



Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Balla Péter Keve

**KIADÁSOKAT KEZELŐ
ALKALMAZÁS FEJLESZTÉSE
FLUTTER
KERETRENDSZERREL**

KONZULENS

Somogyi Norbert Zsolt

BUDAPEST, 2023

Tartalomjegyzék

Összefoglaló	5
Abstract.....	6
1 Bevezetés	7
1.1 Motiváció	7
1.2 A dolgozat felépítése	9
2 Felhasznált technológiák	11
2.1 Firebase Authentication	11
2.2 Firebase Firestore Database	11
2.3 SQLite	12
2.4 GitHub	12
2.5 Visual Studio Code	13
2.6 Flutter	13
2.7 Android Studio emulátor	14
3 Tervezés	15
3.1 Feladat specifikációja.....	15
3.1.1 Legfontosabb funkciók	15
3.1.2 Use Case diagram	16
3.2 A program architektúrája	17
4 Implementáció	20
4.1 A program logikai felépítése.....	20
4.1.1 Mappa struktúra, logika	20
4.1.2 Adatbáziskezelés.....	26
4.2 A projekt megvalósításának folyamata	29
4.2.1 verziókezelés.....	29
4.3 A program tesztelése.....	30
4.3.1 Manuális tesztelés	30
4.3.2 Tesztelt funkciók.....	31
4.3.3 Teszt felépítése	31
4.3.4 Példa teszt	31
4.4 A program használata	34
5 Összefoglalás.....	45

5.1 Értékelés.....	45
5.2 Továbbfejlesztési lehetőségek	45
6 Köszönetnyilvánítás	47
Irodalomjegyzék.....	48

HALLGATÓI NYILATKOZAT

Alulírott **Balla Péter Keve**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2023. 12. 07.

.....
Balla Péter Keve

Összefoglaló

A szakdolgozatom témája egy olyan alkalmazás tervezése és megvalósítása, amely a pénzügyi tervezés és költségkezelés egyszerűsítését célozza meg. A kiinduló pontom az volt, hogy nem találtam számomra megfelelő alkalmazást a pénzügyeim vezetésére, márpedig ez egy igen fontos része egy felelős ember életének. Erről részletesen a motivációm cím alatt írok.

A probléma felismerése alapján elhatároztam, hogy egy olyan alkalmazást tervezek és valósítok meg, amely segít a felhasználóknak a pénzügyi tervezésben, és mindezt egy nagyon felhasználóbarát módon. Az alkalmazás neve BuXa, és a Flutter keretrendszer segítségével íródott, lehetővé téve az elérhetőségét mind weben, mind Android platformon. Az alkalmazás funkciói között szerepel a bevételek és kiadások nyomon követése, különféle zsebek létrehozása a kényelmes és átlátható rendszerezéshez, valamint a tartozások kezelése, beleértve a felvezetett emberek közötti tartozások optimális megadási módjának kiszámítását. Ezeken kívül különböző lekérdezések végrehajtását lehet elvégezni az alkalmazással.

Abstract

The topic of my thesis is the design and implementation of an application aimed at simplifying financial planning and expense management. My starting point was the lack of a suitable application for managing my finances, which is an important aspect of a responsible individual's life. I elaborate on this in detail under the title of my motivation.

Based on the recognition of the problem, I decided to design and implement an application that helps users with financial planning in a very user-friendly manner. The application is named BuXa and is written using the Flutter framework, allowing its accessibility both on the web and on the Android platform. The features of the application include tracking income and expenses, creating various pockets for convenient and transparent organization, and managing debts, including calculating the optimal method of entering debts between introduced individuals. In addition, various queries can be executed with the application.

1 Bevezetés

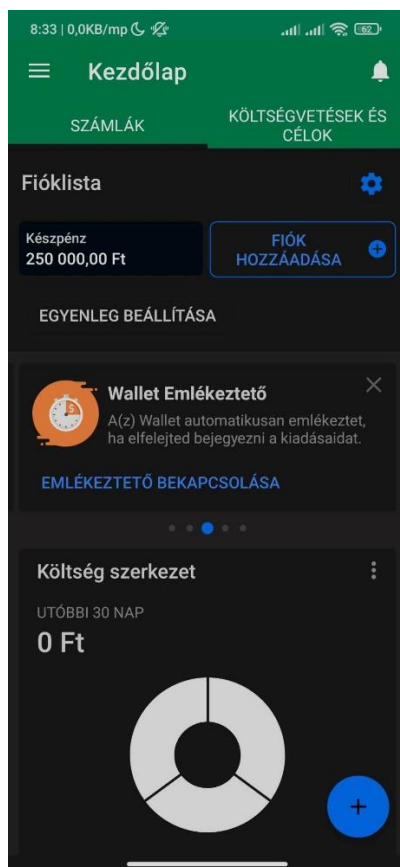
1.1 Motiváció

Motivációm az alkalmazás fejlesztése iránt a pénzügyeim tudatos kezelésére való elkötelezettségemből ered. Az anyagi felelősségvállalás iránti érdeklődésem már fiatalon, 8. osztályos koromban kezdődött. Ez az érdeklődés az én nagypapámtól származik, aki szintén gondosan kezelte pénzügyeit. Az első próbálkozásaimat egy egyszerű Excel táblázatban tettem, de sajnos az adatok egy telefonhiba miatt elvesztek.

Az eset tanulságát felismerve, Google táblázatokat kezdtem el használni, így az adataim szinkronizálva voltak a Google Drive-fal, és biztonságban maradtak. Ekkor már elterveztem, hogy létrehozok egy alkalmazást, amely automatikusan beolvassa és értékeli a pénzügyeimet.

Ezen elképzelésemet az motiválta, hogy nem találtam olyan alkalmazást, amely teljes mértékben testreszabható lenne számomra. Más alkalmazásokban hiányzott a zsebek létrehozásának lehetősége, vagy túl sok olyan felesleges funkció volt, amelyeket nem lehetett kikapcsolni, ami zavaróvá tette a használatukat. Példaként megemlíteném a Wallet-et, mint az egyike legnépszerűbb "payment tracker" mobilalkalmazás. Ebben az alkalmazásban, számomra szükségtelennek tűnő, nem kikapcsolható funkciók közé tartoznak például a garanciák nyilvántartása, üzleti pénztárcák kezelése, törzsvásárlói kártyák beállításai és az alkalmazáshirdetések, melyek mindig felugranak és elvonják a felhasználó figyelmét.

Az alkalmazás fő oldalán található gombok közül igen kevés az, amelyeket gyorsan vagy rendszeresen használnék. (1. ábra) A lényeges funkciókért mélyebben kell elmerülni az alkalmazás struktúrájában, és azoknak elérése érdekében több lépés szükséges.



1. ábra: Wallet főmenü.

Ez a példa rámutat arra, hogy a funkcionalitás túlzott diverzifikációja vagy az egyszerű elérhetőség hiánya csökkentheti az alkalmazás hatékonyságát és felhasználói élményét. Azt tapasztaltam, hogy az alkalmazás egyszerűségének hiánya frusztrációhoz vezet a felhasználóban, ezáltal kevésbé használja szívesen az alkalmazást, ami odáig is vezethet, hogy abbahagyja használni, vagy másik alkalmazást keres, ami jobban kielégíti az igényeit. Ezért tervezésekor igyekeztem gondoskodni arról, hogy a fő funkcionalitások könnyen elérhetők legyenek, és az alkalmazás használata ne legyen túl bonyolult vagy időigényes a felhasználók számára.

Az elkészült alkalmazás belépési lehetőséget biztosít egyedi e-mail cím és jelszó használatával. Az alkalmazás tartalmaz egy funkciót, amely lehetővé teszi különböző tartozások felvezetését, melyek személyek között optimalizáltan oszthatók szét. Ezen túlmenően az alkalmazásban részletes információk tárolhatók a személyekről, mint például a saját nevük, e-mail címük, és az esetleges Revolut-felhasználói státuszuk.

Az alkalmazás további funkcionalitása magában foglalja a bevételek és kiadások nyomon követését, a zsebekben történő pénzügyi tranzakciók vezetését, valamint lekérdezések végrehajtásának lehetőségét. Ezen lekérdezéseket különböző paraméterekre, például dátumra, összegre, zsebre lehet szűrni, biztosítva ezzel az átlátható és személyre szabott adatmegjelenítést.

Az alkalmazás fejlesztése során a Firebase SDK-t [10] alkalmaztam a felhasználói autentikációhoz és a Firestore adatbázishoz, amely lehetővé teszi a hatékony és biztonságos adatkezelést. A fejlesztést a Flutter nyílt forráskódú UI keretrendszerével hajtottam végre, ahol a programozási nyelv a Dart [14] volt. A kódbázist az MVVM Repository (Model-View-ViewModel) tervezési mintával készítettem el, biztosítva ezzel a kód struktúrájának tisztaságát és könnyű karbantarthatóságát a különböző rétegek szétválasztásával (ViewModel, Repository).

1.2 A dolgozat felépítése

Szakedolgozatom kezdetén részletesen kifejtem a motivációm hátterét, amely az alkalmazás elkészítésének hajtóerejeként szolgált. Bemutatom, miért esett választásom éppen erre a témára, és milyen problémákat igyekszem megoldani az alkalmazás fejlesztésével.¹

Ezt követően áttekintem a felhasznált technológiákat és azok választásának indokait. Röviden ismertetem, miért esett választásom konkrét eszközökre, és hogyan járultak hozzá a projekt sikeréhez.²

Itt részletesen specifikálom az alkalmazás főbb funkcióit. A use case diagram segítségével szemléltetem a felhasználók lehetséges interakcióit, továbbá elemzem az alkalmazás architektúráját.³

A Feladat Megoldása (Implementáció) részben részletesen leírom a program logikai felépítését, az adatbáziskezelést, és bemutatom a tesztelés folyamatát. Mindezen lépéseket vizuális elemekkel is alátámasztom a könnyebb megértés érdekében.⁴

A Program Használata fejezetben ismertetem, hogyan lehet hatékonyan használni az elkészült alkalmazást, bemutatva az interfészt és kiemelve az alkalmazás főbb funkcióit.^{4.4}

Az Értékelés cím alatt személyes értékelést adok saját munkámról. Elemzem, milyen tanulságokat vontam le a projekt során, milyen tapasztalatokat szereztem, és

miben fejlődtem. E rész számottevő információkat tartalmaz a fejlődésemről. Itt kimelem az alkalmazás azon területeit, amelyeken további fejlesztési lehetőségek rejlenek.⁵

Végül, a Köszönetnyilvánításban hálás köszönetet fejezek ki mindazoknak, akik segítettek és támogattak a szakdolgozat elkészítésében. Az Irodalomjegyzékben felsorolom a dolgozat elkészítéséhez felhasznált irodalmi és forrásmunkákat.⁶

2 Felhasznált technológiák

Az alábbi fejezetben kifejtem, hogy milyen fontosabb technológiákat használtam az alkalmazásom megalkotásakor, valamint leírom, hogy az app milyen területein volt szükség ezekre.

2.1 Firebase Authentication

A legtöbb alkalmazásnak szüksége van a felhasználó személyazonosságának igazolására és az azonosításra. Ez biztosítja az alkalmazás számára a felhasználó adatainak biztonságos tárolását, és lehetővé teszi, hogy a személyre szabott élményt konzisztensen kínálja a felhasználó összes eszközén. A Firebase Authentication lehetőséget biztosít az alkalmazásra való regisztrációra, regisztrációt követő e-mailes megerősítésre, bejelentkezésre, kijelentkezésre, valamint e-mail és jelszó módosításra. A regisztráció és a későbbi bejelentkezés során többféle azonosítási módszer áll rendelkezésre, például e-mail, telefon, Google vagy Facebook alapú azonosítás. Én az alkalmazásomhoz az e-mai címes és jelszavas bejelentkezést választottam. Ennek oka, hogy szükségem volt az alkalmazásban egy egyedi azonosítóra, ami ebben az esetben a bejelentkezett felhasználó e-mail címe. [1]

2.2 Firebase Firestore Database

A Cloud Firestore egy rendkívül alkalmazkodó, könnyen skálázható adatbázis, amelyet a Firebase és a Google Cloud mobil-, web- és szerverfejlesztések során alkalmaznak. Felhőalapú és valós idejű működés jellemzi, azaz adatait szinkronban tartja az adatbázis használói között valós idejű figyelőkön keresztül. Emellett offline támogatást is nyújt, lehetővé téve érzékeny alkalmazások létrehozását, amelyek zökkenőmentesen működnek a hálózati késleltetéstől vagy az internetkapcsolat hiányától függetlenül. Ezen tulajdonságoknak köszönhetően a Cloud Firestore ideális választás az olyan projektek számára, amelyek rugalmasságot, megbízhatóságot és valós idejű adatkezelést igényelnek. A Cloud Firestore NoSQL adatbázis, ami rugalmas dokumentumorientált modellt alkalmaz, nem relációs adatstruktúrával. A NoSQL [4] adatbázisok előnye, hogy alkalmazkodóképesebbek és könnyebben skálázhatók változó adatstruktúrákhoz és nagyobb adatmennyiségekhez. Ez a rugalmasság lehetővé teszi a

gyorsabb fejlesztést és a hatékonyabb adatkezelést dinamikus vagy változó adatok esetén. [2]

2.3 SQLite

A Flutter SQLite egy hatékony és könnyen használható adatbáziskezelő megoldás, amelyet a Flutter keretrendszerrel való mobilalkalmazás-fejlesztés során alkalmaznak. A SQLite adatbázis motorra épül, amely egy könnyű, beépített relációs adatbázis, így a Flutter [7] fejlesztőknek lehetőségük van egyszerűen és hatékonyan tárolni, lekérdezni és frissíteni az adatokat az alkalmazásukban. A Flutter SQLite technológia előnyei közé tartozik a platformfüggetlenség, mivel a Flutter maga is egy cross-platform keretrendszer, így az alkalmazások egyszerre futtathatók iOS-en és Androidon. (weben sajnos nem működik) Az SQLite használata lehetővé teszi az adatok helyi tárolását, ami különösen előnyös offline működés vagy adatok gyors elérésének szükségessége esetén. Az én alkalmazásom esetében a Flutter SQLite technológiát kizárólag a mobil Android platformon alkalmazom annak érdekében, hogy az alkalmazás offline módban is hatékonyan működjön. Ennek előnyei közé tartoznak a következők: gyors adatelérés, egyszerű implementáció, offline funkcionalitás. [3]

2.4 GitHub

A GitHub egy kollaboratív fejlesztési platform, melyet a fejlesztők világszerte használnak projektjeik verziókezelésére és együttműködésük elősegítésére az egyik legnépszerűbb eszköz a programozók körében. A GitHub egy webes platform és hosting szolgáltatás, amely lehetővé teszi a fejlesztők számára, hogy kövessék és közösen dolgozzanak együtt projektjeiken. Az alapvető funkciók közé tartozik a verziókezelés, a problémamegoldás, az ágazatkezelés és a kódkiemelés. Ezekből én a verziókezelést és az ágazatkezelést használtam a dolgozatom írásához. A verziókezelés révén könnyen visszavonhatók a módosítások, és minden fejlesztő tudja, mi az aktuális verzió. (amennyiben többen dolgoznak egy projekten) Az ágazatkezelés segítségével párhuzamosan is dolgozhatnak különböző funkciókon, mielőtt azokat a fő kódbázisba „main” ágra integrálnák. A GitHub egyúttal szolgálhat személyes portfólióként is. A fejlesztők bemutathatják kódjaikat, projektmunkáikat és az általuk elért eredményeket, ami segíthet az álláskeresésben vagy új üzleti lehetőségek teremtésében. [6]

2.5 Visual Studio Code

A Visual Studio Code (VSCode) a modern, könnyűsúlyú, és kiterjedt szövegszerkesztő és fejlesztőkörnyezet, amely széles körben elterjedt a fejlesztők között, nem kivétel innen a Flutter alkalmazások készítői. A VSCode rendelkezik olyan kiegészítőkkel, amelyek lehetővé teszik a hatékony Flutter fejlesztést. A Flutter és Dart kiterjesztések széles skáláját kínálja, amelyek segítik a fejlesztőket a kódírásban, hibakeresésben és az alkalmazások gyors és hatékony építésében. A VSCode beépített Git integrációval rendelkezik, ami lehetővé teszi a projekt verziókezelését és együttműködését a Git segítségével. A verziókezelési funkciók segítik a változtatások nyomon követését, visszavonását, és könnyű együttműködést biztosítanak. A VSCode kiváló kódkiemelési funkciókkal rendelkezik, amelyek segítik a fejlesztőket a kód áttekinthetőségében és a könnyű olvashatóságban. Az intelligens kódkiegészítés gyorsabbá teszi a kódírást, és segít elkerülni a tipográfiai hibákat. Ezenkívül támogatja a gyors és hatékony hibakeresést. A beépített hibakereső és a tesztelési eszközök segítenek az alkalmazások hibamentes fejlesztésében. Az integrált terminál lehetővé teszi a parancssoros műveletek elvégzését közvetlenül a fejlesztőkörnyezeten belül. Ez elősegíti az alkalmazások futtatását és tesztelését egyetlen ablakban. [8]

2.6 Flutter

A Flutter egy nyílt forráskódú mobilalkalmazás-fejlesztő keretrendszer, melyet a Google készített, és kifejezetten a gyors és hatékony crossplatform-ú mobilalkalmazások létrehozására szolgál. A fejlesztői közösségben széles körű népszerűségnek örvend, és a rendszer fontos jellegzetességei miatt számos előnnyel jár a programozók számára.

A Flutterben az alkalmazásokat widgetek segítségével építjük fel, amelyek önálló, kis részegységek és az UI-elemek képviselői. A widgetek moduláris alkotók, amelyeket könnyedén kombinálhatunk és testreszabhatunk az alkalmazás egyedi igényei szerint.

Ezenkívül a Flutter lehetővé teszi a crossplatform-ú fejlesztést, így a fejlesztők egyszerre hozhatnak létre és futtathatnak Androidra és Web-re optimalizált alkalmazásokat. (ezen kívül IOS-re is) Ez jelentős időmegtakarítást és erőforrások hatékonyabb felhasználását eredményezi. Az élő változtatások lehetőségét kínáló "Hot Reload" funkció egyedi fejlesztői élményt nyújt, lehetővé téve az azonnali visszajelzést és a gyors iterációt a kód fejlesztése során. A Flutter segítségével könnyedén

kialakíthatók gyönyörű és testreszabható felületek, a fejlesztők pedig könnyen integrálhatnak platformspecifikus kódot az alkalmazásokba. A keretrendszer további előnyeit az optimális teljesítmény, a kiterjedt fejlesztői közösség és a platformfüggetlenség teszi teljessé. A Flutter segítségével modern, hatékony és crossplatform-ú alkalmazásokat készíthetünk, miközben élvezhetjük a fejlesztés során nyújtott széleskörű támogatást és kényelmes eszközöket. [7]

2.7 Android Studio emulátor

Az Android Studio Emulátor az Android alkalmazásfejlesztés kiemelkedő eszköze, amely lehetővé teszi a fejlesztők számára, hogy könnyedén teszteljék és finomítsák alkalmazásaikat a különböző Android eszközökön és verziókon anélkül, hogy fizikai eszközre lenne szükségük. Az emulátor számos előnnyel jár, amelyek kiemelik azt a fejlesztők számára.

Az Android Studio Emulátor lehetővé teszi a különböző Android verziók, képernyőméretek és felbontások szimulálását. Ez nagyban hozzájárul a fejlesztői folyamat hatékonyságához, mivel az alkalmazások különböző környezetekben történő próbálgatása lehetővé teszi a széles körű kompatibilitás biztosítását.

A fejlett kezelési funkciók, mint például a forgatás, a képernyőfelbontás változtatása és az érzékelőszimuláció, lehetővé teszik a fejlesztők számára, hogy valós körülmények között teszteljék az alkalmazások viselkedését anélkül, hogy fizikai eszközt kellene használniuk. Ez a rugalmasság fontos szerepet játszik az alkalmazások optimalizálásában és a felhasználói élmény finomhangolásában. [9]

3 Tervezés

Ebben a szakaszban bemutatom az alkalmazás fejlődésének lépéseit az ötleteléstől kezdve egészen az implementációig. Részletezem az eredeti terveimet, amelyeket a szakdolgozatom témájában megvalósítani kívántam, és beszámolok azokról a tervekről, amelyek végül nem kerültek megvalósításra. Emellett részletesen kifejtem, hogy milyen alkalmazásokból merítettem inspirációt, és ezek milyen módon befolyásolták a projektet. Továbbá, részletesen elemzem az alkalmazás architektúráját, kihangsúlyozva azokat a tervezési döntéseket, amelyek a projekt alapját képezik. Ennek során érintem a különböző komponensek közötti kapcsolatokat és az alkalmazás belső struktúráját, hogy átfogó képet adjak az alkalmazás fejlesztési folyamatáról és szervezeti felépítéséről.

3.1 Feladat specifikációja

3.1.1 Legfontosabb funkciók

Az alkalmazás tervezése során kiemelten fontosnak tartottam néhány alapvető funkciónak a megvalósítását. Ezek közé tartozott az egyedi zsebek létrehozása, amelyekben könnyedén nyomon követhető a pénzügyek alakulása, és amelyekből egyszerűen lehet költeni. Emellett kulcsfontosságú volt számomra a lekérdezések lehetősége, amelyek segítségével a felhasználó bármely kiválasztott időintervallumban lekérdezheti az összes kiadást és bevételt. Ezáltal lehetőség nyílik kulcsszavak vagy a tranzakciókhoz tartozó megjegyzések alapján könnyen keresni és adott esetben módosítani.

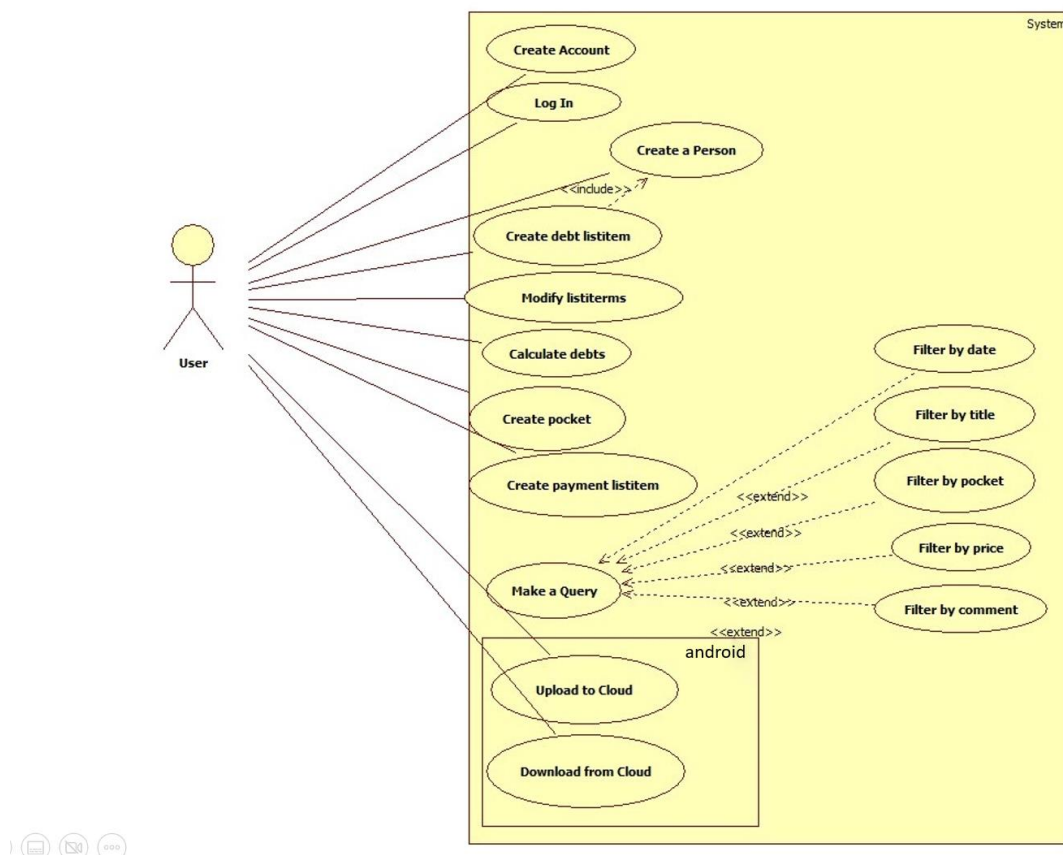
Az alkalmazás tervezésekor mindenképpen szükségesnek éreztem, hogy ne csak lokálisan tárolja az adatbázist, hanem kihasználja valamilyen felhőszolgáltatás előnyeit. Így a felhasználók számára lehetővé váljon az adatok fel- és letöltése, amely további kényelmet biztosít.

Továbbá, kiemelt figyelmet terveztem fordítani a tartozások egyszerű és okos szétosztásának lehetőségére. Amennyiben valamilyen eseményen, programon sok tartozást kell kezelni, az alkalmazás segítségével egyetlen gombnyomással könnyedén kiszámolható legyen, hogy az egyes személyek mennyit tartoznak, és ki és mennyit

fizessen, hogy mindenki megadja és megkapja a neki járó pénzt, mindezt úgy, hogy a lehető legkevesebb tranzakció menjen végbe. Ezáltal a pénzmozgás minimálisra csökkenthető, és a folyamat hatékonyabbá tehető.

Végezetül, ami számomra az egyik legfontosabb, az alkalmazás felhasználói élménye érdekében nagy hangsúlyt szerettem volna fektetni egy egyedi, egyszerű, esztétikus és letisztult UI felület kialakítására. Azt vallom, hogy a felhasználóknak az alkalmazás használata során kellemes és zökkenőmentes élményt kell nyújtania, és ennek érdekében igyekeztem az intuitív tervezési elveket követni.

3.1.2 Use Case diagram



2. ábra: Use case diagram.

A felhasználói scénáriókat és funkciókat bemutató fentebb látható (2. ábra) use case diagram alapján a következő use case-ket, főbb funkciókat definiálja az alkalmazás:

- **Felhasználókezelés:**

- A felhasználó képes új fiók létrehozására.
- Bejelentkezés a Firebase szerverre meglévő felhasználóval.
- **Listaelemek kezelése:**
 - Létrehozhat tartozás elemet, ami egyúttal új személyt is létrehoz.
 - Külön is létrehozhat személyeket.
 - Létrehozhat tranzakciókat, beleértve bevételeket és kiadásokat.
 - Létrehozhat zsebeket, amikben a bevételek és kiadások szerepelhetnek.
 - Módosíthatja a listaelemeket, ideértve a személyeket, tartozásokat és tranzakciókat. Ezeket törölheti és frissítheti a felhasználó
- **Tartozás számítások és kalkulációk:**
 - Kalkulálhatja az optimális tartozás megadás módját.
- **Adatlekérdezések és szűrés:**
 - Lekérdezheti a tranzakciókat a megadott időszakra.
 - Szűrheti a tranzakciókat két dátum között.
 - Szűrhet cím alapján.
 - Szűrhet zseb alapján.
 - Ár alapján is szűrheti a tranzakciókat.
 - Megjegyzések alapján történő szűrés.
- **Szinkronizáció és adatátvitel:**
 - Android platformon lehetőség van adatok feltöltésére és lekérdezésére a lokális és a szerver oldali tárhely között.
 - Weben az adatok automatikusan szinkronizálódnak a lokális és szerver oldali tárhelyek között.

3.2 A program architektúrája

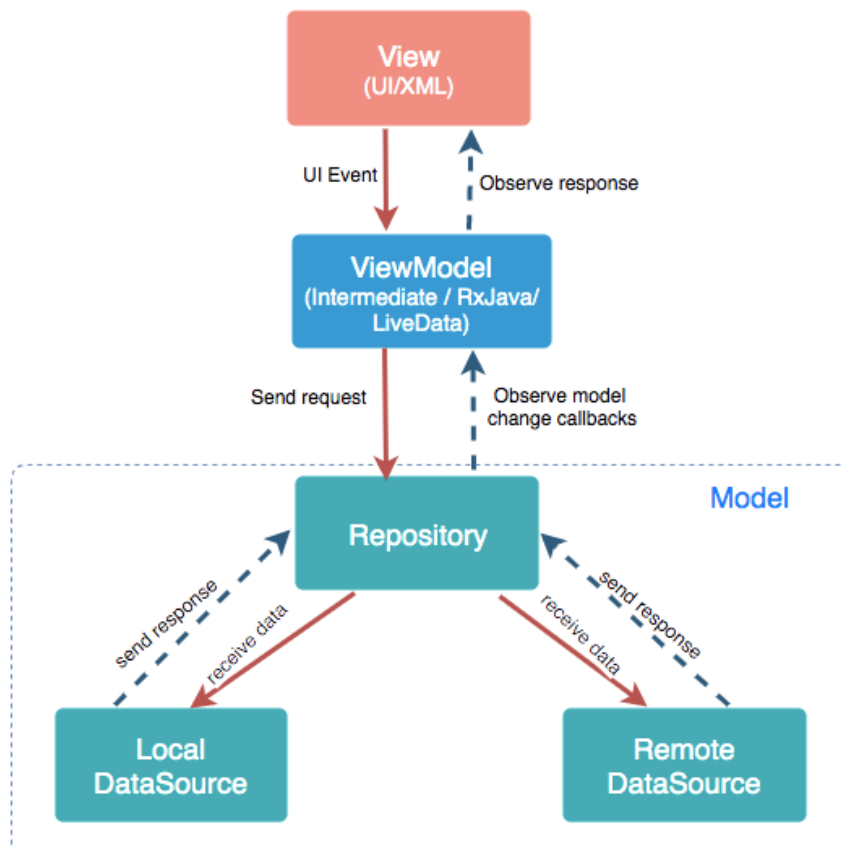
Az alkalmazásom fejlesztésénél az MVVM és a Repository mintát/architectúrát [12] követtem.

Az MVVM (Model-View-ViewModel) architektúra főbb céljai:

- Elválasztás: Különválasztja az alkalmazás logikáját, az adatokat és a felhasználói felületet.
- Könnyű karbantarthatóság: Egyszerűsíti a kód bázis karbantarthatóságát és fejlesztését. Könnyebb megtalálni mi hol van.
- Tesztelhetőség: Fokozza a kód tesztelhetőségét.
- Újrafelhasználhatóság: Lehetővé teszi az osztályok és komponensek könnyű újrafelhasználását.

Repository architektúra főbb céljai:

- Adatelérés absztrakciója: Elválasztja az alkalmazás logikáját az adatelérés részleteitől.
- Könnyű cserélhetőség: Lehetővé teszi az adatforrások könnyű cserélhetőségét anélkül, hogy az alkalmazás logikáját érintené.
- Tesztelhetőség: Fokozza az adatelérési réteg tesztelhetőségét.



3. ábra: MVVM, Repository Architektúra [5]

View:

A View a felhasználói felület megjelenítéséért felel, és fogadja a felhasználói interakciókat. Üzleti logikát nem tartalmaz, csupán a felhasználói felületet reprezentálja és kommunikál a ViewModel réteggel.

ViewModel:

A ViewModel az üzleti logikát közvetíti a View és a Model rétegek között. Transformálja az adatokat, amelyeket a View kér vagy amelyeket a Model réteg szolgáltat. Kapcsolódik a View-hoz és a Modelhez is.

Model:

A Model réteg az alkalmazás üzleti logikájáért felel, beleértve az adatmanipulációkat és adatkezelést. Kommunikál a ViewModel réteggel.

Repository:

A Repository réteg az adatelérés absztrakcióját jelenti, elválasztva az alkalmazás üzleti logikáját az adatelérési részletektől. Kommunikál a lokális (SQLite) és távoli (Firebase Database) adatforrásokkal, és biztosítja az adatokat a ViewModel számára.

Local Data Source (SQLite):

A Lokális Adatforrás (SQLite) a Repository réteggel kommunikál, felelős a lokális adatok tárolásáért és lekérdezéséért, például egy SQLite adatbázis használatával.

Remote Data Source (Firebase Database):

A Távoli Adatforrás (Firebase Database) a Repository réteggel kommunikál, és felelős a távoli adatok lekérdezéséért és frissítéséért, jelen esetben egy Firebase adatbázis használatával.

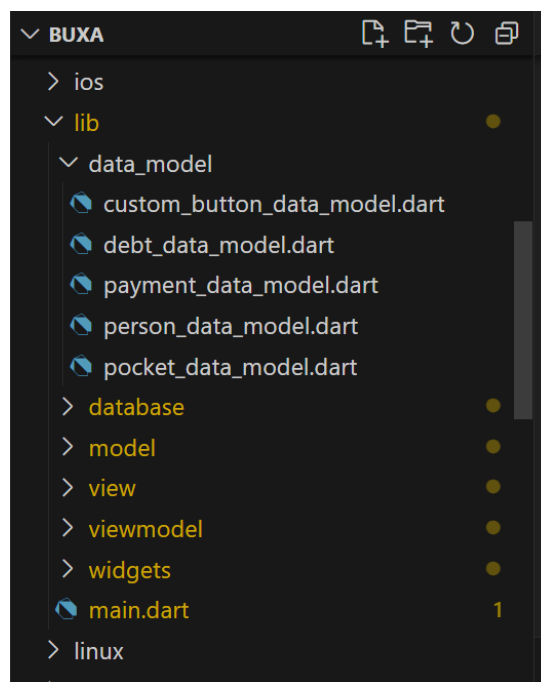
4 Implementáció

4.1 A program logikai felépítése

Ebben a szakaszban a tervezési döntéseket és az alkalmazás logikai felépítését fogom elemzeni. Részletesen kifejtem, hogy miért döntöttem az adott logikai struktúra, osztálystruktúra, objektumorientált megközelítés és szétválasztások mellett, és miként szolgálják ezek a döntések az alkalmazás hatékony működését.

4.1.1 Mappa struktúra, logika

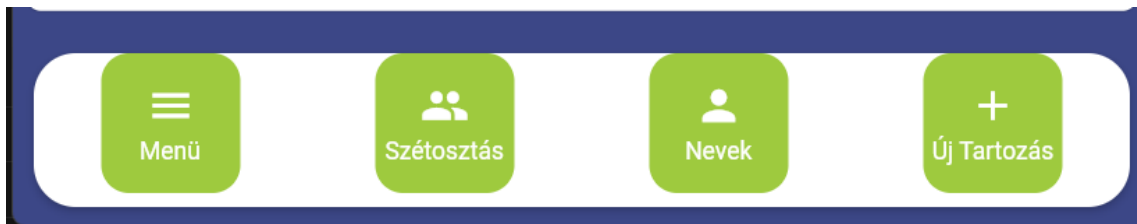
Az osztályaimat különböző könyvtárakban rendszereztem el (4. ábra), melyek a "lib" mappában találhatók.



4. ábra: Mappa struktúra.

A "data_model" mappában elhelyeztem az adatmodelljeimet, mint például a "pocket_data_model", "person_data_model", "payment_data_model", "debt_data_model" és a "custom_button_data_model". Ebben az utóbbi osztályban definiálom a gombok tulajdonságait, mint például a szín, ikon, cím és a paraméterként kapott funkció. Ennek segítségével minden olyan gomb, ami ebből az osztályból származik, rendkívül hasonló lesz, tiszteletben tartva az objektumorientált elveket,

minimalizálva a kódismétlést és egyszerűsítve a fejlesztő munkáját. Ezek a gombok a "desk" widgeteken helyezkednek el az alkalmazásban. (5. ábra)



5. ábra: Desk widget.

Az adott modellek (beleértve a többit is) az adattípusokat reprezentálják, melyeket később a kódban egyszerűen fel lehet használni.

A "database" mappában található a "repository" osztályok, melyek felelősek a lokális adatbázisba történő írás és olvasás menedzseléséért. Négy táblát kezelnek: tartozások, [költségek és bevételek], személyek és zsebek. Minden tartozáshoz két személy tartozik, és ezeket az adatmodelljeimben külső kulcsokkal reprezentálom, amelyek azoknak a személyeknek az azonosítójára mutatnak, akikhez az adott tartozás kapcsolódik. Hasonlóan, bevételek és kiadások esetén is van egy zsebazonosító, amely egy külső kulcs, és az adott tranzakcióhoz tartozó zseb kulcsára mutat.

A „view” könyvtár a nézeteket tartalmazza, melyek a Flutter UI keretrendszer widget-jeiből vannak összeállítva. Az alkalmazás Menü képernyőjén egy kis különlegességet találunk: ha az alkalmazás webes felületen van megnyitva, két gomb elrejtődik a Menü képernyőről. Ezek a gombok a „feltöltés” és „letöltés”, és azért vannak elrejtve, mert webes felület esetén az alkalmazás automatikusan a Firestore szerverre tölti fel az adatokat, míg Android operációs rendszer esetén először lokálisan tárolja azokat. Ez a differenciálás biztosítja az alkalmazás optimális működését a különböző platformokon. (kód)

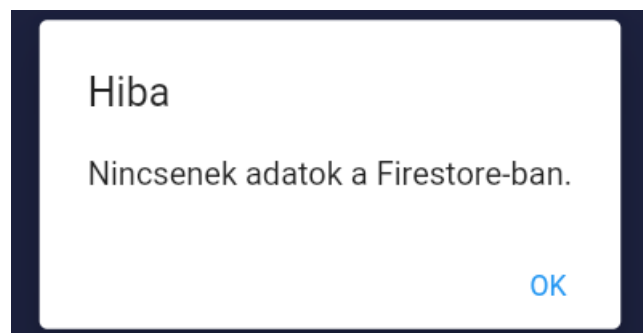
```
if (!isWeb)
  Container(
    color: Color(0xFF232B59),
    padding: EdgeInsets.all(10),
    child: Row(
      children: [
        Expanded(
          child: CardWidget(
            icon: Icons.cloud_upload,
            label: 'Feltöltés',
            onTap: () {
              viewModel?.upload(context);
            },
          ),
        ],
      ),
    ),
```

),
)

A „viewmodel” könyvtárban található viewModel osztályok felelnek a view-k és model osztályok közötti adatkommunikációért. Ezek az osztályok tartalmazzák a controllereket, (kód) azokban az osztályokban, ahol adatokat kell beállítani a view-k számára. A viewModelek segítik az adatok formázását és közvetítését, támogatva ezzel az alkalmazás modularitását és karbantarthatóságát.

```
class NewDebtViewModel {  
    final NewDebtDialogModel model = NewDebtDialogModel();  
    final TextEditingController amountController = TextEditingController();  
    final TextEditingController nameController = TextEditingController();  
    final TextEditingController debtorNameController =  
    TextEditingController();  
    bool hasRevolut = false;
```

A „widgets” mappában találhatók a gyakran használt widgetek, mint például az error_dialog. (6. ábra) Ennek használatához csupán meg kell adni a paramétereket, mint az üzenet, és a widget maga felel a megjelenítésért. Ezeket a widgeteket a view-k hívják elő, de előfordul, hogy a logikában, például a model osztályokban is szükség lehet az error_widget-re, mivel ott is szükség lehet a hibaüzenetekre. Ilyenkor a kontextust is meg kell adni a helyes működéshez.



6. ábra: Error dialog hibaüzenettel.

A "model" könyvtárban a különböző nézetekhez tartozó modellosztályok találhatók, melyek az MVVM (Model-View-ViewModel) [12] architektúrával összhangban vannak kialakítva. Ezekben az osztályokban találhatók a logikák, egy példa erre a "debt_details_model" osztályban található "calculateDebts" függvény.

Ez a függvény optimalizált tartozásokat számol ki, azaz arra törekszik, hogy a lehető legkevesebb tartozásadás mellett minden érintett személy nullára jöjjön ki. A folyamat a következő lépésekből áll: először a "loadFromDatabase" függvény összegyűjti az összes tartozást egy listába. Ezt követően egy ciklus kiveszi az összes

nevet a tartozásokból, majd egy külön listába rendeli ezeket a neveket. Ez a lista egy map, tehát kulcs-érték párokat tartalmaz, ahol a kulcs a név (egy String), az érték pedig egy int típusú szám, ami azt mutatja, hogy az adott személy mennyivel tartozik vagy mennyivel tartoznak neki. A szám pozitív vagy negatív attól függően, hogy az adott személy tartozik vagy neki tartoznak.

Amint az algoritmus kigyűjti az összes nevet (kulcsoknak), hozzárendel egy számot, amelyet a tartozásokból szed ki, ahol szerepel az adott név. Ezután hozzáadja vagy kivonja az adott személy számából. (attól függően, hogy melyik szám abszolútértéke a nagyobb) A folyamat során a legnagyobb és a legkisebb számot kiválasztja, majd ezeket összegzi vagy kivonja egymásból (attól függően, hogy melyik abszolútértéke a nagyobb). Ezzel az egyik személy száma nullává válik, és a kivonás egy tartozásadást reprezentál. Ezzel párhuzamosan létrejön egy új tartozás, ami már az optimalizált tartozások egyike. Ezt a műveletet véges sokszor megismételve (egy while ciklussal) létrejön az optimalizált tartozások listája.

```
//kiveszem az összes nevet a debtListItem-ekből
for (DebtDataModel debt in allDebts) {
    final debtorPerson = kIsWeb
        ? await getPersonByIdWeb(debt.debtorPersonId!)
        : await personDbHelper!.getPersonById(debt.debtorPersonId!);

    final personTo = kIsWeb
        ? await getPersonByIdWeb(debt.personToId!)
        : await personDbHelper!.getPersonById(debt.personToId!);

    if (debtorPerson != null) {
        debtsMap[debtorPerson.name] =
        debtsMap.containsKey(debtorPerson.name)
            ? debtsMap[debtorPerson.name]! + (debt.amount ?? 0)
            : debt.amount ?? 0;
    }
}
```

Az itt fent látható kódrészlet a `debt_details_model.dart` fájlban található, a `calculateDebts` függvényben és annak az elejét mutatja be. A kód kigyűjti az összes nevet a tartozások listaelemekből és beleteszi a `debtsMap`-ba ügyelve arra, hogy minden név, ami legalább egyszer szerepelt a tartozások között, az pontosan egyszer szerepeljen a `debtsMap`-ban.

A kód második része arra szolgál, hogy frissítse a `debtsMap`-ben a személyekhez tartozó számokat. Amennyiben az adott név mellett már szerepel egy összeg a map-ban, a kód hozzáadja ehhez az összeghez a tartozás listában lévő összeget, (amennyiben még nem szerepel, akkor ezt az összeget egyenlővé teszi a személyhez tartozó számmal)

feltéve, hogy a személy az, aki tartozik. Ha a személy az, akinek tartoznak, akkor kivonja a név melletti számból a tartozás listában található összeget.

Ez a megközelítés biztosítja, hogy minden név mellett egy szám lesz, amely az adott személy aktuális pénzügyi helyzetét mutatja. Ha a szám negatív, az azt jelenti, hogy a személynek még tartozik valaki pénzzel (az ő zsebéből hiányzik pénz), míg a pozitív érték azt mutatja, hogy több pénz van a zsebében, mint amennyinek kéne lennie, tehát ő tartozik másnak vagy másoknak pénzzel és nála most több van, még hozzá annyival, amennyi a map-ban a neve mellett álló szám.

```
//inicializálom a változókat
String debtorWithSmallestDebt = '';
String debtorWithLargestDebt = '';
int smallestDebt = 0;
int largestDebt = 0;
List<DebtDataModel> resultDebts = [];

debtsMap.forEach((debtor, amount) {
    if (amount > largestDebt) {
        largestDebt = amount;
        debtorWithLargestDebt = debtor;
    }
    if (amount < smallestDebt) {
        smallestDebt = amount;
        debtorWithSmallestDebt = debtor;
    }
});
```

Az itt fent látható kódrészlet először inicializál két nevet: az egyik a személyé, akinek a legnagyobb a tartozása, a másik pedig azé, akihez a legtöbb pénzzel tartoznak. Emellett három további változó inicializálódik: az egyik az összeg, ami a legnagyobb tartozáshoz kapcsolódik, a másik pedig az, akihez a legtöbb pénzzel tartoznak, annak a személynek a hitel összege.

Ezután egy lista inicializálódik, amely az eredményeket, azaz az optimalizált tartozásokat fogja tartalmazni. A következő lépés egy kereső algoritmus, amely megtalálja a legnagyobb és legkisebb tartozást, valamint ezekhez tartozó neveket, vagyis az inicializált változókat feltölti.

A következő kódrészlet bemutatja az algoritmus esszenciáját egy while ciklusban. Ez a ciklus addig fut, amíg a debtsMap nevekhez tartozó szám értékek mindegyike nem lesz nulla. Ennek eléréséhez a fentebb említett kódrészletet végrehajtja, inicializálja és feltölti a változókat a megfelelő értékekkel. Majd a legnagyobb tartozáshoz rendeli a személyt, akinek a legnagyobb összeggel tartoznak, (Tehát a legnagyobb számhoz, ami

pozitív, rendeli a legkisebb számot, ami nyilván negatív) és ezeket a párokat összerendeli. Ez látható itt:

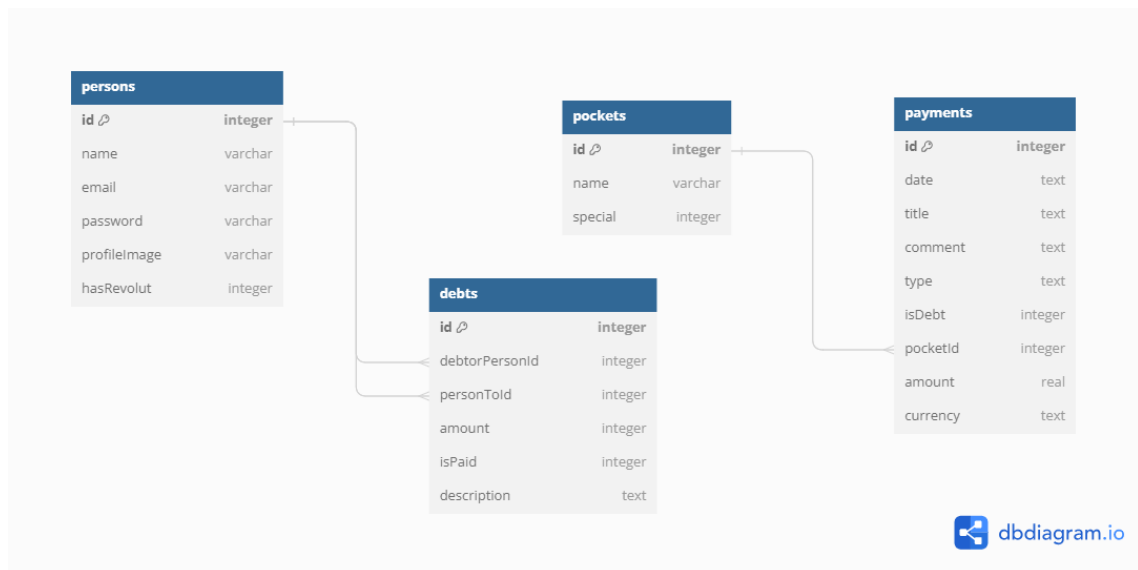
```
while (debtsMap.values.any((value) => value != 0)) {  
  ...  
  if (smallestDebt.abs() > largestDebt.abs()) {  
    debtsMap[debtorWithSmallestDebt] =  
      (debtsMap[debtorWithSmallestDebt] ?? 0) + largestDebt;  
  
    debtsMap[debtorWithLargestDebt] =  
      (debtsMap[debtorWithLargestDebt] ?? 0) - largestDebt;  
  
    final debtorPerson = kIsWeb  
      ? await getPersonByNameWeb(debtorWithLargestDebt)  
      : await  
personDbHelper!.getPersonByName(debtorWithLargestDebt);  
    final personTo = kIsWeb  
      ? await getPersonByNameWeb(debtorWithSmallestDebt)  
      : await  
personDbHelper!.getPersonByName(debtorWithSmallestDebt);  
  
    if (debtorPerson != null && personTo != null) {  
      resultDebts.add(DebtDataModel(  
        debtorPersonId: debtorPerson.id,  
        personToId: personTo.id,  
        amount: largestDebt.abs(),  
      ));  
    }  
  } else {...
```

Ebben a ciklusban, ha a legnagyobb tartozó személy száma nagyobb, mint annak a személynek a számának az abszolútértéke, akihez a legtöbb pénzzel tartoznak, akkor a legnagyobb tartozásból kivonják ezt az abszolút értéket (vagy más szóval a nagyobb számhoz hozzáadjuk magát a negatív értéket). Ezt követően a személy, aki a legtöbb pénzzel tartozik, visszafizetett egy tartozást egy másik személynek. Ez a folyamat egy új tartozással kerül reprezentálásra, amit a resultDebts listába helyezünk el. Ilyenkor még a kódban annak a személynek számát nullára állítjuk, akinek tartoznak, mert neki már megadták a tartozást.

Amennyiben a negatív szám abszolút értéke nagyobb, akkor ehhez a negatív számhoz adjuk hozzá a legnagyobb pozitív számot, amely szintén egy tartozást reprezentál. Ebben az esetben is létrejön egy új debtDataModel, amit a resultDebts listába helyezünk.

Minden ilyen ciklus végén legalább egy személy száma a map-ban nullára lesz állítva. Ezt véges sokszor elvégezve minden személy esetén megtörténik, tehát a ciklus blokkból kilépünk, az eredménylista feltöltődik a megfelelő tartozásokkal, és a függvény ezt adja vissza, amit lehet kiírni a képernyőre.

4.1.2 Adatbáziskezelés



7. ábra relációs adatbázis diagram

A fenti diagramon (7. ábrán) szemléltetem az alkalmazásom adatbázis tábláit, melyeket a következőképpen fogok elemezni. Elsőként megfigyelhető a "személyek" tábla, amely tartalmaz egy ID-t, ami az elsődleges kulcs, továbbá a személy nevét, e-mail címét, jelszavát, profilképét, valamint egy változót, amely jelzi, hogy a személynek van-e Revolut fiókja. Fontos megjegyezni, hogy az alkalmazásban csak a név kötelező megadása, míg a többi mező teljesen tetszőlegesen kitölthető.

Emellett, a "tartozások" táblában szintén található egy ID, ami az elsődleges kulcs, egy tartozó ID (debtorPersonId), amely azonosítja azt a személyt, akinek már létező és akire a tartozás vonatkozik. Ez egy idegen kulcs, kapcsolódva a "személyek" tábla ID-jéhez. Ezenkívül van egy másik ID (personToId), ami szintén egy külső kulcs, és ez az a személy, akinek tartoznak. Ez is egy olyan mező, amely egy már létező személy ID-jét tartalmazza. A táblában található egy összeg mező, ami a tartozás összegéről ad információt, egy "kifizetve" mező, ami azt jelzi, hogy a tartozás kifizetésre került-e, és egy leírás mező, ahol rövid leírást lehet adni a tartozásról.

A diagramon még további két fontos tábla látható, nevezetesen a "zsebek" és "fizetések" táblák. A "zsebek" táblában található egy egyedi azonosító és egy kötelezően kitöltendő név mező, ami a zseb nevét jelöli. Ezenkívül található egy speciális változó, amely, ha igaz, akkor az összes tranzakciót (bevételt vagy kiadást)

tartalmazza és mutatja majd, ellenkező esetben csak az adott zsebhez tartozó tranzakciók kerülnek bele.

A "fizetések" táblában a mezők az ID, dátum, cím, megjegyzés, típus (bevétel vagy kiadás), tartozás állapot, zsebhez tartozó ID, összeg és valuta.

A "zsebek" és "fizetések" táblák közötti kapcsolatot a "zseb ID" kulcs teremti meg, mely a "fizetések" táblában egy külső kulcsként szerepel. Ez a kulcs segít az azonos zsebbe tartozó tranzakciók azonosításában.

A Firebase Cloud Firestore Database [2] és a hagyományos relációs adatbázisok közötti kulcskülönbségek kiemelkedő fontosságúak és ezekre nekem is figyelniem kellett, amikor a Firebase szerver database szolgáltatásait használtam.

```
List<PersonDataModel> peopleList = [];  
final peopleQuerySnapshot = await peopleCollectionRef.get();  
if (peopleQuerySnapshot.docs.isNotEmpty) {  
  peopleList = await Future.wait(peopleQuerySnapshot.docs.map(  
    (doc) async => PersonDataModel.fromMap(doc.data()),  
  ));  
  return peopleList;  
}
```

Ez a kód a `person_repository`-ből van, ami egy példa arra, hogy hogyan figyelek a NoSQL adatbázisból a lokális relációs adatbázis által ismert adatmodellekre való áttérésre. (jelen esetben egy személy, tehát egy `PersonDataModel`-t használok)

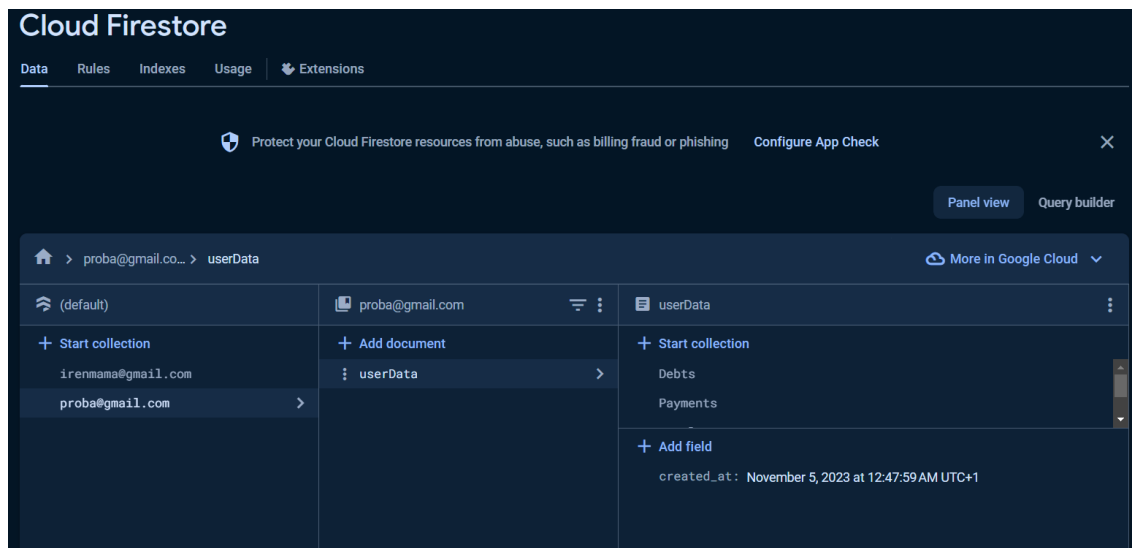
A kód a Firestore adatbázisból lekér egy gyűjteményt, amely a "people" nevet viseli. (a kódban először egy üres lista, a `peopleList` inicializálódik, amibe a személyek kerülnek) Ezután aszinkron módon megtörténik a Firestore-ból a "people" gyűjtemény lekérése a `peopleCollectionRef` referencia segítségével.

Az összes dokumentumot lekérdezi és átalakítja azokat `PersonDataModel` objektumokká a **`PersonDataModel.fromMap`** metódus segítségével. Ezek az objektumok a `peopleList` listába kerülnek.

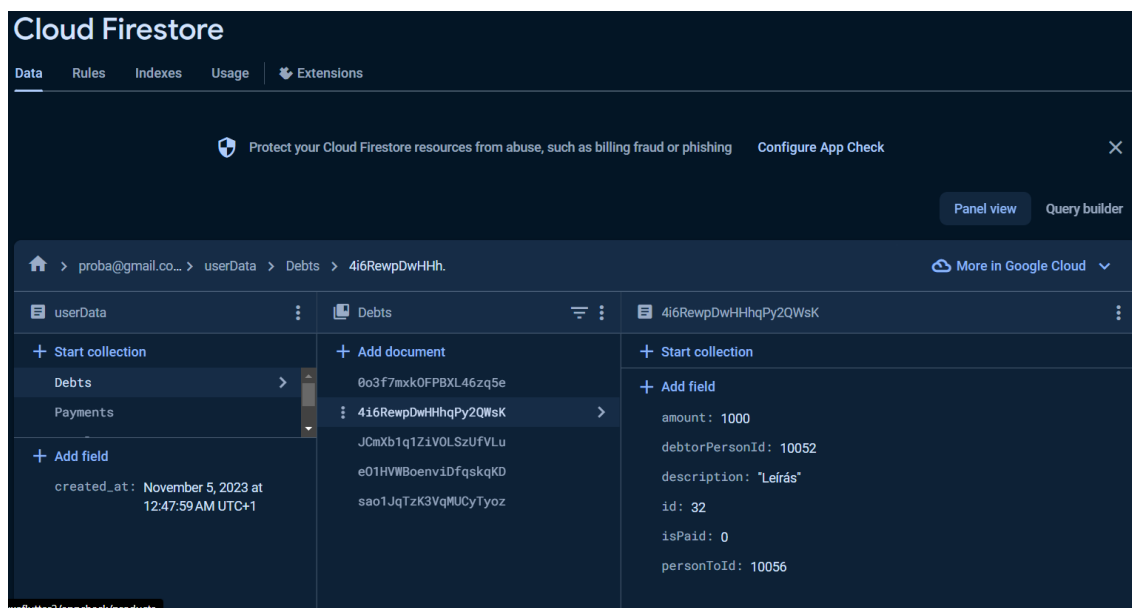
A Firestore NoSQL [4] adatmodellje rugalmasságot biztosít, dokumentumokba szervezni az adatokat, és JSON-struktúrát alkalmaz. Ezzel szemben a relációs adatbázisok táblákat használnak előre definiált sémákkal és szorosan szabályozott relációs kapcsolatokkal.

A konzisztencia terén a Firestore [2] inkább a gyenge konzisztenciára összpontosít, míg a relációs adatbázisok erős tranzakciókat és ACID [13] tulajdonságokat kínálnak.

A Firestore [2] könnyen skálázható és gyors, kiválóan alkalmas modern alkalmazásokhoz, miközben a relációs adatbázisok komplexebb skálázhatóságot és hosszabb beállítási időt igényelnek.



8. ábra Firestore adatstruktúra első ábra



9. ábra Firestore adatstruktúra második ábra

A Firestore adatbázisom struktúrájában a felhasználók e-mail címei alapján elnevezett kollekciók találhatók a hierarchia legfelső szintjén. Az alkalmazásban egyedi e-mail címek használatát követeltem meg a regisztráció során, így ezek a felhasználók kollekcióinak nevei lettek, amelyek segítik, hogy mindig az aktuális bejelentkezett felhasználó csak a saját kollekcióját tudja elérni. (a kódban csak arra kell figyelni, hogy a felhasználó be legyen jelentkezve mielőtt a Firestore-t használná és hogy csak a

bejelentkezett e-mail című kollekció nyíljon meg) Minden ilyen kollekcióban megtalálható egy "userData" nevű dokumentum, amely tartalmaz további kollekciókat: "fizetések", "tartozások", "emberek" és "zsebek" néven. Ezekben a kollekciókban különböző dokumentumok találhatók, amelyek már konkrét listaelemeket tartalmaznak. (Ez látható a második képen)

4.2 A projekt megvalósításának folyamata

A projekt kezdeti szakaszában, a Kotlin nyelvű natív Android alkalmazáson dolgozva, rálátást nyertem az alkalmazás megvalósítására. A szakdolgozat során megismerkedtem a Flutterrel és a Dart-tal, készítettem egy rövid próbaprojektet a widget-ek működésének tesztelésére és az objektumorientált programozás gyakorlására.

Az éles projektben elkészítettem a belépő felületet, de a Firebase regisztráció terén nehézségekbe ütköztem, amelyeket nehezen sikerült megoldani, bár sok forrást és dokumentációt tanulmányoztam. A problémákat követően azonban folytattam az alkalmazás fejlesztését, és megkezdtem az MVVM architektúra alapján történő kialakítását. Az osztályhierarchiát, a mappastruktúrát és a kódot rendeztem, bár inicializálás során néhány nehézségbe ütköztem.

A projekt kezdeti szakaszában lokális adatbázist használtam és az Android emulátoron teszteltem. A következő nehézség a webes felület kialakításával és optimalizálásával jött, amely során problémákat tapasztaltam a Firebase konzolos felületén. Ezt követően újból haladtam, megoldva a Firebase projekt és az alkalmazás regisztrációjával kapcsolatos kihívásokat.

4.2.1 verziókezelés

A projekt verziókezeléséhez és szinkronizálásához a GitHub kód megosztó felületét alkalmaztam. Gyakran commit-oltam és push-oltam külön branch-eken az egyes részeket és modulokat. Ezeket időnként, általában két-három hetente merge-eltem a main branch-re, így könnyedén visszanézhettem korábbi kódokat, amikor valamit úgy változtattam meg, hogy az már nem működött, ugyanakkor tudtam, hogy korábban viszont működött.

A GitHub [6] szinkronizációjához a GitHub Desktop [15] alkalmazást választottam, ahol egyszerűen kezelhettem a kódom és még a szakdolgozatom Microsoft Word-ben lévő irományom szinkronizálását is. Az alkalmazás írásának

kezdeti szakaszaiban gyakran kellett módosítani az adatbázis sémát, ekkor a migráció kihívásával szembesültem. Ezt mindig megoldottam a teljes alkalmazás törlésével és újratelepítésével, de ennek hátránya volt, hogy az előzőleg bevitt tesztadatokat újra létre kellett hozni.

4.3 A program tesztelése

Az alkalmazást az Android Studio Emulátor [9] segítségével teszteltem, amelyen egy Google Pixel 5-ös eszközt emuláltam, mely Android 11.0 operációs rendszert futtat. A webes részét az alkalmazásnak a Visual Studio Code [8] beépített webszerverén localhost-on teszteltem. Emellett saját fizikai telefonomon, egy Xiaomi Mi 9 SE-n is kipróbáltam az alkalmazást, ahol már az Android 12.5.1 verzió fut.



10. ábra: Emulátor.

4.3.1 Manuális tesztelés

A programot manuálisan teszteltem, amelynek minden lépése megtalálható a program gyökér mappájában. Ezekből a tesztek közül itt a dolgozatomban bemutatnék egyet.

A maximális felhasználói élmény érdekében elengedhetetlen a manuális tesztelés a felhasználók területén. A program valós idejű tesztelése fontos lépés a tervezett funkciók és viselkedés ellenőrzésében, hogy biztosítsuk a specifikációk szerinti működést. A manuális tesztelés során a tesztelő interaktívan használja az appot, kipróbálja a különböző menüpontokat, funkciókat és forgatókönyveket. Ez lehetővé

teszi a valós problémák, hibák és hiányosságok azonosítását, amelyek befolyásolhatják a felhasználói élményt. A manuális tesztelés által nyújtott visszajelzés segít a fejlesztőknek az app minőségének javításában és finomhangolásában. Ezáltal a manuális tesztelés létfontosságú mindenféle program fejlesztésében, hogy az app valóban úgy működjön, ahogy tervezték, és a felhasználóknak optimális élményt nyújtson.

4.3.2 Tesztelt funkciók

- felhasználó létrehozása, autentikáció (mobilon, vagy weben)
- tartozások létrehozása és szétosztása (mobilon, vagy weben)
- zsebek létrehozása (mobilon, vagy weben)
- fizetések létrehozása (mobilon, vagy weben)
- szerverre való feltöltés és letöltés (mobilon)
- lekérdezés végrehajtása (mobilon, vagy weben)

4.3.3 Teszt felépítése

Minden teszt az alábbi pontokból épül fel: Van egy leírás, ahol leírom, hogy mit fogok tesztelni és milyen lépéseket fogok tenni, valamint pár elvárást is megfogalmazok. Azután "Lépések" cím alatt megfogalmazom röviden pontokba szedve az összes lépést, amit a teszt során elvégzek.

Ezek után következik egy "Elvárt eredmények" pont, ahol összehasonlítom az elvárt eredményeket a valós eredményekkel többnyire képek formájában.

Mivel a tesztelést androidos operációsrendszeren, telefonon és weben is elvégzem ezért a képek lehetnek a webes alkalmazásból, de a telefonos felületről is.

Az utolsó két pont minden teszt esetében a "Siker/Hiba" és a "Megjegyzések" pont. Az előbbinél leírom az eredményekből elkönyvelt sikereket és az esetleges hibákat is pontokba szedve. A megjegyzéseknél meg ha van valami fontos, amit hozzá kéne fűzni az adott teszthez azt leírom.

4.3.4 Példa teszt

4.3.4.1 Leírás

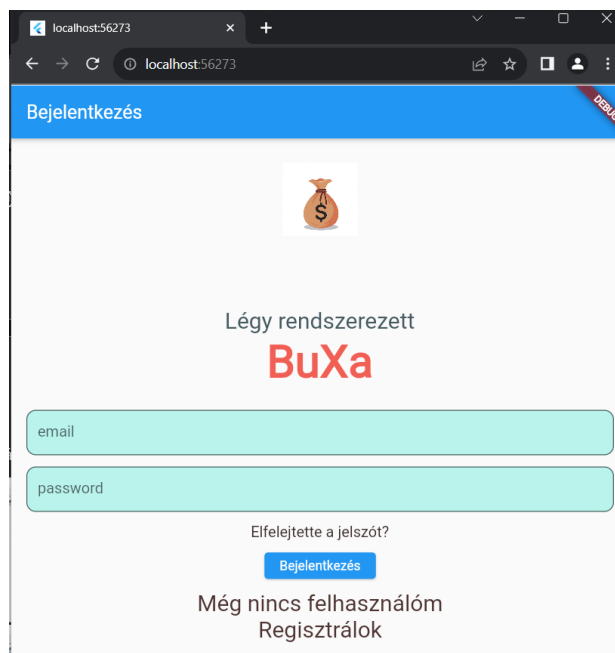
Az alkalmazás indításakor létrehozunk egy új felhasználót a „Még nincs felhasználóm” gomb megnyomásával. Ebben az ablakban egyedi e-mail címet és jelszót adunk meg. Sikeres regisztráció esetén az alkalmazás a bejelentkező képernyőre navigál minket.

A bejelentkező képernyőn először megpróbáljuk a bejelentkezést egy helytelen jelszóval, majd egy helyes jelszóval. Az alkalmazástól elvárjuk, hogy megfelelő visszajelzéseket adjon az eseményekről. Például, ha a rossz jelszóval történő bejelentkezés nem sikerül, az alkalmazás érthető módon jelezze ezt a felhasználónak. Amennyiben a helyes jelszóval sikeres a bejelentkezés, az alkalmazás navigáljon minket a menü képernyőre.

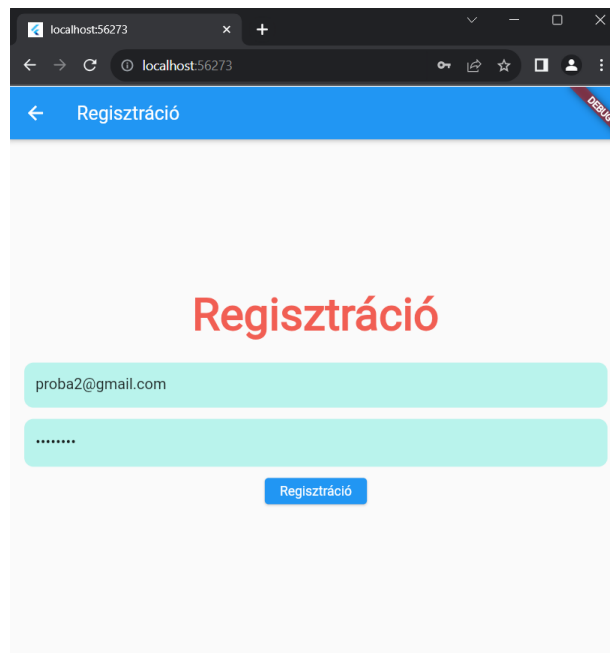
4.3.4.2 Lépések

1. program elindítása emulátoron, vagy telefonon, vagy weben
2. „Még nincs felhasználóm gomb lenyomása
3. egyedi email cím és jelszó megadása
4. Rossz jelszó megadása a helyes email cím mellé
5. Jó jelszó megadása a helyes emailcím mellé
6. Figyelni a nem logikus és rendellenes viselkedést

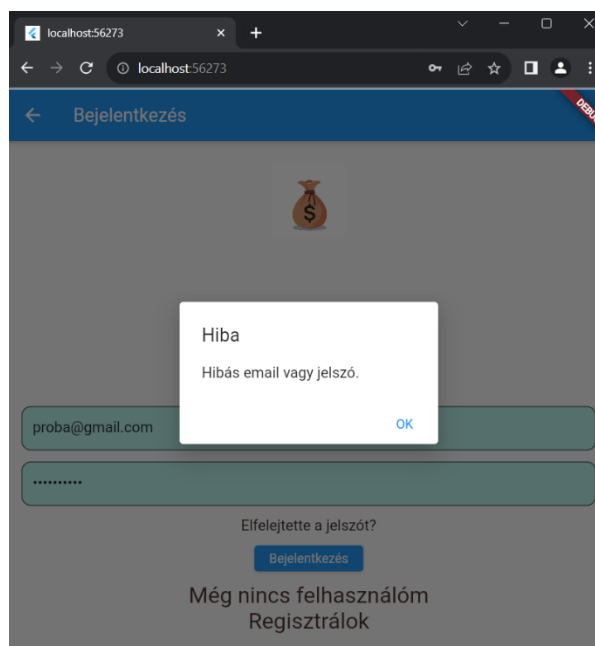
4.3.4.3 Elvárt eredmények



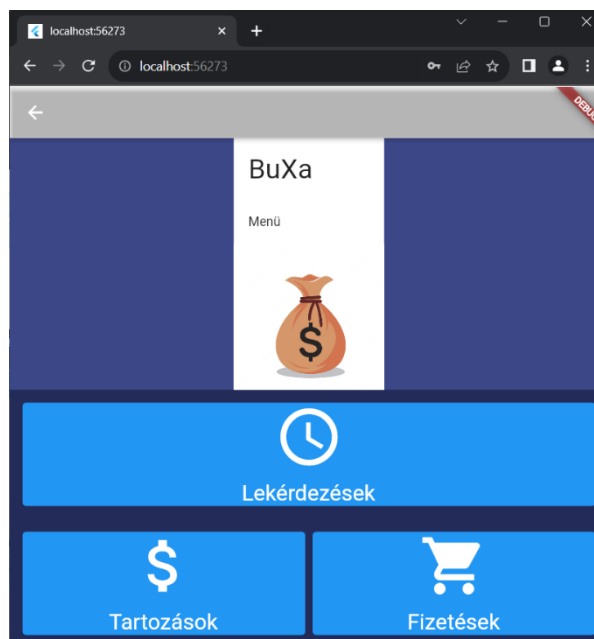
11. ábra: Teszt első elvárt eredménye.



12. ábra: Teszt második elvárt eredménye.



13. ábra: Teszt harmadik elvárt eredménye.



14. ábra: Teszt negyedik elvárt eredménye.

4.3.4.4 Siker/Hiba

- A felugró ablak, ami a hibaüzenetet jelzi, nem ad pontos információt a hibáról (pl, hogy az e-mail, vagy a jelszó a rossz)
- Ezen kívül a teszt sikeresen elvégezhető
- A regisztrációs ablaknál, ha olyan e-mailt adunk meg, ami már szerepel a szerveren, akkor is hiba ablak ugrik fel, de ez sem ad pontos információkat

4.3.4.5 Megjegyzések

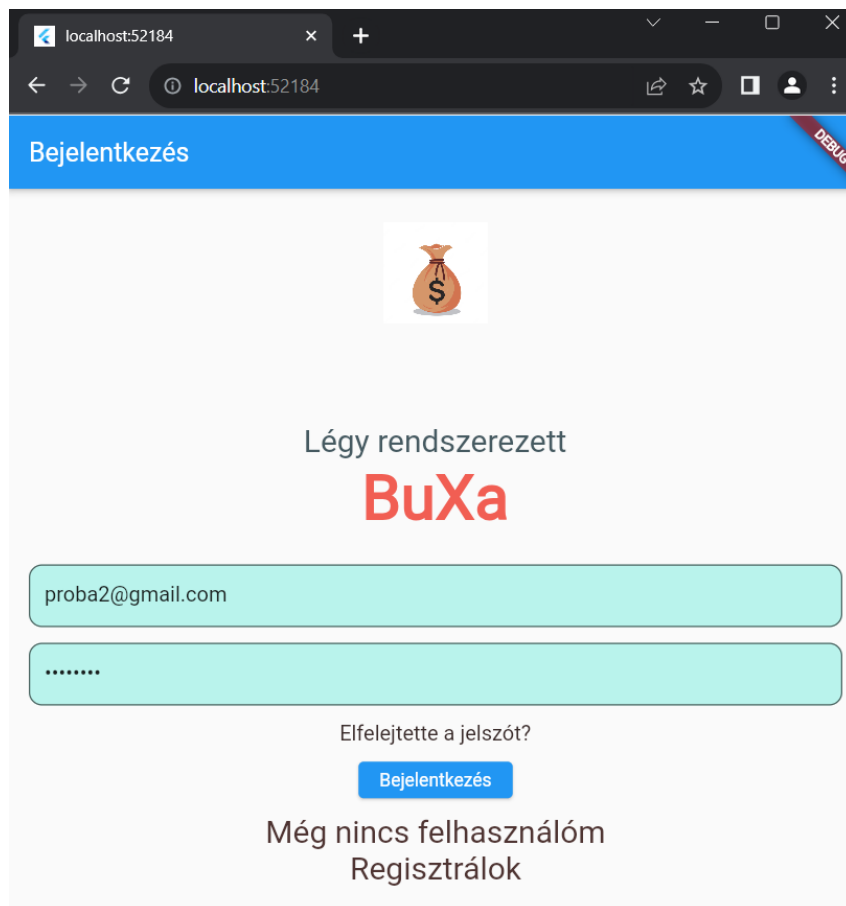
- androidos telefonon tesztelve szintén ugyanezekre az eredményekre jutunk

4.4 A program használata

Az alkalmazás általános funkcionalitása kiemelkedő mértékű sokoldalúságot kínál, ugyanis mind webes, mind Android operációs rendszeren való használatra optimalizált. Android platformon történő alkalmazásindítás után egy intuitív bejelentkezési felületre érkezünk, ahol a már regisztrált felhasználók egyszerűen adhatják meg az email címüket és jelszavukat a rendszerbeli belépéshez. Amennyiben a felhasználónak még nincs regisztrált fiókja, könnyedén navigálhat, a regisztráció gomb segítségével a regisztrációs oldalra, ahol egyedi email címet és jelszót adhat meg a fiók létrehozásához. Az email cím egyedisége kardinális fontosságú, mivel az adatazonosítási szempontból elengedhetetlen.

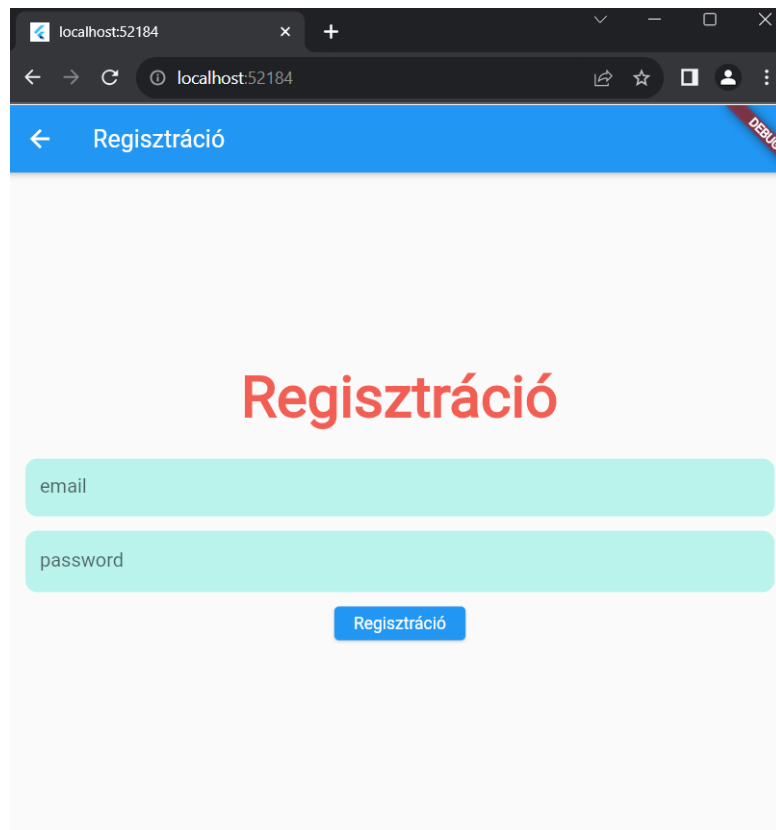
A regisztrációs folyamat során előforduló esetleges hibákat egy értesítő ablak konglomerátumával szemlélteti a rendszer, mely szükséges visszajelzéseket nyújt a felhasználónak. A sikeres regisztrációt követően ismét a bejelentkezési oldal tárul elénk, ahol a felhasználó az előzőleg megadott adatokkal beléphet saját fiókjába. Az alkalmazás a felhasználó internetkapcsolatát ellenőrzi, majd a sikeres belépést követően a menüoldal megjelenik.

Az Android platformon található menüoldal rendkívül áttekinthető és felhasználóbarát, öt különböző gombbal rendelkezik, melyek a lekérdezéseket, tartozásokat, költséket/bevételeket, a szerverre való adatfeltöltést és a szerverről való adatletöltést szolgálják. A webes platformon történő alkalmazásnyitás az utóbbi két gomb, tehát a szerverre való feltöltés és szerverről való letöltés gombok nem jelennek meg, azért, mert a webes felület automatikusan kommunikál a szerverrel, így a lekérdezéseket és adatfeltöltéseket könnyedén kezelhetjük és nem kell figyelniük a szerverrel való szinkronizálásra.

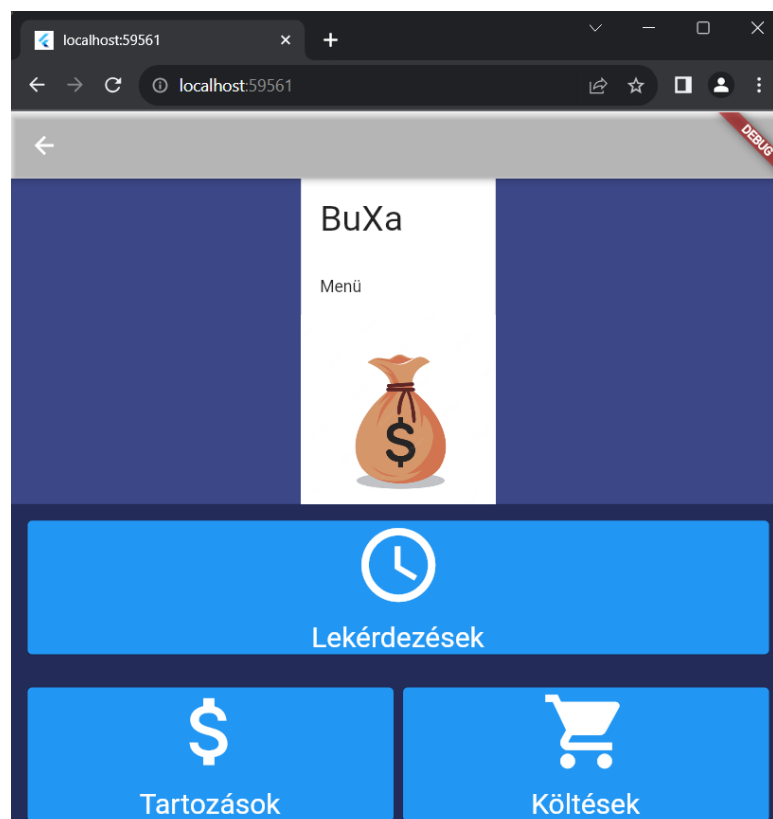


The screenshot shows a web browser window with the address bar displaying 'localhost:52184'. The page has a blue header with the text 'Bejelentkezés' (Login) and a red 'DEBUG' banner in the top right corner. Below the header, there is a money bag icon with a dollar sign. The main text reads 'Légy rendszerezett' (Be organized) followed by the 'BuXa' logo in red. There are two light blue input fields: the first contains the email 'proba2@gmail.com' and the second contains masked characters '.....'. Below the input fields, there is a link 'Elfelejtette a jelszót?' (Forgot your password?) and a blue button labeled 'Bejelentkezés' (Login). At the bottom, the text 'Még nincs felhasználóm' (I don't have an account yet) and 'Regisztrálok' (I register) is displayed.

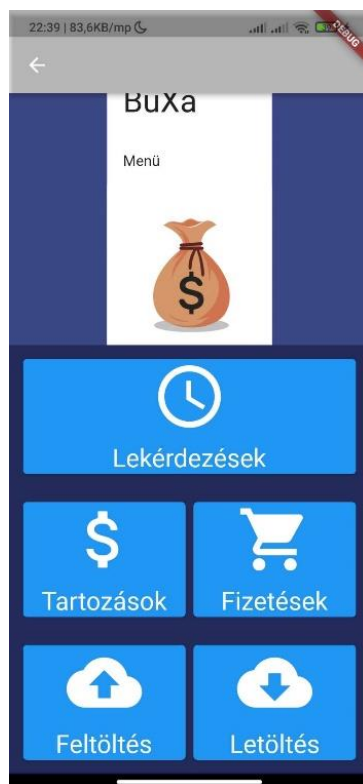
15. ábra: Login képernyő.



16. ábra: Regisztrációs képernyő.



17. ábra: Menü képernyő weben.



18. ábra: Menü képernyő mobilon.

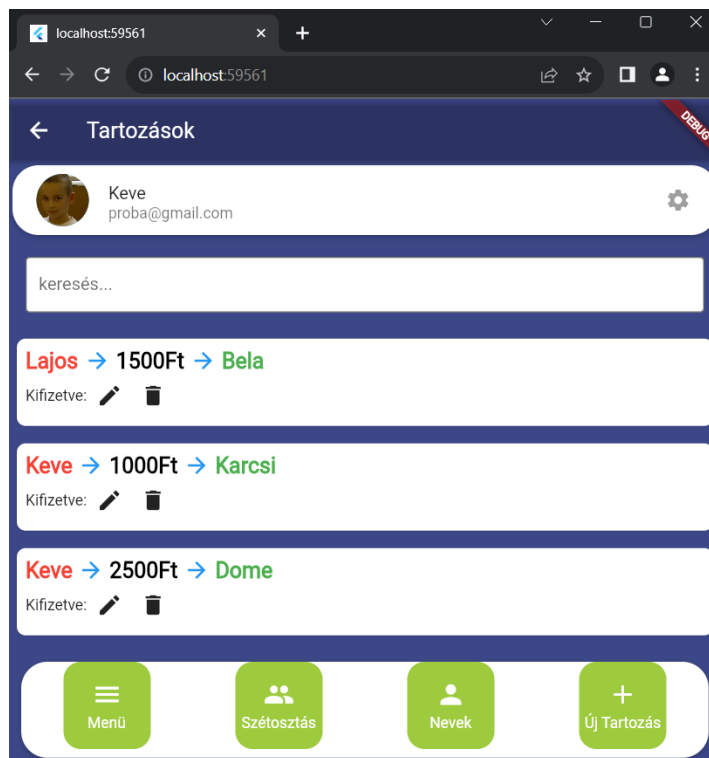
Az alkalmazás tartozások menüpontjára kattintva egy speciális képernyő jelenik meg, ahol a profilunk adatai, beleértve a bejelentkezési email címünket és a névünket, felül láthatók. Az aktuális tartozásaink egy görgethető ablakban jelennek meg, míg az alsó részen egy Deák widget található. Ezen a widgeten különböző gombok találhatók.

Az első gomb a "Vissza" lehetőséget kínálja, amely segít visszatérni a fő menübe. A második "Szétosztás" gomb a meglévő tartozásokat úgy osztja el, hogy minimális tartozás megadásával minden résztvevő megkapja azt az összeget, amely jár neki. Ezáltal mindenki nullára áll majd és csak a saját költségeiért fizetett.

A következő gomb a "Nevek" lehetőség, amelyen keresztül új személyeket lehet felvenni a rendszerbe. Ezeket a neveket később könnyedén kiválaszthatjuk, amikor új tartozást hozunk létre. Az utolsó gomb a "Tartozás hozzáadása", amely lehetővé teszi egy új tartozás rögzítését. Itt kiválaszthatjuk a személy nevét, megadhatjuk a tartozás összegét, adhatunk hozzá egy rövid leírást, majd a "Hozzáadás" gombbal rögzíthetjük az adatokat.

Amennyiben mégis meggondolnánk magunkat, a "Mégsem" gomb segítségével törölhetjük az új tartozást. Az alkalmazás így rendkívül rugalmasan kezeli a tartozásokat, és a kényelmes használat érdekében lehetőséget biztosít a tartozások

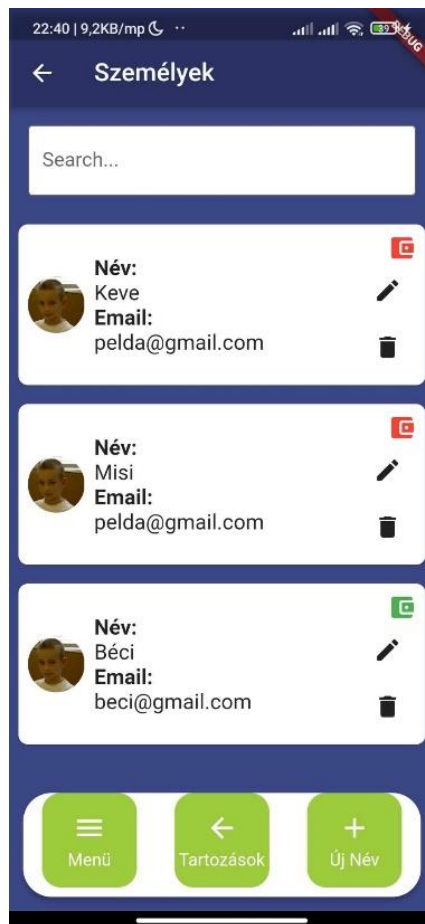
könnyű kezelésére és nyomon követésére. A képernyő közepén megjelent tartozás listaelemeken lévő kuka gomb lenyomásával törölni tudjuk az aktuális listaelemet és az alkalmazás automatikusan frissül és törli az elemet.



19. ábra: Tartozások képernyő.

A "Nevek" menüpontra kattintva szintén egy görgethető lista jelenik meg, ahol láthatjuk a személyek listáját. Új személyt egyszerűen felvihetünk a "Új név" gombra kattintva. Ekkor megadhatjuk a személy nevét, email címét és hogy van-e Revolut bankszámlája, ami azért hasznos, mivel ennek az alkalmazásnak a segítségével díjmentesen és azonnal lehet pénzt küldeni bármilyen másik Revolut számlára, ezzel egyszerűvé téve a tartozások megadását. Az új személy felvitelénél a név mező kötelező és egyedinek kell lennie. Az új személyt könnyedén hozzáadhatjuk a listához, majd a kuka gomb segítségével törölhetjük az adott listaelemet.

Ha egy új tartozás felvételénél olyan személyt írunk be, aki még nincs a listán, az alkalmazás automatikusan létrehoz egy új személyt ezen a néven. Azonban fontos megjegyezni, hogy ha töröljük egy személyt a "Nevek" menüben, akkor azhoz tartozó tartozások érvénytelenekké válnak, így gondoskodnunk kell arról, hogy az adott személyhez kapcsolódó tartozások rendelkezésre álljanak.

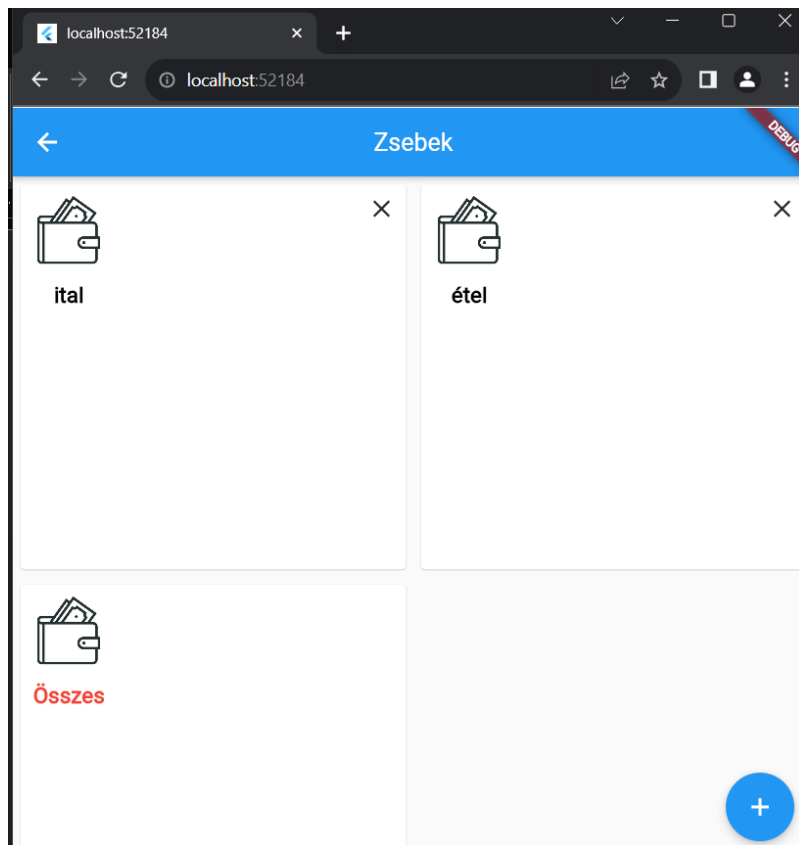


20. ábra: Személyek képernyő mobilon.

A "Költségek" fül alatt, a menüben a "Költségek" gombra kattintva, a Zsebek képernyőn találjuk magunkat. Itt lehetőségünk van új zsebek létrehozására és meglévő zsebek megtekintésére. Két különböző típusú zseb létezik: az egyik pirossal szedett névvel rendelkezik, és ezt a zsebet nem lehet törölni. Ebben a zsebben mindenféle költség és bevétel megjelenik, míg a többi zsebbe csak akkor kerülnek bele a költségek és bevételek, ha létrehozás vagy frissítés során megfelelően beállítjuk a kapcsolódó zsebet.

A pirossal szedett zsebet nem lehet törölni, és ennek érdekében a zseb jobb felső sarkában nem elérhető a kis "x" ikon. A többi zseb esetében az „x” ikonra kattintva megjelenik egy megerősítő képernyő, amely biztosítja, hogy véletlenül ne törölhessük le a zsebet. Ez a felhasználóbarát megközelítés segíti a biztonságos zsebbekezelést. A jobb alsó sarokban található kis plusz ikon segítségével könnyedén létrehozhatunk új zsebeket, csak meg kell adnunk a nevüket, majd rá kell nyomnunk a "Mentés" gombra.

Az adott zsebre rákattintva megjelennek a benne található költség és betvétel listaelemek.



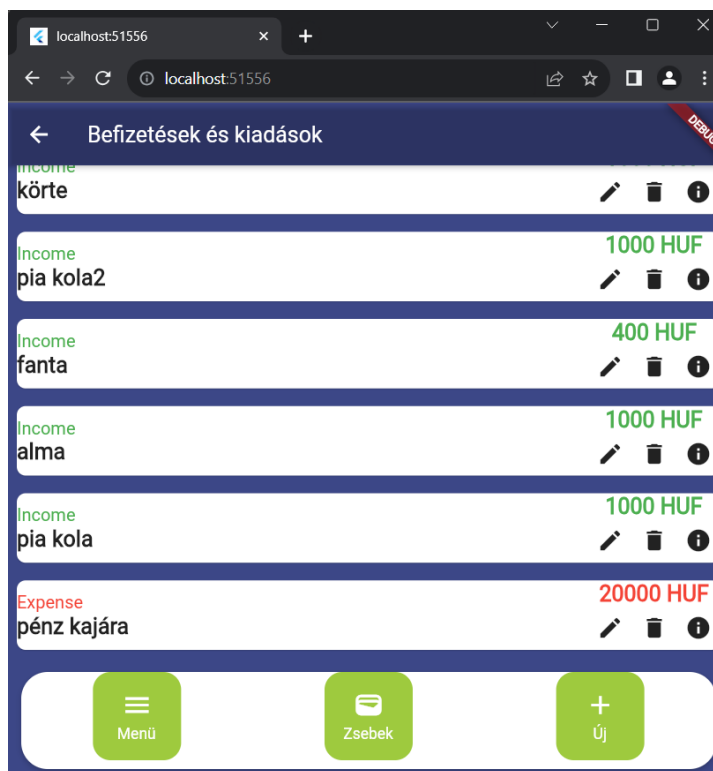
21. ábra: Zsebek képernyő weben.

Az alkalmazásban a "Zsebek" menüpont alatt, egy adott zsebre kattintva előtűnik egy új nézet, melyet "Befizetések és kiadások" néven találhatók. Ezen a nézeten egy görgethető elem található, ahol láthatók a különböző bevételi és kiadási tételek. Alatta elhelyezkedik egy Desk widget, ahol a képernyőre szabott gombok jelennek meg.

Az első gomb a "Menü", ami visszavezet a fő menübe, a második pedig a "Zsebek" gomb, amely visszavezet a zsebek nézetére. Itt található még egy "Új" gomb is, amely segítségével új listaelemet hozhatunk létre. Ebben a nézetben megadhatjuk a bevétel vagy kiadás címét, hozzáfűzhetünk emellett egy megjegyzést, és egy kis lenyíló ablakban kiválaszthatjuk, hogy bevétel vagy kiadás, valamint bejelölhetjük, hogy tartozás-e az adott tétel.

A következő lépés a zseb kiválasztása, ahol választhatunk a már létező zsebek közül vagy kézzel beírhatjuk a zseb nevét. Ezt követően meg kell adnunk az összeget, a devizát (euró, font, dollár stb.), és a dátumot, amikor a tranzakció történt. Az intuitív date-picker felület lehetővé teszi a dátum könnyű kiválasztását, és ha nem választjuk ki, automatikusan a jelenlegi dátumra állítódik be.

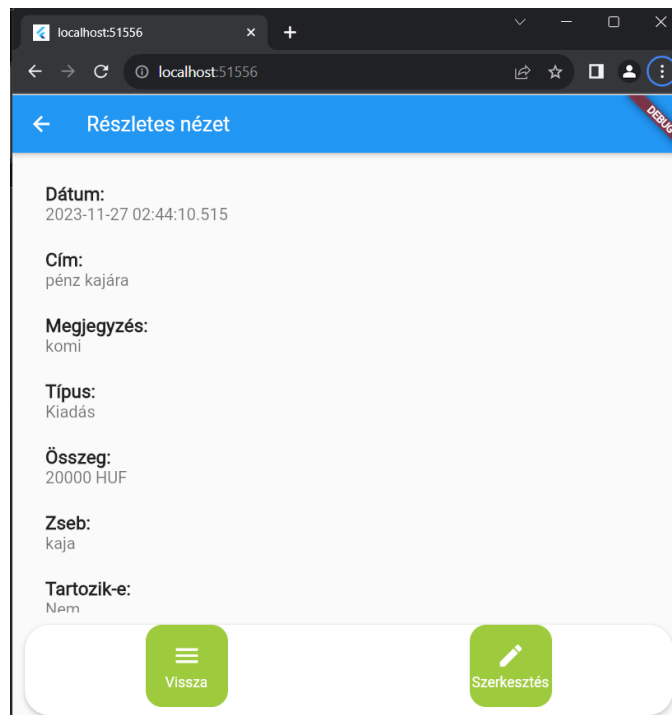
Miután minden adatot megadtunk, a "Mentés" gomb lenyomása után az új listaelem megjelenik zölddel, ha bevétel, és pirossal, ha kiadás. A lista elemeken három gomb található: az első segítségével törölhetjük az aktuális elemet, a kis "i" betűre kattintva pedig részletesebb információkat kaphatunk az adott tranzakcióról.



22. ábra: Bevételek, kiadások oldal.

A "Kis i" betűre kattintva előtűnik az adott tranzakció vagy listaelem részletes nézete. Ebben a görgethető nézetben átfogóan megtaláljuk a tranzakció összes tulajdonságát, beleértve azokat, amelyeket megadtunk neki, és azokat, amelyek alapján rendezhető vagy kategorizálható. A részletes nézeten láthatóak a dátum, cím, megjegyzés, típus (amely megmutatja, hogy kiadás vagy bevétel volt), az összeg, a zseb, amiben a tranzakció szerepel, a tartozás információja, és természetesen a deviza is.

Az alsó Desk widget-en két személyre szabott gomb található. Az egyik a "Vissza" gomb, amely lehetővé teszi az egyszerű visszatérést az előző nézethez, míg a másik a "Szerkesztés" gomb, amely segítségével bármilyen változtatást elvégezhetünk az adott listaelemen vagy tranzakción. Ezen gombra kattintva módosíthatjuk a tranzakció adatait, frissíthetjük a megjegyzéseket, vagy akár a típusát is megváltoztathatjuk. Ez az intuitív szerkesztési lehetőség fokozza a felhasználói kényelmet és rugalmasságot a pénzügyek kezelésében.



23. ábra: Részletes nézet weben.

A részletes nézetben, a "Szerkesztés" gombra kattintva lehetőség nyílik az adott listaelem szerkesztésére. Ezen a felületen valamennyi tulajdonságát módosíthatjuk, majd a "Hozzáadás" gomb segítségével frissíthetjük az adott tranzakciót vagy listaelemet. Ezáltal könnyedén és rugalmasan kezelhetjük és aktualizálhatjuk a pénzügyi tranzakciók adatait a saját igényeink szerint.

localhost:52184

← → ↻ localhost:52184

← Fizetés szerkesztése

Cím

kóla

Megjegyzés

kóla

típus

Expense

☐ Tartozás-e

Zseb név

Add meg a zseb nevét

Összeg

500

Mégsem Hozzáad

DEBUG

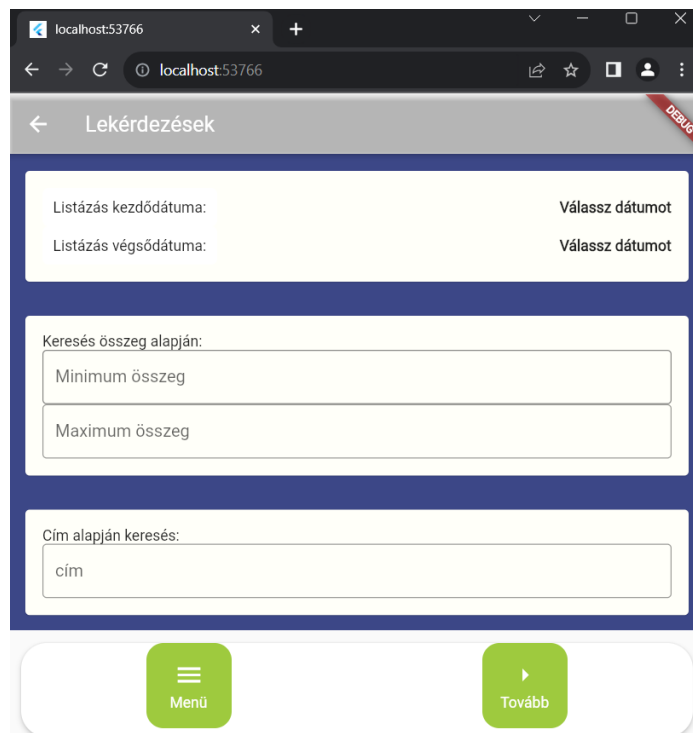
24. ábra: Fizetések szerkesztése nézet weben.

A menüben található "Lekérdezések" gombra kattintva elénk tárul a lekérdezések oldal, ahol számos lehetőség áll rendelkezésre a bevételek és kiadások szűrésére. Először is date-picker-ek segítségével kiválaszthatjuk a szűrés kezdő és végső dátumát, így csak az ezek közötti tranzakciók jelennek meg a szűrt nézetben. Emellett lehetőségünk van beállítani egy minimum és maximum összeget is, és a szűrt nézet csak ezek közötti összegű tranzakciókat fogja megjeleníteni. Ha mindkét helyre ugyanazt az összeget adjuk meg, akkor kizárólag az adott összegű tranzakciókat fogja kiszűrni a program.

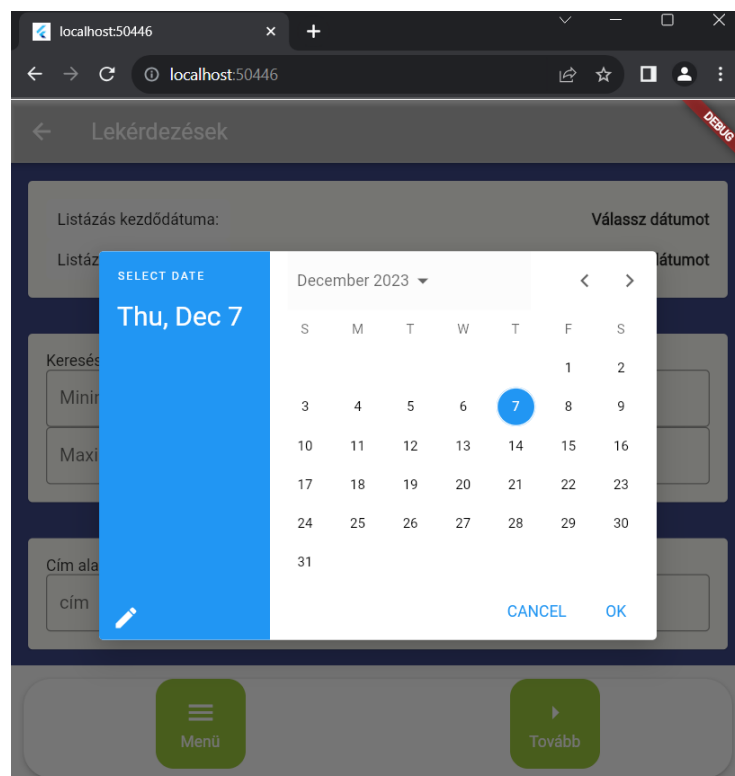
A további lehetőségek között szerepel a cím alapján keresés textbox, ahol egy szó beírásával kiszűrhetjük azokat a tranzakciókat, amelyeknek a címében szerepel a megadott szó. Ugyanezen módon a megjegyzések textbox is lehetőséget nyújt arra, hogy a szóra szűrve megjelenjenek azok a tranzakciók, amelyeknek a megjegyzése tartalmazza azt a szót. Emellett ki tudjuk választani a zsebet, amelyben szeretnénk a tranzakciókat szűrni, és végső soron eldönthetjük, hogy kiadásra vagy bevételre szűrünk.

A "Tovább" gombra kattintva megjelenik a szűrt nézet, ahol minden olyan elem látható, amely megfelelt az előző oldalon beállított feltételeknek. Itt is lehetőség van a

menügomb segítségével visszatérni a főmenübe, biztosítva ezzel a felhasználó számára a navigáció egyszerűségét.



25. ábra: Lekérdezések nézet weben.



26. ábra: Dátum választó weben.

5 Összefoglalás

5.1 Értékelés

A szakdolgozat írása során számos értékes tapasztalattal gazdagodtam, különösen a Flutter keretrendszer összetett projektfejlesztési folyamatának megismerése révén. A projekt során megtapasztalt nehézségek és kihívások által bepillantást nyertem az alkalmazásfejlesztés komplexitásába. Konzulensem segítségével megismerkedtem a hatékony időgazdálkodás fontosságával, időbeosztással, ugyanakkor felismertem, hogy ezen a területen még további fejlődésre van szükségem. Általában 40% körüli határidőre történő befejezést tudtam elérni, és ezáltal felismertem az időmenedzsment terén rejlő lehetőségeket.

Az alkalmazásfejlesztés során számos funkció és finomhangolás megvalósítása lett volna szükséges, amelyekre időhiány miatt nem került sor. A projekt során ráébredtem, hogy a problémamegoldás során sokszor váratlan akadályokba ütközünk, és megtanultam, hogyan lehet hatékonyan és pozitív hozzáállással kezelni ezeket a kihívásokat. A problémák pihentetése és későbbi újbóli megközelítése során többször is tapasztaltam, hogy új nézőpontból közelítve vagy friss elmeállapotban sikerült megtalálni a megoldást. Ezen képességem fejlesztése során számos olyan problémát oldottam meg, amelyek kezdetben lehetetlennek tűntek.

Egy egyetemi tanárom, Dr Gajdos Sándor, inspiráló mondata, miszerint "az a mérnöki, ami működik", gyakran foglalkoztatott a szakdolgozatom írása során. E mondat által tudatosult bennem, hogy az optimalizált működésre kell koncentrálnom, és bár szívesen foglalkoztam volna apró részletekkel, az idő és energia korlátai miatt a hangsúlyt a projekt fontosabb aspektusaira kellett helyeznem. Ennek tudatában vagyok, és büszke vagyok arra, hogy sikerült megtalálnom az egyensúlyt a részletgazdagság és a projekt teljes funkcionalitása között.

5.2 Továbbfejlesztési lehetőségek

Az alkalmazás jelenlegi funkcionalitását tovább fejlesztve számos újításon gondolkodtam és terveztem. Először is, a felhasználói felületet lehetne újragondolni, hogy még vonzóbb és felhasználóbarátabb legyen. Különböző releváns ábrák és egy

passzoló szín és annak különböző árnyalatainak használatával lehetne feldobni a UI-t, ezáltal növelve annak vizuális vonzerejét és használhatóságát.

Az alkalmazás funkcionalitásának további javítása érdekében a zsebek (zseb ikonok) mozgathatóságának bevezetését, így a felhasználók testre szabhatnák a zsebek sorrendjét az egyéni preferenciáik szerint, ezáltal növelve az alkalmazás kényelmét.

Az automatikus bejelentkezés funkció implementálásával a bejelentkezett felhasználó adatait meg lehetne jegyezni, így elkerülhető lenne a folyamatos újra bejelentkezés szükségessége. Akár a biometrikus azonosítást is hozzá lehetne adni.

Az alkalmazás személyiségét és azonosíthatóságát emelhetnénk egyedi logóval, mely a felhasználók számára könnyen felismerhető lenne és a BuXa alkalmazás védjegyévé válhatna.

6 Köszönetnyilvánítás

Elsődlegesen szeretném kifejezni köszönetemet Somogyi Norbert Zsolt konzulensem felé, aki rengeteg segítséget nyújtott a dolgozat elkészítése során. Külön köszönet jár neki, hogy felkarolta a választott témát, és nagyban hozzájárult a sikeres munkához. Az elmúlt hónapok alatt számos értékes tapasztalatot szereztem, amelyek nélkülözhetetlenek voltak a fejlődésemhez.

Köszönettel tartozom családomnak is, akik rendkívüli támogatást nyújtottak számomra, különösen a munka kezdeti és befejező szakaszaiban.

Szeretném külön kiemelni Sitkéry Iván nevét, aki az utolsó pillanatban az év elején az én nevemben visszalépett a Kotlin alapú szoftverfejlesztés témától, mivel nem volt már hely és tudta, hogy nekem viszont nagyon kedves ez a téma.

Mindezekért hálás vagyok, és mély tisztelettel adózom mindazoknak, akik részt vettek a személyes és szakmai fejlődésemben.

Irodalomjegyzék

- [1] Firebase Authentication <https://firebase.google.com/docs/auth>
(2023 nov. 23)
- [2] Firebase Firestore Database <https://firebase.google.com/docs/firestore>
(2023 dec. 6)
- [3] Flutter SQLite <https://docs.flutter.dev/>
(2023 dec. 6)
- [4] Firestore NoSQL https://youtu.be/v_hR4K4auoQ?si=DxwGKohOsnFBwBsL
(2023 dec. 6)
- [5] MVVM Repository architektúra ábra
https://www.google.com/search?sca_esv=587017952&sxsrf=AM9HkKl2-4fwjKcNvqPpK6tWGWzbpYgixg:1701452922090&q=mvvm+repository&tbm=isch&source=lnms&sa=X&sqi=2&ved=2ahUKEwiSvZeI5u6CAxWy2QIHHRlBCQQ0pQJegQICxAB&biw=1536&bih=707&dpr=1.25#imgsrc=rmBesAdArRZbhM
(2023 nov.)
- [6] GitHub <https://kinsta.com/knowledgebase/what-is-github/>
(2023 dec. 7)
- [7] Flutter <https://docs.flutter.dev/get-started/editor>
(2023 dec. 7)
- [8] Visual Studio Code https://docs.flutter.dev/tools/vs-code?gclid=CjwKCAiApaarBhB7EiwAYiMwqgafYL7obBKo8GDa-94skv1Uc7PphJNgjs1II_4dyRFNX6JLJmFpEBoCNaAQAvD_BwE&gclidsrc=aw.d_s
(2023 nov. 10)
- [9] Android Studio Emulator <https://developer.android.com/studio/run/emulator>
(2023 nov. 6)
- [10] Firebase SDK <https://firebase.google.com/docs/firestore/client/libraries>
(2023 dec.)
- [11] Revolut <https://en.wikipedia.org/wiki/Revolut>
(2023 nov. 5)
- [12] MVVM Repository architektúra <https://digital-solutions.consulting/uncategorized/repository-in-androids-mvvm-architecture/>
(2023 dec. 7)
- [13] ACID <https://hu.wikipedia.org/wiki/ACID>
(2023 dec. 2)

- [14] Dart language <https://dart.dev/>
(2023 dec. 2)
- [15] GitHub Desktop <https://desktop.github.com/>
(2023 dec. 2)