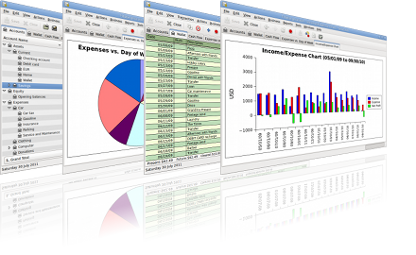
# 2. A pénzügyi szoftverek

Alkalmazást írni szinte bármiről lehet. Ha veszünk egy problémát, és találunk rá egy ötletet, egy megoldást, akkor máris elkezdhetünk vele dolgozni, és ezzel akár az emberek életét is jobbá tudjuk tenni. Ilyen alkalmazások a pénzügyi szoftverek is. Ezek az alkalmazások a pénzügyi gondjainkat hivatottak csökkenteni, mivel sokan bele szoktak esni abba a csapdába, hogy valamiért nehezükre esik a pénzzel bánni, így egy útmutatókat kaphatnak a fogyasztók a pénzügyi világban. Ebben a fejezetben a pénzügyi alkalmazásokat, és annak lehetőségeit vizsgálom meg.

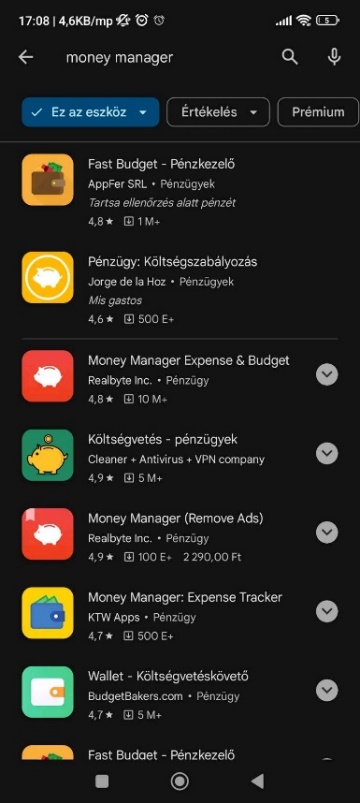
## 2.1. Személyes pénzügyi asszisztens szoftverek

Manapság már szinte mindenre képesek vagyunk egy alkalmazást találni bármilyen tekintetben, legyen az otthoni alkalmazás, játék, sport, vagy bármilyen egyéb tevékenység. Ennek következtében a pénzügyi paletta sem üres, sokféle alkalmazás tárul elénk, mikor egy alkalmazást szeretnénk találni a pénzügyi tevékenységünk monitorozásához. Ezen alkalmazásoknak az a célja, hogy a felhasználó számára egy átláthatóbb és megfoghatóbb költési képet tudjon nyújtani, amivel át tudjuk gondolni, hogy egy adott időszakban mi az, amire költünk, és szemünk elé kerülhetnek olyan dolgok is, amire nem feltétlenül lenne szükséges költeni. Néhány program még esetlegesen képes többet is nyújtani, lehet akár egy megtakarítási portfóliót is kezelni egy ilyen alkalmazással. Nagyon fontos azonban megkülönböztetni a különböző platformokat, amik a rendelkezésünkre állnak. Vegyük elsőként a számítógépre készült alkalmazásokat. Ilyenek lehetnek például a GnuCash, a HomeBank [1] alkalmazások. „A könnyen használható, nagy tudású és rugalmas GnuCash lehetővé teszi a bankszámlák, részvények, bevételek és kiadások nyomon követését. Ugyanolyan gyors és egyszerű, mintha füzetben vezetnénk a nyilvántartásunkat, mégis professzionális számviteli elveken alapul, biztosítva ezzel a mérleg és a kimutatások pontosságát.” [2] Előnye lehet, hogy mivel ezek a programok asztali gépekre készülnek, így nagyobb felület áll a rendelkezésünkre, mint egy telefonnál, ezért a vizualizáció sokkal erősebb eszközzé válhat. Ezekben a programokban tudunk számlákat felállítani, költést és jövedelmet rögzíteni, esetlegesen havidíjas szolgáltatásokat is tudunk benne rögzíteni, így jobban láthatjuk, hogy egy hónapban milyen alapvető dolgaink vannak, amire költünk, és ezáltal jobban tudjuk mérlegelni, hogy mire is van szükségünk. A számítógépes platform hátránya véleményem szerint az, hogy minden alkalommal, mikor valamit rögzíteni vagy ellenőrizni szeretnénk, akkor bizony a számítógépet kell elővennünk, ami nem feltétlen jelent egy hordozható eszközt (például laptopot), így nagyobbá válik annak a valószínűsége, hogy egy idő után a felhasználó megfeledkezik a program használatáról, és így elesik a program potenciáljától is. Azonban, ha rendeltetésszerűen használjuk az alkalmazást, akkor egy teljesen részletes pénzügyi vizualizálás tárul elénk, de ezen alkalmazások más egyébre nem hivatottak működni, nincs bennük megtakarítási cél, sem bevásárlási összegző sem. További tapasztalat, hogy az ilyen programokat nem nagyon ismerik, csak akkor van lehetőségünk ilyenekkel találkozni, ha kifejezetten keressük az efféle alkalmazásokat. Személy szerint én nem is használtam eddig számítógépes költségkezelő alkalmazást, ennek okát a többi mód összehasonlításában kell keresni.



2.1 ábra: A GnuCash sokszínű felülete.

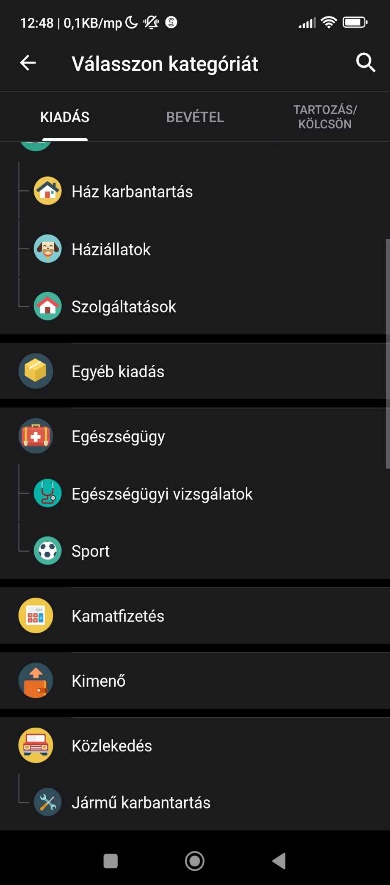
Térjünk rá a mobilos eszközökre. A mai világban már sokan elképzelni sem tudják a világot telefon nélkül, hatalmas teret hódított magának ez az eszköz, mivel nagyon egyszerű a használata, és sok esetben megkönnyítheti az életünket. Olyan fejletté váltak ezek az eszközök, hogy szinte mindenünket képesek leszünk irányítani vele (például, ha belegondolok abba, hogy az otthoni klímámat a kollégiumból is be tudom indítani). A telefonos rendszereken temérdek alkalmazás áll a rendelkezésünkre a rendszer „áruházában”, amiből telepíteni tudjuk a nekünk tetsző alkalmazást, így ezt az alkalmazást akár órákig is képesek lehetnénk görgetni. Így tehát nem meglepő, hogy több opció közül választhatunk, ha a pénzügyi asszisztensre rákeresünk:



2.1 ábra: Általános pénzügykezelő alkalmazások.

Ezek a programok általában annyit tudnak nyújtani a felhasználó számára, hogy egy egyenleg betáplálása után rögzítheti a költekezéseit, majd az alkalmazás ellátja különböző adatokkal és ábrákkal, hogy hogy áll a havi egyenlege stb. Ha valaki nem látja át, hogy egy hónapban mire költ, akkor egy ilyen alkalmazás nagyon hasznos lehet számára. Egy másik előnye a mobilos alkalmazásoknak, az az, hogy mivel a mobil eszközünk a nap nagy részében nálunk van, így sokkal egyszerűbb a költéseket rögzíteni, mint egy asztali gépes verziónál, illetve egy ilyen típusú problémára a legracionálisabb megoldás szerintem a telefonos megoldás.

Példaként vegyük a Money Lover nevezetű alkalmazást. Ebben az alkalmazásban rögzíthetjük a költéseinket kategória szerint, illetve a kategóriáknál megszabhatjuk, hogy az adott hónapban mennyit szeretnénk az adott kategóriára ráfordítani. Hátrányként tapasztalom, hogy mivel sok kategória került implementálásra, így a rögzítési folyamat nehezebbé válik, nehézkessé, hosszassá válik egy-egy költség rögzítése.



2.2 ábra: A Money Lover alkalmazásban rengeteg kategória közül választhatunk.

Egy másik alkalmazás, ami pénzügyi nyilvántartást mutat, az az OTP Bank alkalmazása. Aki ennél a banknál rendelkezik számlával, az gond nélkül követheti a számláján történő eseményeket. A program végez költségkimutatást csoportosítva, megmutatja, hogy mennyit költöttünk a hónapban, illetve visszanézhetjük a korábbi számlatörténetet is. 2023. augusztusában került bele a persely nevezetű újítás is, amivel már meghatározhatunk egy célt amire gyűjteni szeretnénk, és az erre fordított összeget bele tudjuk helyezni ebbe a perselybe. Hátránya viszont, hogy ebben a programban nincs beleépítve a tudatossági rész, hiszen csak összegez, tehát itt nem olyan számottevően látszódik, hogy mire mennyi megy el egy hónapban.



2.3 ábra: Az OTP Bank persely funkciója egy ígéretes megtakarítási eszköz.

Elmondható tehát, hogy az ilyen típusú programok általában csak azt hivatottak elérni, hogy egy képet kapjunk a költéseinkről, más opciót azonban nem, vagy csak ritkán találhatunk meg. A saját ötletem egy olyan alkalmazás implementálása, amely a költség megfigyelésen kívül a megtakarítási célokra is fókuszál, illetve, ha vásárlunk, egy kényelmi funkciót is igénybe tudjunk venni, amivel a vásárlás során sem leszünk vakon, mivel a kosár értékét meg tudjuk vele figyelni.

# 3. A MoneyWatch felhasználói útmutatója

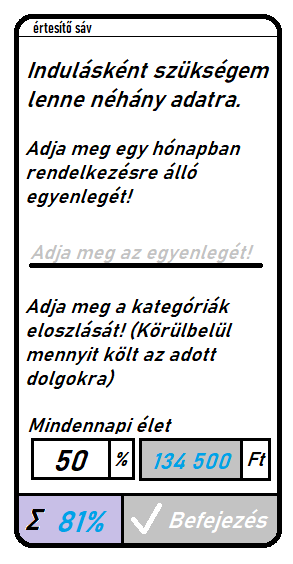
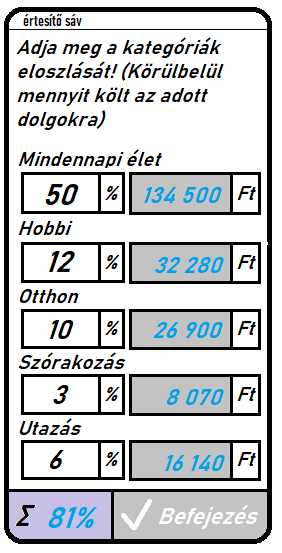
Az alábbi fejezetben az alkalmazás számára készült felhasználói útmutató kerül bemutatásra, ami bemutatja a program funkcionalitásait, a megtervezett programot, és végig vezeti a felhasználót a program működésében.

## 3.1. Az alkalmazás bemutatása, célja

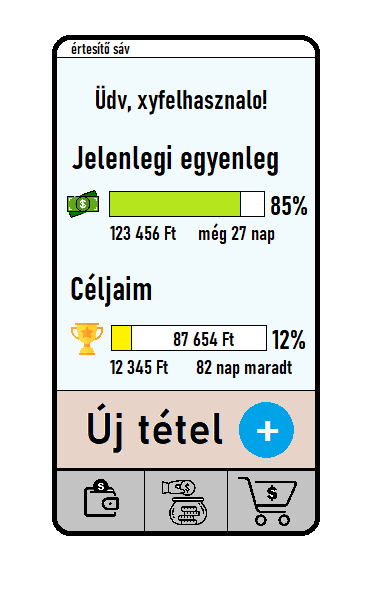
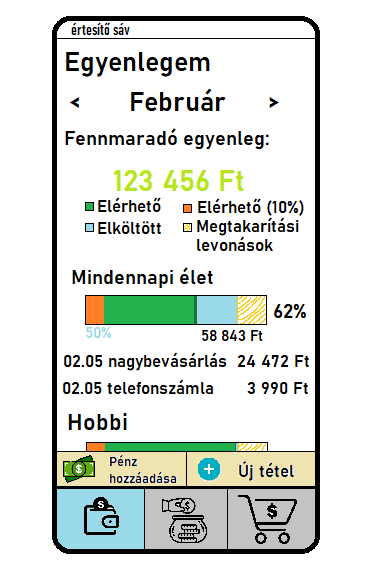
Az alkalmazás célja a felhasználó számára a pénzügyi monitorozás, ezen alkalmazás segítségével a felhasználó jobban átláthatóvá teszi a kiadásokat, illetve megkönnyíti/segíti a megtakarításokat.

Az alkalmazás nagy segítséget nyújthat továbbá a bevásárlás során a költségek figyelésére.

## 3.2. Letöltés utáni teendők

Az alkalmazás kér egy *felhasználónevet*, hogy a későbbiekben meg tudja majd őt szólítani.

A név bekérés után megkérdezi az alkalmazás, hogy egy hónapban mekkora összeg áll a rendelkezésére (az alkalmazás minden megadott adatot bizalmasan kezel, az adatokat nem adja ki más személynek), illetve megkérdezi azt is, hogy körülbelül mekkora az az összeg, amit egy hónapban meg szeretne takarítani a felhasználó.

Ezután az alkalmazáson belül kezelt kategóriák arányairól érdeklődik az alkalmazás. Hogy milyen kategóriák kerüljenek a programba, az egy nagyon fontos része az egész alkalmazásnak. Ez szerintem egy lényeges része a programnak, mivel szerintem sok alkalmazás itt rontja el az egyszerűségi oldalát az egész alkalmazásnak, holott egy ilyen programnál nagyon fontos az egyszerűség főleg azért, mert így nem fogjuk majd elfelejteni betáplálni az adatokat, kevesebb kedvünk akad majd megcsinálni a felviteleket stb. Éppen ezért a kategóriákat az OTP Bank alkalmazása és a már elérhető pénzügyi asszisztensek alapján próbáltam meg kialakítani. az alkalmazásban jelenleg megadott kategóriák a következők:

* *Mindennapi élet* (ide sorolhatóak a következő tárgyak: Étkezés, ruházkodás, havidíjak, szolgáltatások)
* *Hobbi* (A felhasználó számára érdekes tevekénységének költségei, pl: kerékpározás)
* *Otthon* (Ritkán előforduló nagyobb tételű vásárlások pl: konyhai eszközök, rezsiköltség)
* *Szórakozás* (Házon kívüli étkezés, mozi, szórakozóhely, házibuli és ezekhez hasonló helyek)
* *Utazás* (Tömegközlekedés, autó költségek, üzemanyag, szállás költségei)
* *Megtakarítás (*Minden olyan tranzakció, ami a nagyobb céljaink elérését szolgálja.)

Az alkalmazás letöltése és megnyitása után előjön a főképernyő, itt láthatjuk az aktuális egyenleget, és néhány egyéb dolgot, amit az alkalmazás kiszámol magától, például azt, hogy egy napra mennyi kiadás jut, a különböző kategóriákat és az arra költött pénzmennyiséget.

Gyorsgombként rendelkezésre áll a *tétel rögzítés*, illetve a főbb menüpontok (ezek az összes menüponton elérhetőek, váltogathatóak).

## 3.3. A főegyenleg kezelése

Ebben a menüpontban jelennek meg a naplózott tételek, illetve az egyenleg állapota is itt látható részletesen lebontva.

Ebben a menüpontban is lehetőségünk van kiadást rögzíteni ugyanolyan módon, mint a főmenünél.

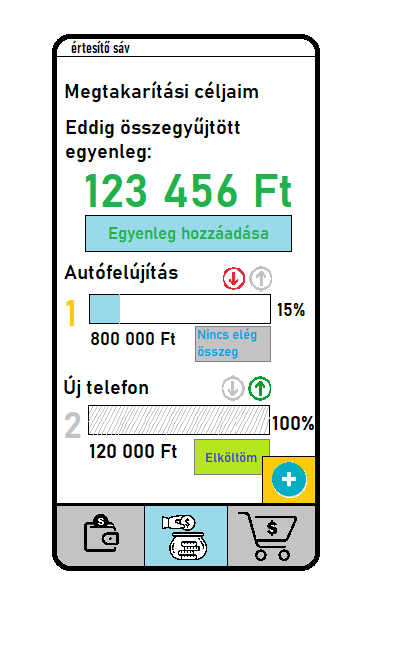
Az alkalmazás a kategóriák szerint naplózza a kiadásokat, így vissza lehet benne keresni, hogy mikor mire költöttünk épp, illetve a havi egyenlegi lebontást is szemlélteti a felhasználónak.

Ha új bevételi forrásunk nyílik, akkor ennél a pontnál szükséges hozzáadni a mennyiséget, ami hasonló módon működik, mint a tételhozzáadás, de pozitív irányba mozdítja az egyenleget.

## 3.4. Egy adott költség rögzítése

Nyomjunk rá az *Új tétel* pontra!

Ekkor megjelenik a rögzítés opciói az alkalmazásban, itt a következő információkat kell megadnunk:

* A tétel összegét
* A tétel kategóriáját
* Egy címkét is, hogy emlékezzünk, mire költöttünk.

A dolgok betáplálása után a rögzítés pontot megnyomva az alkalmazás automatikusan hozzáadja a kívánt kategóriához a megadott tételt, levonja az összeget az egyenlegből.

## 3.5. Megtakarítási célok beállítása

Az alkalmazás egyik funkciója, hogy beállíthatunk rajta olyan dolgokat, amikre össze szeretnénk gyűjteni az összeget, és az alkalmazás végig vezet az úton ahhoz, hogy el tudjuk érni a kívánt összeget.

A *megtakarítások* gombra nyomva megjelennek az eddig megadott célok, és az előrehaladásuk.

Az alkalmazásban bármennyi célt meg lehet adni, de azért, hogy ne a végtelenségig gyűljön úgy a pénzünk, hogy semmink nem gyűlik ki, ezért a program egyenleg alapon számítja a megtakarításokat. Ennek az a jelentése, hogy a céloknál meg kell határozni egy prioritást, majd a program mindig a legnagyobb prioritású dolgokhoz adja a már megtakarított pénzmennyiséget.

Ha kigyűjtöttük az egyik célt, akkor az *elköltöm* gombbal kiválthatjuk a célt, így az eltűnik, és a cél összege levonódik a megtakarítási egyenlegből.

## 3.6. Új megtakarítási cél létrehozása

Az alkalmazás kétféle célt különböztet meg:

* *Határidős cél* (ilyen lehet pl: nyaralás, utazás, gyűjtés egy eseményre)
* *Nemhatáridős cél* (bármilyen cél, amihez nem tudunk konkrét határidőt rendelni, pl: lakásmegtakarítás)

Fontos információ: A két típus különböző módon viselkedik, ezért körültekintően válasszunk!

## 3.7. Határidős megtakarítási cél hozzáadása

Nyomjunk rá az *Új tétel* pontra!

Ekkor megjelenik a célrögzítés menüpont az alkalmazásban.

Ezek után a következő adatokat szükséges megadni:

* A céltárgy típusa ebben az esetben határidős.
* Céltárgy neve
* Cél összege
* A cél határideje (hónapokban vagy dátum szerint)

Elfogadás után az alkalmazás hozzáadja a célokhoz a megadott adatokat, illetve az elérhető havi egyenleget az alkalmazás csökkenti az összeggel (hónapokra leosztva, tegyük fel, hogy 250 ezer forintunk van, de ha egy olyan célt adunk hozzá, aminek a teljesítéséhez havi 30 ezer forintot kell megtakarítanunk, a program csak 220 ezer forintnak megfelelő egyenleggel fog számolni, mivel így tudjuk csak időben elérni a célunkat).

A cél esetleges törlése esetén a célra eddig megtakarított (levont) egyenleget a megtakarítás számlára helyezi vissza.

Fontos információ: Ahhoz, hogy a gyűjtés ne váljon teljesíthetetlenné, ha a rendelkezésre álló egyenleg 20%-a célokra alakul, akkor az alkalmazás figyelmezteti a felhasználót, hogy nagy arányú megtakarításokat szeretne, aminek a gyűjtése nehezen teljesíthetővé válhat.

## 3.8. Határidő nélküli megtakarítási cél létrehozása

Nyomjunk rá az *Új tétel* pontra!

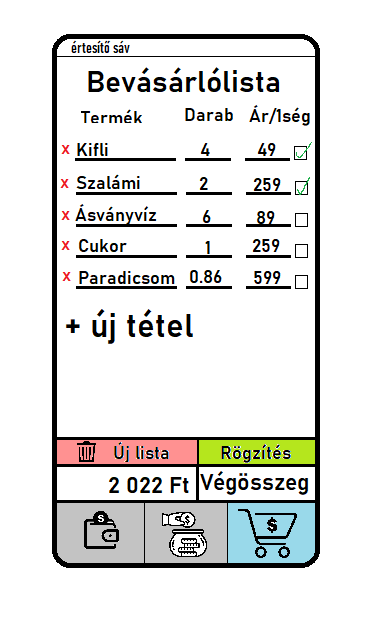
Ekkor megjelenik a célrögzítés opció az alkalmazásban, itt válasszuk ki a *határidő nélküli* célt!

Ezek után a következő adatokat szükséges megadni:

* A célnál ebben az esetben nem szabunk meg határidőt.
* Céltárgy neve
* Cél összege

Az ilyen típusú tételeket a megtakarított egyenleg segítségével tudjuk összegyűjteni. Ehhez az egyenleghez a *Megtakarítások* fülnél az egyenleg alatti pontnál tudunk összeget adni a rendelkezésre álló főegyenlegből (illetve pénzhez is tudunk jutni a megtakarítási egyenlegből, bár az alkalmazás célja nem ez lenne.)

## 3.9. A Bevásárlás asszisztens működése

****Az egyik legproblémásabb dolog amikor bevásárolunk, hogy még ha listát is készítünk, fizetéskor nem tudjuk mennyit költünk majd. Ennek kiküszöbölésére áll rendelkezésünkre a *bevásárlás asszisztens*, amiben a bevásárlólista mellett az eddig költött összeget is tudjuk figyelni.

Működése:

* Írjuk össze a bevásárolni kívánt dolgokat! Új tárgyat az *Új tétel* gomb megnyomásával tudunk létrehozni.
* Adjuk meg a megvásárolni kívánt tárgy nevét! Mellé pedig, hogy mennyit szeretnénk az adott dologból vásárolni.
* Ha meggondoltuk magunkat egy bizonyos tétellel kapcsolatban, akkor a kuka gomb megnyomásával el tudjuk azt távolítani a listából.
* A bevásárlólistát bármikor össze lehet írni, ha újra ide lépünk, akkor is a legutóbbi listát fogja mutatni.

A bevásárlás közben megadhatjuk az alkalmazásban, hogy az adott terméknek mennyi az egységára! Ezekkel az adatokkal az alkalmazás automatikusan kiszámolja a termék árát, illetve a képernyő alján folyamatosan írja az eddigi összes kiadást.

A bevásárlás véglegesítése gombbal hozzá tudjuk adni az egyenlegfigyelőhöz is, így nem kell külön tételt hozzáadni. Ekkor megjelenik előttünk a tétel rögzítő felület, ekkor már csak a kategóriát és a tárgy címét kell ellenőrizni (Ennek megtétele után törlődik a bevásárlólista.)

# 4. A fejlesztőkörnyezet kiválasztása

A fejlesztés során nagyon fontos elem, hogy kiválasszuk, milyen platformra és milyen nyelven szeretnénk az alkalmazásunkat fejleszteni. Ebben a fejezetben a választott programozási nyelvet és a fejlesztőkörnyezetet mutatom be.

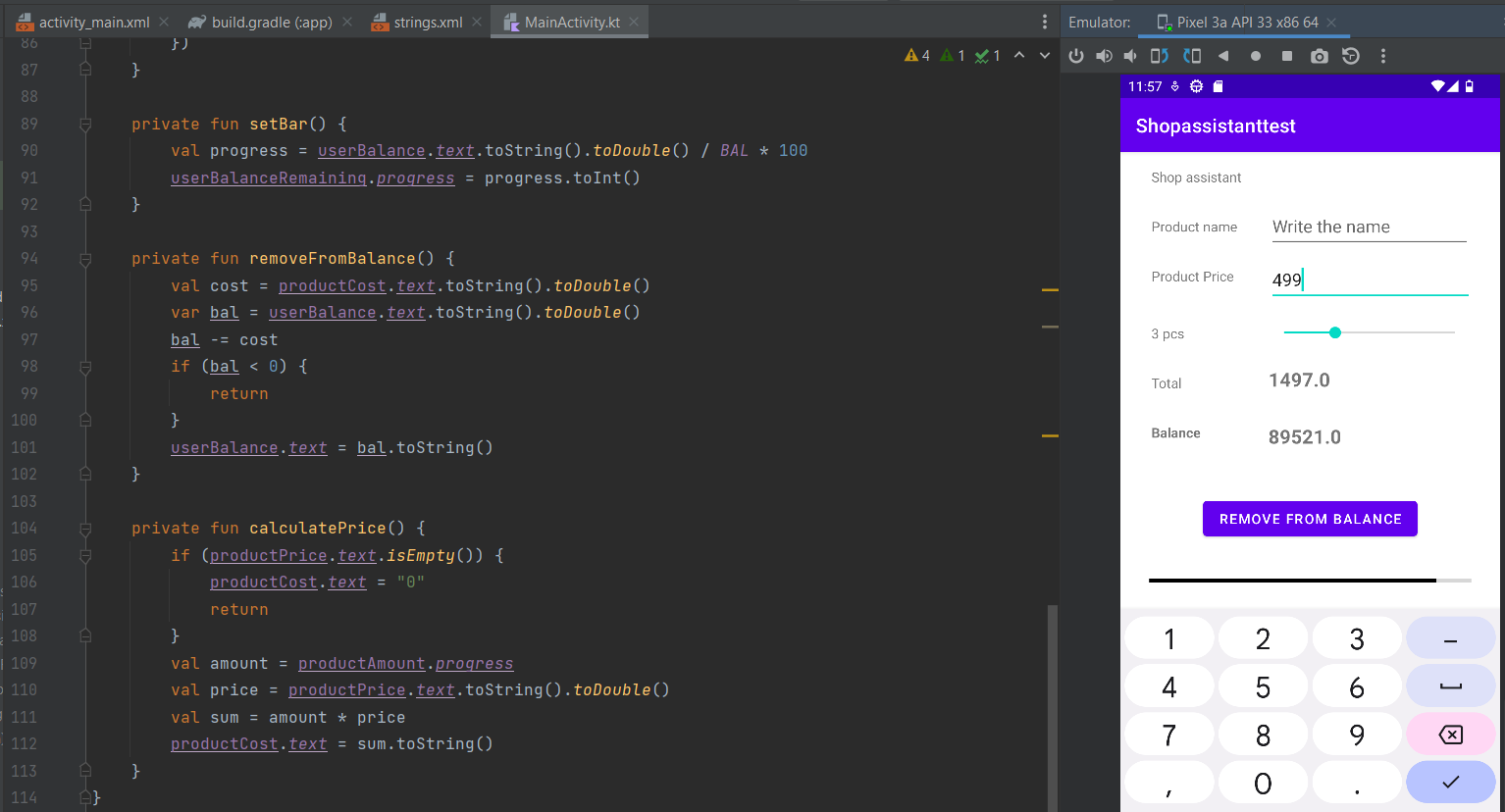
## 4.1. A kiválasztott programozási nyelv

Mivel egy telefonos alkalmazást szeretnék programozni, így egy telefonnal kompatibilis programozási nyelvet szeretnék választani. Itt szóba jöhet a Flutter és a Kotlin, de mivel az eddigi összes tapasztalatom az Android rendszerrel kapcsolatban létezik, így a Kotlin nyelvre esett a választásom, ami az Android hivatalos programozási nyelve. A Kotlin egy modern programozási nyelv, amelyet a professzionális Android-fejlesztők több mint 60%-a használ, és amely segít a termelékenység, a fejlesztői elégedettség és a kódbiztonság növelésében. [5] A Kotlin programozási nyelvet eredetileg a Java programozási nyelv továbbfejlesztésére tervezték, és gyakran használják a Javával együtt. Annak ellenére, hogy a Kotlin az Android preferált fejlesztői nyelve, a Java-val való átjárhatósága miatt számos alkalmazástípushoz használják. A nyelvnek számos előnye létezik, a fontosabbak a következők:

* Átjárhatóság. A Kotlin átjárható a Javával, mivel ugyanazt a bájtkódot fordítja le. A Kotlin lefordítható JavaScriptbe Java kódra, ami lehetővé teszi a programozók számára, hogy just-in-time fordítást végezzenek annak érdekében, hogy a másik programba ágyazott kód zökkenőmentesen fusson. A Java-val közös eszközrendszereket is használ. Ezek a funkciók megkönnyítik a Java-alkalmazások Kotlinra való áttelepítését.
* Biztonság. A Kotlint úgy tervezték, hogy segítsen elkerülni a gyakori kódolási hibákat, amelyek tönkretehetik a kódot, vagy sebezhetőséget hagyhatnak benne. A nyelv rendelkezik nullbiztonsággal és a nullmutatós kivételi hibák kiküszöbölésével.
* Egyértelműség. A Kotlin kiküszöböli a népszerű nyelvek, például a Java alapszintaktikájának néhány redundanciáját. A Kotlin időt takarít meg a fejlesztők számára, mivel tömörebb kódot biztosít. A fejlesztők kevesebb kóddal írhatnak programokat, ami növeli a termelékenységüket.
* Eszköztámogatás. A Kotlin rendelkezik az Android eszköztámogatásával az Android fejlesztésre optimalizált eszközökkel, beleértve az Android Studio-t, az Android KTX-et és az Android SDK-t.
* Közösségi támogatás. Bár a Kotlin viszonylag új nyelv a Javához képest, van egy fejlesztőkből álló közössége, akik a nyelv fejlesztésén és a dokumentáció biztosításán dolgoznak.

## 4.2. A kiválasztott fejlesztőkörnyezet

A kiválasztott programozási nyelv után a fejlesztőkörnyezet kiválasztása következett. A választásom a Kotlin hivatalos feljesztőkörnyezetére, az Android Studiora esett. Az IntelliJ IDEA nagy teljesítményű kódszerkesztőjére és fejlesztői eszközeire épülő Android Studio még több olyan funkciót kínál, amelyek növelik a termelékenységet az Android-alkalmazások készítése során, például: gyors és funkciógazdag emulátor, egységes környezet, ahol az összes Android eszközre fejleszthetünk, kiterjedt tesztelési eszközök és keretrendszerek [7]. Ez a fejlesztőeszköz szabadon használható, és a saját oldaláról tölthetjük le a fejlesztőkörnyezetet. A fejlesztőkörnyezet előnye, hogy az oldal design-ját és a benne zajló műveletek implementálását is képesek vagyunk egy programon belül elvégezni. Maga a programot ilyen formában használom, de még egy előnye, hogy rengeteg módon testre tudjuk szabni a program felületét, ahogy tetszik mindenki számára.



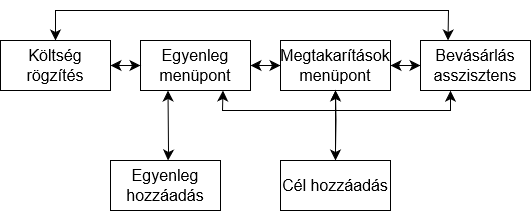
4.1 ábra: Az Android Studio felülete.

# 5. Az alkalmazás implementálása

Az alkalmazás fő tervezete elkészült a felhasználói leírás alapján. Ezek után az implementálás következik, amit tehát Android Studioban vittem véghez Kotlin nyelvet használva.

## 5.1. Az alkalmazás lapátmenete

Elsősorban meg kell határoznunk, hogy melyik lapból hogyan tudunk melyik lapra jutni. Induljunk ki a főegyenlegből! Mivel nagyon gyakori a navigációs csík használata a telefonos alkalmazásokban, így ebben a programban is használhatjuk ezt a módot, mivel három fő oldallal fog majd a program rendelkezni. Ez azt jelenti, hogy a *főegyenleg* képernyőről eljuthatunk a *célok* oldalra és a *bevásárlás asszisztens* oldalra is, és ezeken az oldalakon ugyanúgy bármikor eljuthatunk vissza az egyenleg képernyőre is. Ezek tehát oda-vissza kapcsolatban vannak, mindenhonnan tudunk mindenhova menni a fő oldalak közt. Azonban a fő oldalakon kívül szükségünk lesznek majd egyéb oldalakra is. Ha az egyenleg teljes funkcionalitását nézzük, akkor ebből az oldalból tudnunk kell majd tranzakciót rögzíteni, és a tranzakció részleteit megnézni, így ezekhez a funkciókhoz egy-egy oldalt kell majd készíteni. Természetesen ebből ki is kell majd lépni, tehát itt is oda-vissza kell majd tudnunk lépkedni. A célok képernyőjén szükség lesz egy olyan oldalra, ahol új célt tudunk majd létrehozni, de a cél részleteit nem feltétlen szükséges megtekinteni, mivel ezt a kevés adatot a fő oldalon is meg tudjuk jeleníteni majd. A bevásárlás asszisztens felületéhez is szükség lesz majd egy olyan felületre, ahol a kosár elemeit tudjuk beszúrni és módosítani, de ezt véleményem szerint jobb lenne egy Dialog segítségével megvalósítani, ami segítségével az oldal meghívja ezt a dialógust, és a háttérben futásban marad az oldal. Ezzel elkészült a lapátmenet tervezete, ami a következőképpen néz ki:



5.1 ábra: Az alkalmazás lapátmenetei.

## 5.2. Az alkalmazás lapjainak bemutatása

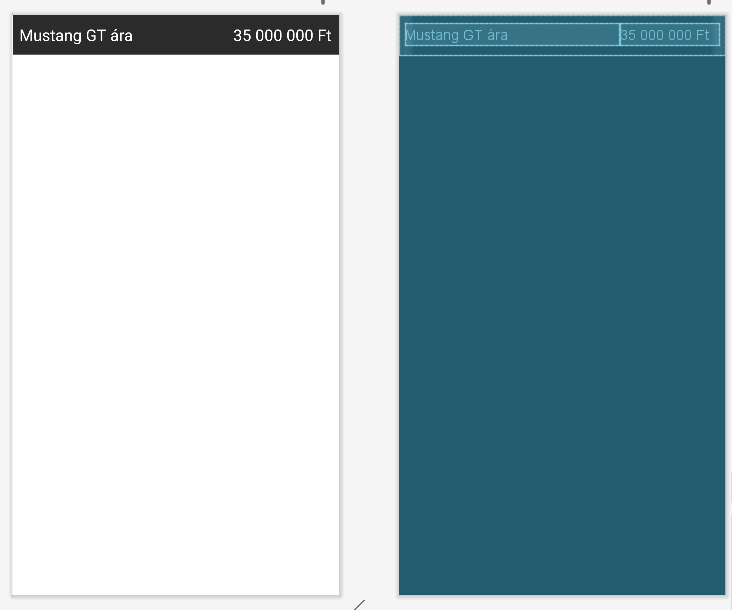
Az alkalmazást lehetőleg három komponensből szeretném elkészíteni, és mind a három fülnek különböző funkcionalitása lesz. A három fül a főegyenleg képernyő, ahol láthatjuk az egyenlegünket, megtakarítások képernyő, ahol a céljainkat adhatjuk meg, és a bevásárlás asszisztens, amikor vásárolni megyünk, egy külön asszisztenst kapunk a kezünkbe.

### 5.2.2. A főegyenleg képernyő

A legfőbb eleme a programnak, hiszen itt fogjuk majd látni a rendelkezésre álló egyenlegünket, és itt tudunk majd költéseket hozzáadni a programhoz, hogy tudjuk követni, hogy milyen dolgokra költöttünk eddig. A program előhozza a legutóbbi költések nevét, és a tranzakció összegét, időrendben megjelenítve azokat. A költést piros színnel látjuk, a pénzforrást pedig zölddel. Lehetőségünk van a kategóriák alapján rendszerezni a költéseinket, így csak az adott kategória tranzakcióit láthatjuk. A programozás design-ját többféle oldalról tudjuk megközelíteni, ezek lehetnek a Model-View-Presenter (MVP), Model-View-ViewModel (MVVM), Model-View-Intent (MVI) architektúrális felépítéseket választhatjuk. Az MVP fő előnye, hogy szétválasztja az alkalmazás problémáit, és megkönnyíti az egyes összetevők tesztelését és újrafelhasználását. Az MVVM fő előnye, hogy egyszerűvé teszi a View és a Model közötti kapcsolást, valamint tesztelhetőbbé és karbantarthatóbbá teszi a kódot. Az MVI fő előnye, hogy világos és kiszámítható adatáramlást hoz létre, amely könnyen értelmezhető és javítható. A nézet csak az állapot megjelenítéséért és az Intentek kibocsátásáért felelős [8]. Az előnyök és hátrányok mérlegelése után az MVVM mód látszik számomra a legjobb módszernek. A tranzakciókat éppen ezért egy adatbázisban kell eltárolni, amit a DAO (Data Access Object) segítségével tárolunk el. Ezzel egy lokális adatbázist tudunk létrehozni, amivel a felhasználó telefonján internethozzáférés nélkül is bármikor a rendelkezésére képes állni. Az adatbázis létrehozásához először egy Interface-t kell létrehoznunk, amiben definiáljuk az adatbázisban szükséges műveleteket, mint például azt, hogy minden elemet le tudjunk kérni:

@Query("SELECT \* FROM expenses order by id desc")  
fun getAll() : List<BalanceTransaction>

Ezzel a Query-vel minden eddigi tranzakciót le tudunk kérni, és ezt majd a felületen meg is fogjuk tudni jeleníteni. Ha bármilyen másféle műveletet végre szeretnénk hajtani az adatbázison belül, akkor az összes SQL parancsot egy ilyen függvény használatával tudjuk létrehozni. Ebben a DAO-ban helyet kapott a beszúrás, a módosítás, egy adott elem törlése, az elemek lehívása, a lehívás kategóriák alapján, és a megtakarítási egyenleget is ezen az adatbázison belül tudjuk lekérdezni. Ezeket a függvényeket később a ViewModel használatával tudjuk ráilleszteni a fő osztályunkra MVVM (Model – View – ViewModel) nézetmodell használatával, ahol a három elemet különválasztjuk egymástól ahhoz, hogy a rétegek szeparáltak legyenek, és könnyebb legyen a különböző rétegeken dolgozni. Ez úgy működik, hogy a View-en keresztül hívjuk meg az adatbázis műveletet a ViewModelt használva, majd a ViewModel a Modelben levő függvénnyel hívódik meg, összekötve ezzel a View-t a Modellel. A tranzakciók adatbázisát tehát egy Model osztályban kell létrehozni, maga a tranzakció majd egy példány lesz, aminek az elemeivel fogunk majd tudni dolgozni. A tranzakció modellének van id-je, neve, összege, kategóriája és egyéb leírása is, amit egy külön fülön tudunk majd megjeleníteni, ha a felhasználó rányom az adott tranzakcióra. Ezután következzék a felület megtervezése. A cím alatt legyen a fennmaradó egyenlegünk, hiszen az az egyik legfontosabb adat a számunkra. Utána a felhasználónak egy összegzést készítünk, hogy mennyi az adott kerete, és eddig ő abból mennyit költött el. Utána a tranzakciók megjelenítése következik. Ezt egy RecyclerView segítségével tudjuk megtenni, amivel ún. kártyákat tudunk megjeleníteni. A kártyát egy külön Layoutban tudjuk létrehozni. Egy ilyen kártya két TextView-ből áll egy LinearLayout-ban, ami a tranzakció nevét és a tranzakció összegét jeleníti meg egy sorban. Tehát a RecyclerView az ilyen Layouttal dolgozik, amit egy osztályban össze kell hangolni, és ebben az osztályban tudunk neki értékeket adni.



5.2. ábra: Így fog kinézni az egyenlegen megjelenő elemünk.

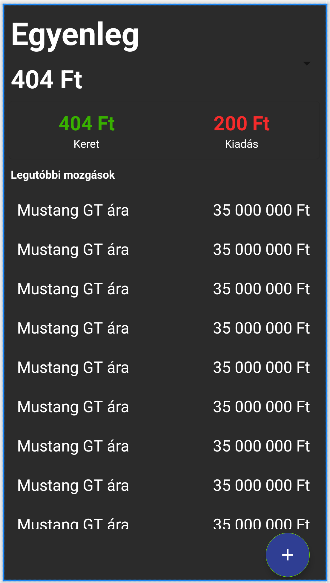
Magát a megjelenítés összekapcsolását a ViewHolder végzi, amit az Adapterben kell létrehozni:

class BalanceTransactionViewHolder(view: View) : RecyclerView.ViewHolder(view) {  
 val description : TextView = view.findViewById(R.id.*idTransactionLabel*)  
 val amount : TextView = view.findViewById(R.id.*idTransactionAmount*)  
}

A findViewById a Layout elemeire tud hivatkozni, és ezzel a metódussal tudjuk összekötni a vizuális megjelenést az osztállyal, és az onBindViewHolder függvényen belül tudunk neki értéket adni. Fontos megjegyezni, hogy a megjelenő elemeket az adatbázisból tudjuk lekérdezni a Balance osztályunkban, így a RecyclerView értékadása is itt történik meg. Mivel minden tranzakciónak egyedi értékei vannak, így a View-ban minden elemnek van egy pozíciója, amivel tudunk az adott tranzakcióra hivatkozni. Ez fontos is lesz a számunkra, mivel majd egy onClickListener-t is kell alkalmaznunk ahhoz, hogy a tranzakció elemeit egy külön képernyőn tudjuk megjeleníteni. Ez a beépített függvény arra szolgál, hogy a koppintásokat figyeli, és ha rányomunk egy tranzakcióra, akkor végrehajtódik, ami ebben a metódusban van.

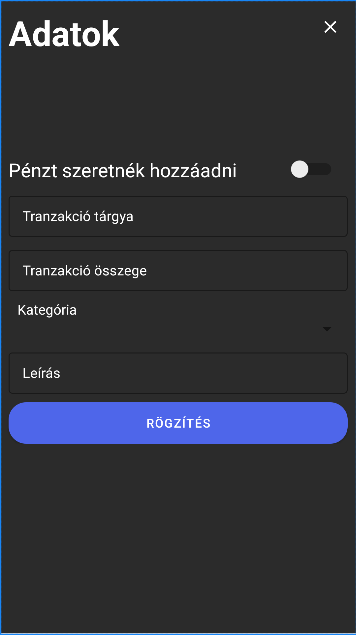
Amire még szükségünk lesz, az egy ActionButton, amivel majd meg tudjuk jeleníteni az új tranzakció rögzítésére szánt felületet.

Ezen elemek implementálásával elkezdhetünk dolgozni a fő Activityben, ami maga az egyenleg képernyőt adja, itt fog megjelenni minden, amit eddig létrehoztunk.



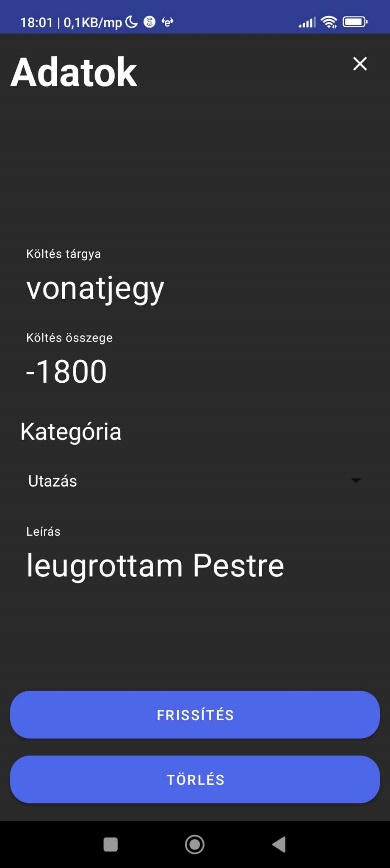
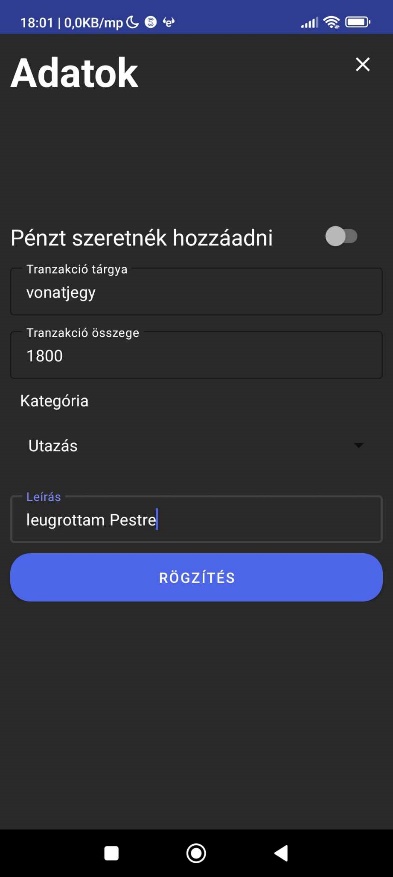
5.3. ábra: Így fog majd kinézni az egyenleg oldalunk.

Ezután hozzuk létre a tranzakció létrehozására szolgáló Activityt! Ha az ActionButtonhoz hozzákötjük a rögzítésre szolgáló Activityt, amit egy activity létrehozásával tudunk megtenni, ami létrehoz egy osztályt és a hozzá tartozó layout.xml fájlt, akkor a gomb lenyomásakor meg fog jelenni ez a képernyő, ahol a tranzakció adatait vagyunk képesek betáplálni:



5.4 ábra: A tranzakciókat ezen az oldalon tudjuk rögzíteni.

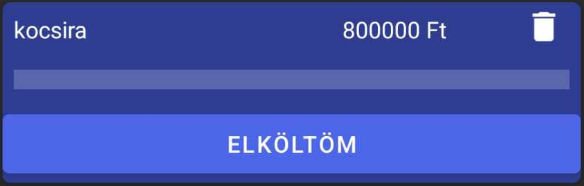
Itt már fontos ellenőrzéseket kell végrehajtanunk, mert ha a felhasználó rosszul adja meg az adatokat, akkor összeomolhat az alkalmazásunk, ha az adott hibát nem kezeljük le. Elsősorban van egy switch elem, amit ha igazra állítunk, akkor a program tudni fogja, hogy bevételi forrást szeretnénk hozzáadni a tranzakcióinkhoz, ezért majd pozitívként jelenik meg az összeg. Ezzel együtt a kategóriánk is megváltozik, és a kategória fizetésre változik. A tranzakció tárgyánál bármilyen szöveget meg lehet adni, de fontos kritérium, hogy ez a mező nem lehet üres. A tranzakció összege egy szám, tehát ha ide szeretnénk betáplálni adatot, akkor csak egész számot fogunk tudni beírni. A kategória kiválasztását egy spinner-rel oldottam meg, ami lényegében egy legördülő listát jelent, és a lista a kategóriák szöveges tömbjéből áll, amit előre definiáltam az alapadatokban. Itt kiválaszthatjuk a számunkra legmegfelelőbb kategóriát. Választhatjuk a mindennapi élethez tartozó kategóriát, ilyen például a bevásárlás, a hobbit, ha van egy hobbink, amire költöttünk, otthont is választhatunk, például, ha veszünk festéket, vagy befizetjük a rezsidíjat stb., akkor ez a kategória a megfelelő. Ha kikapcsolódunk, akkor azok a költések a szórakozás kategóriájába esnek, illetve, ha például levonatozunk a Balatonra, akkor a vonatjegyet az utazás kategóriába tehetjük. Ha pénzt szerzünk, az a fizetés kategóriába kerül. Külön kategóriát képvisel a megtakarítás, ennek a kategóriának a megtakarítások menüpontban lesz fontos szerepe.

5.5 ábra: A tranzakció rögzítése és a tranzakció részletei.

### 5.2.2. A megtakarítási felület

A program másik fontos eleme, hogy legyen egy olyan felület, ahol félre tudjuk tenni a pénzünket, és ezáltal elindulhassunk a megtakarítási célunk felé. Legyen az bármilyen cél, azt hasonló módon hozzá tudjuk adni a programhoz. Az activity felépítése nagyjából hasonló módon épül fel, mint az egyenlegé, ugyanúgy szükségünk lesz majd egy adatbázisra, ahol a céljainkat el tudjuk tárolni, tehát ismét definiálnunk kell egy DAO-t, és mellé a szükséges elemeket is, az adatbázist, az adaptert a recyclerview-höz ahhoz, hogy rendeltetésszerűen tudjuk tárolni az adatokat. Az egyenleges adatbázishoz képest azonban itt más változókra lesz szükség. A célunknak kell majd egy nevet adnunk, hogy tudjuk mi az a cél, amire költünk, illetve azt, hogy maga a cél milyen összegben nyilvánul meg, tehát mennyi pénzt kell megspórolnunk ahhoz, hogy a célunkat elérjük. A megtakarítás fül mellé elkészült a cél létrehozására szolgáló lap is, ahol meg tudjuk adni a cél nevét és az összeget, az integritási szabályoknak eleget téve. Ezeket az elemeket is egy recyclerview-en keresztül jelenítem meg, ahol egy cél egy kártyán jelenik meg.

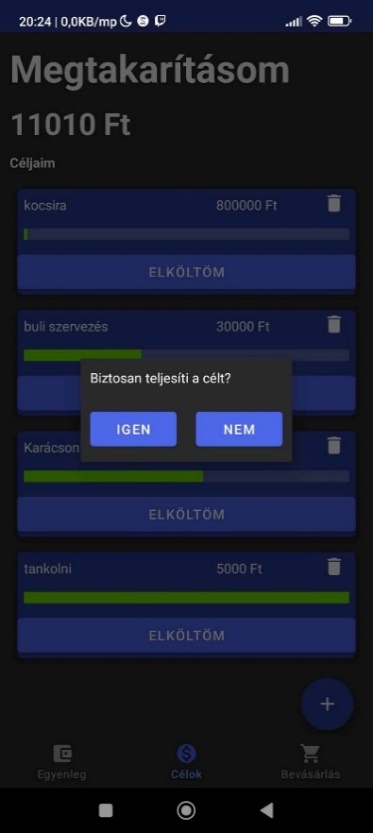


5.6 ábra: A célunk ilyen formában fog megjelenni.

Látható, hogy a kártyán megjelenik a cél neve és az összege. Középen egy progress bar mutatja meg nekünk azt, hogy a megtakarított egyenlegünk milyen arányban teljesíti a meghatározott összeget. Ezt úgy számolja ki az alkalmazás, hogy a megtakarítási egyenleget és a cél összegét elosztja egymással, majd a százalékos arányát képezi. Ha a megtakarításunk nagyobb, mint a cél összege, akkor a csík haladása 100% lesz, mivel az összeg egészével rendelkezünk. Azt, hogy mennyi megtakarításunk gyűlt össze eddig, azt a kategória segítségével tudjuk lekérdezni.



5.7 ábra: A megtakarítási felület.

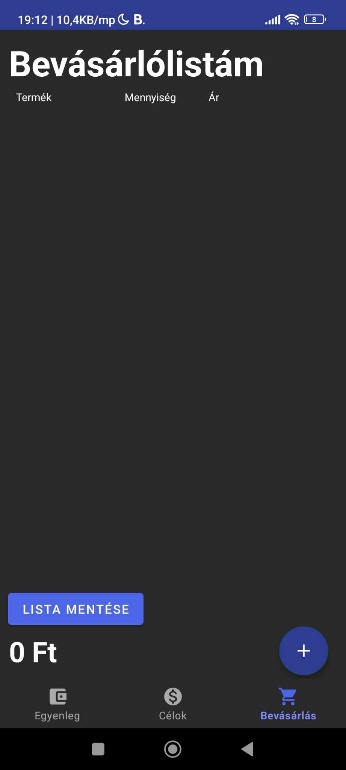
Az összegyűlt megtakarítás összegét úgy számolhatjuk ki, hogy az összes rögzített megtakarítás kategóriájú tranzakció összegét összeadjuk, és ebből kivonjuk a már teljesített célokat. A jobb oldali *kukára* nyomva törölhetjük az adott célt, ekkor nem látjuk többé azt a célt, az *elköltöm* gombbal pedig véglegesíthetjük azt, amivel le tudjuk vonni a megtakarításunkat, ha már valóban is sikerült elérnünk a célunkat.

5.8 ábra: Az elköltés gomb különböző interakciói.

Azonban itt még történik egy ellenőrzés, hogy rendelkezik-e a felhasználó a szükséges megtakarítással ahhoz, hogy ezt ki tudja-e váltani vagy sem. Ha nem elég a megtakarított egyenleg, akkor egy üzenet jelenik meg, ami tájékoztat bennünket arról, hogy még nem elég az a mennyiség, amit összegyűjtöttünk. Viszont, ha már kigyűlt az összeg amire szükségünk van, akkor egy dialógus jelenik meg, ahol meg kell erősítenünk a kiváltási szándékunkat. Ha itt is az igenre nyomunk, akkor a cél törlődik, és a cél összege levonásra kerül a megtakarítási egyenlegből, a főegyenlegre viszont nincs hatással. A céljaink úgy jelennek meg a felületen, hogy a legmagasabb költségű cél van legfelül, majd csökkenő sorrendben jelennek meg a további célok is. Természetesen ez a funkció akkor a leghatásosabb, ha a valós környezetben is le tudjuk tenni azt az összeget, amit megtakarításként tartunk számon, és ha elérjük a célunkat, akkor már csak elő kell venni azt, amit eddig megspóroltunk.

5.2.3. A bevásárlás asszisztens

Az egyik legérdekesebb dolog mikor vásárolni megyünk, az szerintem az, mikor csak vásárlunk és vásárlunk, majd egy kis idő után már nem tudjuk követni azt, hogy mekkora összegnél is tartunk. Szerintem sokan járnak így, elég csak abba belegondolni, hogy mennyi ember lepődik meg azon, hogy milyen összeg jelenik meg a pénztár előtt. Természetesen ennek sok oka lehet, de én leginkább azt az oldalt szeretném megfogni, ami az impulzus vásárlásokkal keletkeznek. Azt nevezzük impulzus vásárlásnak, amit nem szerettünk volna venni a boltba érkezés előtt, de aztán mégis a kosárban landolnak ezek a termékek. Sokszor még olyankor is előfordul az ilyen eset, mikor a vásárló egy bevásárlólistával megy el vásárolni, de arról ne is beszéljünk, hogy még rosszabb a helyzet akkor, ha csak úgy – bevásárlólista nélkül – megyünk vásárolni. Ennek a problémának a javítására ötleteltem ki a bevásárlás asszisztens felületet, amire a vásárlás közben be tudunk táplálni minden adatot, így folyamatosan tudjuk követni, hogy milyen összegnél tartunk, illetve, hogy mi mennyibe kerül a kosarunkban. Az elemeket egy adatbázisban tudjuk letárolni, így akár a vásárlás kigondolásakor már használhatjuk is a programot.



5.9 ábra: A bevásárlólista megnyitásakor ez a kép fogad minket.

Egy elem rögzítéséhez szükség van a termék nevére, a mennyiségre és az árára. Egy új terméket a lent található plusz gomb megnyomásával tehetjük meg, amivel előjön a termék rögzítésére szolgáló dialógus. A termék neve egy stringként kerül eltárolásra, így az bármilyen formátumú lehet, de fontos, hogy nem lehet üres, hiszen később bajban lennénk, hogy vettünk két darab nevenincs terméket, aminek 999 Ft az ára. A termék mennyiség eltárolását már érdekesebb kigondolni, hiszen mit írjunk be akkor, ha nem egész számú mennyiségű dolgot veszünk, hanem a termék például Ft/kg egységben van megadva, főleg a zöldségek és gyümölcsök? Éppen ezért a mennyiség egy lebegőpontos szám, tehát megadáskor képesek vagyunk valamiből akár 0,644 egységet is venni. A termék ára viszont egész számként kerül letárolásra, mivel az alkalmazás forintra lett tervezve, és manapság már nem értelmezünk filléreket.



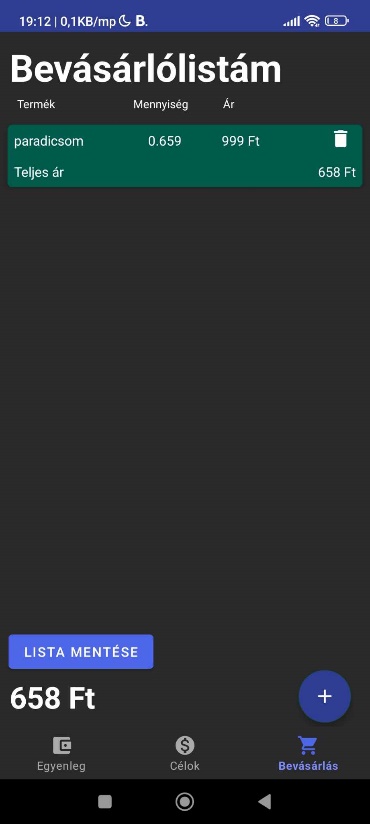
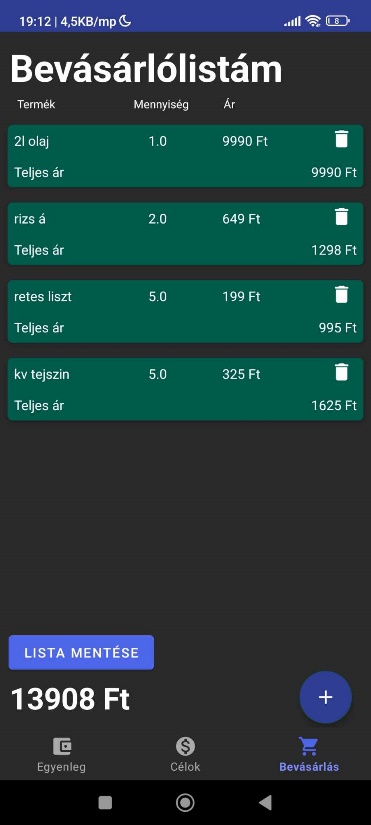
5.10 ábra: Egy termék adatainak rögzítése.

Ha rögzítjük a terméket, ezután a program kiszámolja, hogy ez összesen mennyibe fog nekünk kerülni, összeszorozva a mennyiséget a termék árával, nem egész mennyiség esetén kerekítést hajt végre. A termék árát a következő kód írja le:

val itemTotal : Int = (list[position].itemPrice \* list[position].itemAmount).roundToInt()

Természetesen ez még nem elég ahhoz, hogy teljesen nyomon tudjuk követni az eddigi költésünket, a felületen az összes termék árát is össze kell adnunk ahhoz, hogy megkapjuk, mennyinél jár a kosarunk értéke. Erre egy Textview nyújt megoldást, aminek a szöveges értékét minden adatváltozáskor frissítjük, összeadva benne az eddig megadott elemek árainak össszegét.

Természetesen van lehetőségünk a rögzítés után az adatok módosítására is, például, ha valami akciós és nem tudunk róla, akkor az árát tudjuk módosítani, illetve, ha valamiből többet szeretnénk venni, akkor a mennyiséget is módosíthatjuk. Ekkor, ha rányomunk egy termékre, akkor hasonló módon előjön a termék rögzítő dialógus, azonban a már megadott adatokat előre betölti számunkra, ezeket lehet módosítani.

A módosítás végrehajtása után nem hoz létre egy új kártyát az alkalmazás, hanem a meglévő kártya adatait módosítja.

5.12 ábra: Egy bevásárlólista vizualizációja.

Ha végeztünk a vásárlással, akkor a lista mentése gombra kattintva behozhatjuk a tranzakció rögzítő activityt, ami megvizsgálja, hogy üres listát akarunk-e menteni, és ha igen, akkor nem enged tovább lépni, azonban, ha rendben találja a listát, akkor előjön a felület, aminél a tranzakció összege automatikusan kiegészül a bevásárlólista összegével. Ha itt végig megyünk a tranzakció rögzítésén, akkor az eddigi bevásárlólista elemei eltűnnek, és egy új listát tudunk majd létrehozni.

# 6. Az elkészült alkalmazás tesztelése és tapasztalatok

Miután a programom implementálásával elkészültünk, elkezdhetjük tesztelni az alkalmazást. A programot a szüleim saját magam, és a szüleim segítségével, használat alapján került tesztelésre. A fejlesztés elkészülte óta napi szinten használatban van a program, és a bevásárlási listákat is ennek segítségével állítjuk össze. Néhány célt nekem is sikerült felállítanom, amikre a dolgozat írásakor szeretnék majd gyűjteni. Öröm látni, hogy a tervezés legtöbb része elkészült, és teljes mértékben egy használható alkalmazást tudtam leprogramozni.

6.12 ábra: A jelenleg használatban levő alkalmazás felületei.

# 7. További fejlesztési javaslatok

A működő alkalmazás elkészültével teljesnek mondható a program fejlesztése. Azonban mindig vannak egyéb lehetőségek, amivel még ki lehet egészíteni, fejleszteni tovább az alkalmazásunkat. Néhány ötletet még összegyűjtöttem, hogy mik azok az opciók, amikkel még a jövőben ki lehetne egészíteni a programot.

## 7.1. Dátumok implementálása minden felületre

Szerintem ez az egyik legnagyobb dolog, amivel még ki lehetne egészíteni a program működését. Mivel az időt egy külön elemként kell eltárolnunk az adatbázisban, ezért az adatbázist újra kellene építeni ahhoz, hogy a dátumot is el tudjuk tárolni a programban.

## 7.2. A határidők és az ismétlődő tranzakciók implementálása

Ennek implementálásával egy teljes költési módszert is lehetne majd implementálni, amivel a felhasználó jobban tudja figyelni, hogy a havi egyenlegéből mennyit költhet még a különböző kategóriára.

## 7.3. Több ösztönzés a megtakarításra

Jelenleg csak akkor tudunk megtakarítást létrehozni, ha kifejezetten az egyenlegből készítünk egy tranzakciót, amivel a megtakarításra helyezzük át az egyenlegünk egy részét. Ezt a funkcionalitást lehetne növelni például kerekítéssel, ha mondjuk valaki 13 700 Ft-os tranzakciót ment el, akkor a program feljajánlhatná, hogy kerekítéssel 14 ezer Ft-ra kiegészítve elhelyezze a kerekített összeget a megtakarítási számlára. Ugyanez a funkcionalitás implementálható lehetne a bevásárlás asszisztens oldalon is.

## 7.4. Több fül a bevásárlás asszisztensen belül

Önmagában ez nem jelentene nagy változást, de szerintem ez egy hatékonyabb bevásárlást tudna eredményezni, főleg akkor, ha egyszerre több boltba is szeretnénk menni egyszerre. Ekkor a boltokhoz külön-külön listát lehetne hozzáadni.

# Köszönetnyilvánítás

Ezúton is szeretném megköszönni a családomnak a rengeteg segítséget, amit az egyetemi éveim alatt kaptam, az egyetemi tanáraimnak, akitől rengeteg hasznos tudást kaptam, és Piller Imrének, aki rengeteg segítséget nyújtott az egyetemi tárgyaimban, illetve a szakdolgozat megírásában is rengeteg segítséget nyújtott.

# Forrásmegjelölés

[1] GnuCash alternatíva: https://alternativeto.net/software/gnucash/

[2] GnuCash alkalmazás: https://www.gnucash.org/

[3] Money lover alkalmazás: https://moneylover.me/

[4] OTP persely: https://www.otpbank.hu/portal/hu/Uj-IBMB-kisokos/Persely

[5] Kotlin nyelv: https://developer.android.com/kotlin

[6] Kotlin nyelv előnyei: https://www.techtarget.com/whatis/definition/Kotlin

[7] Android Studio: https://developer.android.com/studio/intro

[8] Android Architektúrák: https://medium.com/@alexandragrosu03/android-app-architecture-with-kotlin-mvp-mvvm-and-more-6faf7cd1d3cf