Az ablak felső részén található navigációsáv Kategóriák gombjával lépjen a kategóriák kezelőfelületére. Ha már ezen a felületen van, és ki van választva egy kategória a jobb oldali listán, akkor kattintson az *Új* feliratú gombra, hogy alaphelyzetbe álljon a bal oldali űrlap. A kategória neve minimum 1, maximum 45 karakter hosszú lehet. A Szülő felirat melletti listából kell kiválasztani, hogy a kategória melyik másik kategória alkategóriája legyen. Ezután a *Hozzáad* gombbal tudja menteni az új kategóriát.

4.2.5. Kategóriák módosítása és törlése

A *Kategóriák* felületen a jobb oldali listából válassza ki a módosítandó kategóriát. A bal oldali űrlapon megjelennek ezek adatai, amelyek átállíthatók a kívánt értékre. Végül a *Módosít* gombbal tudja menteni a változásokat. Ha a kategóriát törölni szeretné, azt a *Töröl* gombbal tudja megtenni. A kategória csak akkor törölhető, ha nincs alkategóriája, és nincs hozzárendelve tranzakció.

4.2.6. Tranzakciók szűrése és megjelenítése

A tranzakciókat mutató képernyőn lehetőség van a tranzakciók szűrésére. Ez azt jelenti, hogy a középen alul található táblázatban, és a fölötte levő diagramon nem fog minden tranzakció megjelenni, csak a szűrési feltételeknek megfelelőek. A szűréssel meg lehet nézni, hogy bizonyos közös tulajdonsággal rendelkező tranzakcióknak mennyi volt az

összege az egyes hónapokban. Ez segítséget nyújthat a pénzügyek elemzésében és tervezésében is.

A program indulásakor minden tranzakció megjelenik. Ennek az az oka, hogy a szűrők alapbeállítása alapján minden tranzakció ki van választva. A tranzakciótípusoknál a bevételek és a kiadások is ki vannak választva, a többi szűrőfeltétel pedig nincs aktiválva.

A tranzakciótípus szűrője mindig aktív. A többi szűrő manuálisan ki- és bekapcsolható, amit a neve előtt található jelölőnégyzet jelez. Egy szűrő lenyitásakor az automatikusan aktiválódik, azonban a megjelenített tranzakciókra csak akkor lesz hatása, ha ki is van töltve. A *Szűrés* gomb megnyomásával tudja alkalmazni a kiválasztott szűrési feltételeket.

A különböző szűrők működése:

- A típusszűrő közvetlenül befolyásolja a tranzakciók megjelenítésének módját. Ha a bevételek és a kiadások is ki lettek választva, akkor a diagramon a kettő tranzakciótípus különbségeként adódó, havonta megtakarított összeg kerül bemutatásra. Ha csak a bevételek vagy kiadások kerülnek megjelenítésre, akkor azok összesített értéke látszik, és a kiadások is pozitív összegek lesznek.

- A kategóriaszűrő esetében egyszerűen ki kell választani az egyik kategóriát, és a tranzakciók csak abból a kategóriából, illetve annak alkategóriáiból fognak megjelenni. A legfelső kategóriák megegyeznek a tranzakciótípusokkal, ezek kiválasztása ugyanakkor nem befolyásolja a tranzakciók grafikus megjelenítését, úgy ahogy a típusszűrő beállításánál.

- A dátumszűrő esetén egy kezdő és egy lezáró dátumot tud beállítani, amik között megjelenjenek a tranzakciók. Ha be van állítva a kezdődátum, akkor annál korábbi tranzakciók nem lesznek megjelenítve. Ha be van állítva a lezáró dátum, akkor az annál későbbi tranzakciók nem jelennek meg. A diagramon történő megjelenést nem csak a kiválasztott időintervallum határozza meg, hanem az is, hogy melyik hónapokban nem jelenik meg tranzakció. Az idősor elején az a hónap jelenik meg először az oszlopdiagramon, amikor volt tranzakció. Hasonló módon az a hónap jelenik meg utolsó oszlopként, amelyben még volt tranzakció.

- A tranzakciók között ismétlődés (gyakoriság) alapján is lehet szűrni. Itt egy listából lehet kiválasztani különböző ismétlődési intervallumokat. A legtöbb felhasználó valószínűleg szeretné nyomon követni a havi kiadásainak alakulását. A havi rendszerességnél ritkábban jelentkező kiadások azonban torzítják ezeket az adatokat. Az ismétlődésszűrő segítségével szét tudja választani a havonta jelentkező kiadásokat a ritkábban előforduló kiadásoktól. A havi és éves rendszerességű tranzakciók tanulmányozásával össze tudja állítani a következő évre szóló költségvetésének alapját, amit már csak néhány ritkább, de nagy hatású tranzakcióval kell kiegészíteni a viszonylag pontos képhez.

5. Továbbfejlesztési javaslatok

5.1. Pénzszámlák

Az alkalmazás kiegészíthető a felhasználók pénzszámláinak kezelésével. Ez azt jelenti, hogy a felhasználó személy vagy család pénzének különböző megjelenési formái lehetnek, például lehet készpénzben valaki pénztárcájában, különböző bankszámlákon, megtakarítási számlán, értékpapírszámlán. Ahogy az *Absztrakció* fejezetben bemutattam, az alkalmazás tervezésénél már figyelembe vettem ezt a lehetőséget. A megvalósításhoz arra van szükség, hogy a program nyilvántartsa a pénzszámlákat, azok kezdő egyenlegét, és azt hogy az egyes tranzakciók melyik számlát érintik. Ezekből az adatokból az alkalmazás ki tudja számolni az egyes számlák egyenlegét is, amit a felhasználók ellenőrzési céllal össze tudnak vetni a valódi egyenlegekkel. Szükség van egy külön lapra a felhasználói felületen, ahol a felhasználók tudnak új számlákat létrehozni, a létezőeket pedig tudják módosítani vagy törölni. Ezen az oldalon lehetne megnézni a számlák egyenlegét is. Új tranzakció hozzáadásánál egy listából kellene kiválasztani, hogy melyik számlához tartozik a tranzakció. A tranzakciók ezen tulajdonságát ezután meg lehet jeleníteni a táblázatban és a szűrési lehetőségeknél is.

5.2. Átvezetések

A pénzszámlák megfelelő kezeléséhez szükség van egy átvezetés funkcióra is, ami azt jelenti, hogy a felhasználók egyik számlájukról a másikra juttatnak pénzt. Ilyen árvezetés jelenítheti meg az alkalmazáson belül a zsebpénzfizetést a gyerekeknek, a készpénzfelvételt bankszámláról, vagy a befizetést megtakarítási számlára. A tervezésnél kitértem rá, hogy egy átvezetés tranzakció két tranzakciórészből áll. A tranzakciórészek a bevételekkel és kiadásokkal együtt megjeleníthetőek a táblázatban, a részletek megjelenítéséhez azonban új űrlapra van szükség. Az átvezetésekhez használt űrlapon két pénzszámlának kell megjelennie, egy küldő és egy fogadó számlának. Ezenkívül a külső tranzakciókhoz hasonlóak lehetnek a beállítások. Az átvezetéseknek létre kéne hozni egy főkategóriát, ami a Bevételek és Kiadások főkategóriákhoz hasonlóan tovább bővíthető. Az átvezetések adatbázisban történő tárolásához szükség van egy új adattáblára, ami lehetővé teszi a küldő- és a fogadó számla elmentését is. A

szűrésnél egy változtatásra van szükség: a típusok kiválasztásánál meg kell jeleníteni az átvezetések bejövő és kimenő részét is, hogy ezekre külön lehessen szűrni.

5.3. Családtagok

A családtagok megkülönböztetésével lehetne igazi családi alkalmazás a programból. Ennek a fejlesztésnek az a lényege, hogy a pénzszámlákat családtagokhoz kell rendelni, így elkülöníthető, hogy a család vagyona milyen eloszlásban van a család egyes tagjainál. A családtagokat a nevükkel lehet azonosítani, amit egy külön felületen lehetne elmenteni. A pénzszámlák kezelőfelületén lehetne a számlákat a családtagok alá csoportosítani, és itt ki is tudná jelezni az alkalmazás, hogy egy családtaghoz tartozó számláknak mennyi az összesített egyenlege.

5.4. Sorrendváltoztatás

A felhasználók által kiegészíthető tranzakciótulajdonságok (kategóriák, számlák, családtagok) alapesetben a felvitel sorrendjében jelennek meg, ami nem feltétlenül felel meg a felhasználók igényeinek. A sorrendjük módosításához szükség van vezérlőelemekre, például gombokra, amivel a kívánt sorrend beállítható. Az összeállított sorrendet pedig el kell menteni az adatbázisban is. Ez úgy lehetséges, ha egy új mezőben eltároljuk a sorrendre vonatkozó információt.

5.5. Szerveralkalmazássá alakítás

Ez jelentősebb fejlesztést jelentene, de az alkalmazást webalapú szolgáltatássá is át lehet alakítani. Ennek több előnye is lehet. A beépített adatbázis helyett fejlettebb, nagyobb teherbírású és gyorsabb adatbáziskezelőre lehetne áttérni. A felhasználók többféle felületen keresztül is elérhetnék az alkalmazást, például webböngészőben vagy telefonos alkalmazással, de a jelenlegi asztali GUI-val is el lehetne érni az internetes szolgáltatásként működő alkalmazást. Ilyen működés mellett a családtagoknak önálló felhasználóra lenne szüksége, amikhez különböző jogosultsági szintek is beállíthatóak. Például a gyerekek csak a saját tranzakcióikat tudnák regisztrálni, de a teljes kimutatáshoz csak a szülők férnek hozzá.

6. Összefoglalás

A záródolgozatom célja egy viszonylag összetett, MVC szerkezettel megvalósított Java alkalmazás elkészítése volt. A tervezés és megvalósítás közben elsajátítottam a JavaFX technológia alapjait, és először használtam az Apache Derby adatbázismotorra épülő JavaDB-t. A Java programozási nyelvet és a NetBeans fejlesztői környezetet is alaposabban meg tudtam ismerni. Néhány eredetileg tervezett funkciót végül nem tudtam megvalósítani időhiány miatt, ezekről részletesebben írtam *a Továbbfejlesztési javaslatok* részben. Ennek ellenére a családi pénzügyi nyilvántartó alkalmazás az elkészült funkciókkal is működőképes és használható, ezért úgy érzem a záródolgozatommal elértem a kitűzött célt.

7. Irodalomjegyzék

1. Angster Erzsébet: Objektumorientált tervezés és programozás, Java. 4KÖR Bt., Martonvásár, 2003

2. Kaczur Sándor: Programozás Java nyelven. Budapest, 2016

Derby 10.11 Reference Manual: http://db.apache.org/derby/docs/10.11/ref/index.html, Letöltve: 2018. 04. 08.

JavaFX 8 API Specification: https://docs.oracle.com/javase/8/javafx/api/, Letöltve: 2018. 04. 08.

JavaFX 8 oktatóanyagok: https://docs.oracle.com/javase/8/javase-clienttechnologies.htm, Letöltve: 2018. 04. 08.

Java™ Platform, Standard Edition 8 API Specification: https://docs.oracle.com/javase/8/docs/api/, Letöltve: 2018. 04. 08.