BuXa

Feladat:

Egy olyan projektet akarok létrehozni, amely átlátható és rendezett módon kezelni a költéseket és bevételeket egy Android alkalmazás segítségével. Az alkalmazás lehetővé teszi a felhasználó számára, hogy egyszerre kezelje a pénzmozgásokat, költéseket és bevételeket, ahol minden információ rendezetten és átláthatóan jelenik meg. A felhasználó képes lesz létrehozni zsebeket, amelyek segítségével csoportosíthatja és nyomon követheti a költéseket és bevételeket. Emellett különböző lekérdezéseket is végezhet a bejegyzéseken.

Az alkalmazás két adattárolási lehetőséget biztosít a felhasználó számára:

Felhőszerveren történő mentés: A felhasználó mentheti az összes adatot egy biztonságos, távoli szerverre, hogy azok megmaradjanak, még akkor is, ha a telefonja meghibásodik vagy elveszik. Ez lehetővé teszi a felhasználó számára, hogy bármilyen eszközről hozzáférjen az adatokhoz, amennyiben bejelentkezik a fiókjába.

Helyi mentés: Az alkalmazás lehetőséget biztosít a felhasználónak arra, hogy helyileg mentse az adatokat a készülékére. Ez lehetővé teszi az adatok hozzáférhetőségét internetkapcsolat nélkül is, így biztosítva a mentésüket és elérhetőségüket akár offline állapotban is.

Az alkalmazás fő szempontja a felhasználóbarát tervezés. Az alkalmazás dizájnja intuitív, ergonomikus és felhasználóbarát, a könnyen navigálható felhasználói felülettel és testreszabható megjelenéssel. A felhasználónak lehetősége lesz testre szabni a gombok kinézetét és a megjelenést az egyéni preferenciák szerint.

Az alkalmazásba való belépéshez jelszóvédelem szolgál, amely megakadályozza illetéktelen hozzáférést az adatokhoz. Ez biztosítja az adatok biztonságát és védelmét.

Az alkalmazás különleges funkciója a tartozások hatékony kiszámítása. Az alkalmazás lehetőséget biztosít arra, hogy az aktuális felhasználó által vezetett tartozások alapján hatékony módon ki tudja számolni, hogy milyen minimális tartozásmegadással lehetséges mindenkit nullára hozni, hogy csak a saját költéseiért fizetett. Ehhez egy megfelelő algoritmus

lesz implementálva, amely optimalizálja a tartozások rendezését és minimálisra csökkenti a tartozások számát.

Az alkalmazás lehetővé teszi a befektetések nyomon követését is. A felhasználó képes lesz rögzíteni és követni a befektetésekkel kapcsolatos információkat, például a portfólió értékét, adott időpontban, vagy a hozamokat.

Az alkalmazás architektúrája MVVM (Model-View-ViewModel) alapú lesz, amely biztosítja a különböző rétegek (modell, nézet, nézetmodell) közötti hatékony adatáramlást és elkülönülést.

A kódolás során az elegáns és hatékony megoldásokat preferálom, amelyek biztosítják az alkalmazás teljesítményét és karbantarthatóságát. A kódolás során alkalmazok majd megfelelő tervezési mintákat és struktúrákat, hogy az alkalmazás könnyen bővíthető legyen a jövőben. (akár mások által is)

Egy mondatban: Egy olyan alkalmazást akarok létrehozni, ami rendszerezetté teszi az ember pénzügyeit és jó használni. :D

Technológiák:

Az alkalmazás fejlesztése során az alábbi technológiákat és eszközöket használtam:

Fejlesztői környezet: Az Android alkalmazás kódolásához az Android Studio-t használtam. Ez az integrált fejlesztői környezet kínál számos funkciót és eszközt az alkalmazás tervezéséhez, fejlesztéséhez és teszteléséhez. Itt Kotlin nyelven írtam a kódot.

Emulátor és fizikai eszköz: Az Android Studio beépített emulátorát használtam az alkalmazás tesztelésére és a felhasználói élmény vizsgálatára. Emellett saját Android telefonomat is használtam a valós környezetben történő teszteléshez és fejlesztéshez.

Lokális adattárolás - Room Database: A költések és bevételek helyi tárolásához a Room adatbázist használtam. Ez egy SQLite alapú adatbáziskezelő rendszer, amely lehetővé teszi az adatok hatékony és strukturált tárolását a készüléken.

Felhőszerver - Firebase: A projektben a Firebase szolgáltatásokat használtam a felhőszerveren történő adattároláshoz és felhasználókezeléshez. Konkrétan a Firestore adatbázist használtam az adatok felhőalapú tárolásához, valamint az Authentication funkciókat használtam a felhasználók bejelentkezésének és az alkalmazásba való hozzáférés korlátozásához.

Adatkötés - Data Binding és View Binding: Az alkalmazásban adatkötést használtam a felhasználói felület és a háttérlogika közötti kommunikációhoz. A *Data Binding* technológiát alkalmaztam, amely lehetővé teszi az adatok automatikus kötését a felhasználói felület elemeihez. Emellett néhány esetben a *View Binding*-et is használtam az XML nézetek könnyű hozzáférhetőségéhez és kezeléséhez.

XML - Felületleírás: A felhasználói felület tervezéséhez és leírásához XML kódot használtam. Az XML nyelv lehetővé teszi a felhasználói felület elemeinek és elrendezésének részletes meghatározását, beleértve a gombokat, szövegeket, képeket stb.

MVVM (Model-View-ViewModel) architektúra: Az alkalmazás fejlesztése során az MVVM tervezési mintát követtem. Ez az architektúra elkülöníti a modellt (adatok és üzleti logika), a nézetet (felhasználói felület) és a nézetmodellt (kommunikáció a modell és a nézet között). Ez a struktúra javítja a kód olvashatóságát, karbantarthatóságát és tesztelhetőségét. *Repository*-kat is létrehoztam a megfelelő helyeken.

Dizájn és felhasználói élmény: A jelenlegi fejlesztési szakaszban az alkalmazásban saját beépített elemeket használtam a dizájnhoz. Azonban a jövőben tervezem a *Material Design* irányelvek alkalmazását is a modern és felhasználóbarát felhasználói élmény eléréséhez.

Ezekkel a technológiákkal és eszközökkel a célom az volt, hogy hatékony és megbízható alkalmazást fejlesszek, amely könnyen bővíthető és felhasználóbarát. Remélem, hogy ezek a részletek segítenek az alkalmazás megtervezésében és fejlesztésében.

Megoldási módszerek, meddig jutottam:

Log In/Bejelentketés

Az alkalmazás minden belépési kísérletnél kéri a felhasználót, hogy adja meg az email címét és a jelszavát. Amennyiben a felhasználónak még nincs regisztrált fiókja, lehetősége van regisztrálni. Az email címnek egyedinek kell lennie, vagyis más felhasználó nem használhatja ugyanazt az email címet. A jelszónak tartalmaznia kell legalább egy számot, legalább egy nagybetűt, és legalább 8 karakter hosszúnak kell lennie.

Az alkalmazás a felhasználóval toast üzenetek segítségével kommunikál, amelyek segítségével tájékoztatja a felhasználót a problémákról vagy hibákról, ha ilyenek adódnak. Így a felhasználó pontosan tudja, mi a gond, ha valami probléma merül fel.

Az alkalmazás Log In funkciójának megvalósításához a következő megoldási módszereket alkalmaztam:

ActivityLogin: Ez az osztály felelős az *Activity* felépítéséért és az UI események kezeléséért. Itt inicializálom a szükséges komponenseket, például a gombot és a *TextView*-t. A regisztráció gombra kattintva átirányítom a felhasználót az *ActivityRegister* aktivitásra. Beállítom a *ViewModelLogin* objektumot és figyeljük a beviteli mezők változásait. Az *onCreate* metódusban létrehozok egy *FirebaseAuth* objektumot a felhasználói hitelesítéshez.

ActivityRegister: Ez az osztály felelős az új felhasználó regisztrációjáért. Az *onCreate* metódusban inicializálom a szükséges komponenseket, mint például a gombot és a *ProgressBar*-t. A *performSignUp* metódus kezeli a regisztrációs folyamatot. Az *auth.createUserWithEmailAndPassword* metódus segítségével regisztrálom az új felhasználót. Ha a regisztráció sikeres, beállítódik a felhasználó összes adata a *Firestore* adatbázisban, majd átirányítódik a felhasználó a bejelentkezési oldalra.

LoginListener interfész: Ez az interfész definiálja a bejelentkezéssel kapcsolatos eseményeket, mint például a *Toast* üzenet megjelenítése, az ellenőrzés és a menü hívása.

ViewModelLogin: Ez a *ViewModel* osztály felelős a bejelentkezési folyamat üzleti logikájáért. Az *onLoginButtonClick* metódusban kezelem a bejelentkezési gombra kattintást. Ellenőrzöm, hogy az email és jelszó mezők üresek-e. Ha nem, akkor a *auth.signInWithEmailAndPassword* metódussal bejelentkeztetjük a felhasználót. Ha a bejelentkezés sikeres, meghívom a *check* függvényt a *LoginListener* segítségével.

Ezek az osztályok és az ezek közötti kommunikáció egy szerintem egyszerű bejelentkezési folyamatot valósítanak meg egy Android alkalmazásban.

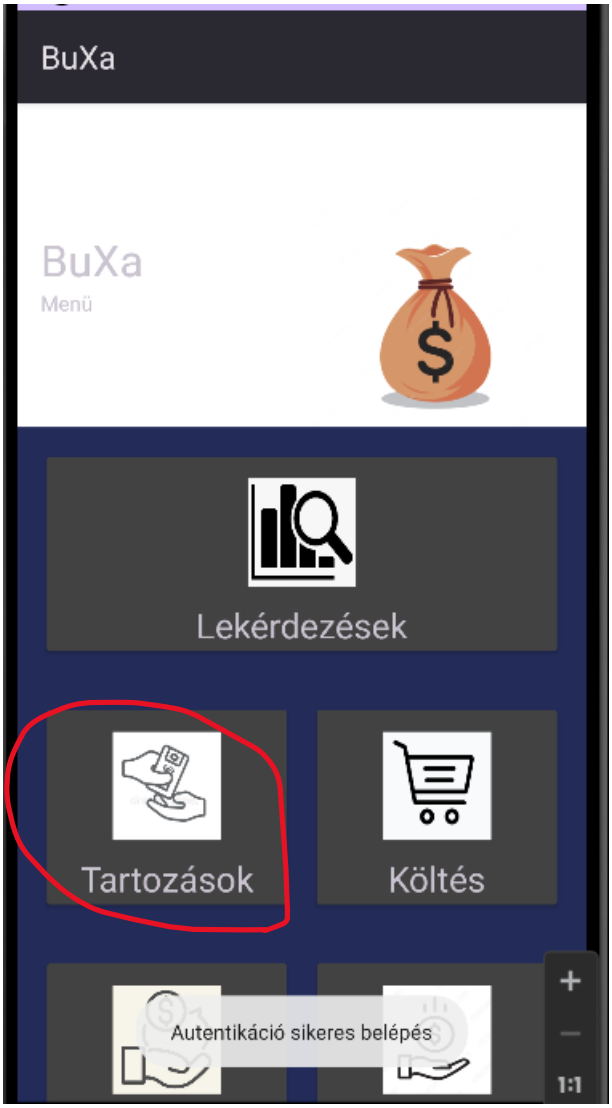
Tartozás:

Az alkalmazás ezen funkciója lehetővé teszi a tartozások felvételét az emberek között. A felhasználók a "plusz" gomb segítségével adhatnak hozzá tartozásokat, amelyek rögzítik, hogy ki tartozik kinek. Emellett lehetőség van a tartozások törlésére is.

Amikor a felhasználó a "Tovább" gombra kattint, egy új activity nyílik meg, ahol a megadott tartozásokat kiválaszthatja pipálás útján. Fontos megjegyezni, hogy jelenleg a pipálás nem vonja vissza az adott tartozást a számítások során.

Az activity-ben található "Számolás" gombra kattintva az alkalmazás kiszámolja, hogy a tartozások között hogyan lehet a lehető legkevesebb tartozásból és mely konkrét tartozásokból felépíteni a tartozás-láncot, hogy mindenki csak a saját tételéért fizessen. Ezzel a funkcionalitással az alkalmazás segít optimalizálni a tartozások rendezését, és megkönnyíti a felhasználók számára a kiadások rendezését és a tartozások kiegyenlítését.

Az alkalmazás debt/tartozás funkciójának megvalósításához az alábbi osztályokat használtam:



DaoDebt: A DebtItem entitások kezelésére használható adatműveleteket definiál. Tartalmazza az adatbázisból való lekérdezést, beszúrást, frissítést és törlést.

DebtItem: Reprzentálja a tartozások elemét az adatbázisban.

RepositoryDebt: Meghatározza az adatbázis verzióját. Az osztály segítségével hozzáférhetünk az adatbázishoz.

FunctionDebt: Az alkalmazás fő tevékenysége (Activity) a tartozások kezeléséhez.
Inicializálja a ViewModel-t, a RecyclerView-t, a gombokat és a fragmenteket.
Megjeleníti az új tartozás hozzáadására szolgáló dialógust.
Az adatbázishoz és az Adapter-hez tartozó metódusokat hívja.

AdapterDebt: Az AdapterDebt osztály egy RecyclerView-hoz tartozó adapter, amely megjeleníti a tartozásokat. (elengedhetetlen) Az osztály felelős az egyes elemek megjelenítéséért és azok eseményeinek kezeléséért. A metódusai segítségével hozzáadhatunk, frissíthetünk vagy törölhetünk elemeket a RecyclerView-ból.

FragmentDebt: Egy dialógusablakot reprezentál, amely a tartozás hozzáadásához szolgál. Implementálja a FragmentInterface interfészt, amely segítségével a dialógusból értesítést küldhetünk a fő Activity-nek.

ViewModelDebt: Egy ViewModel osztály, amely az adatok üzleti logikájáért felelős. Tartalmazza az adatbázis és az adapter referenciáit. Kezeli a tartozások hozzáadását, törlését és frissítését az adatbázisban. Details osztályt, amely felelős a tartozások részleteinek megjelenítéséért. A DetailsActivity az alkalmazás második fő tevékenysége, és az Intent segítségével kapja meg a kiválasztott tartozás azonosítóját a FunctionDebt tevékenységtől. Az osztály inicializálja a tartozás részleteinek megjelenítésére szolgáló nézeteket és betölti a kiválasztott tartozás adatait az adatbázisból.

ViewModelDetails: Egy ViewModel osztály, amely felelős a tartozások részleteinek üzleti logikájáért. Tartalmazza az adatbázis és a kiválasztott tartozás referenciáit. A ViewModelDetails osztály lehetővé teszi a tartozás adatainak lekérdezését és frissítését.

Ez az osztály tartalmaz egy fontos algoritmust:

Az algoritmus, amely kiszámolja a minimális szükséges tartozások számát és az ezekhez tartozó tételeket annak érdekében, hogy mindenki csak a saját tartozásait fizetve nullára jöjjön ki, az alábbi módon működik:

- **Előkészületek:**

Készítsünk egy listát, amelyben minden személyhez hozzárendelünk egy értéket.(ezt a kódban a Key osztály valósítja meg) Ez az érték pozitív lesz, ha a személynek pénzt kell tennie a közös kosárba, és negatív lesz, ha pénzt kell onnan vennie.

- Rendezzük a személyeket az értékek alapján csökkenő sorrendbe.
- Ciklus:

Amíg van olyan személy, akinek az értéke nem nulla, folytassuk az alábbi lépéseket:

Válasszuk ki a legnagyobb pozitív értékű személyt és a legkisebb negatív értékű személyt.

Kiszámítjuk, hogy a pozitív értékű személynek mennyit kell fizetnie a negatív értékű személynek. Ez a két érték az abszolút értékükkel lesz megegyező, tehát a pozitív értékű személy ennyit helyez a közös kosárba, míg a negatív értékű személy ezt a mennyiséget veszi ki onnan.

  - A két személyhez hozzárendelt értékek módosítása:

A pozitív értékű személyhez tartozó értéket csökkentjük a fizetett összeggel.
A negatív értékű személyhez tartozó értéket növeljük a fizetett összeggel.

Az algoritmust rekurzívan hívjuk meg, és folytatjuk a ciklust.

- Eredmények:

Amikor minden személyhez hozzárendelt érték nulla, az algoritmus befejeződik.

Az eredmény egy olyan lista, amely tartalmazza a fizetéseket vagy tartozásokat. Minden bejegyzés tartalmazza a fizető személyt, a kedvezményezettet és az összeget.

Röviden: Az algoritmus végrehajtása során a személyek közötti tartozásokat úgy kezeljük, mintha lenne egy közös kosár, ahová a tartozásokat helyezzük be vagy onnan veszünk ki. A cél az, hogy mindenki a saját tartozásait fizetve nullára jöjjön ki, és a kosár végül üres legyen. Az algoritmus a személyekhez hozzárendelt értékeket módosítja a fizetések során, és rekurzívan folytatja a folyamatot, amíg mindenki egyenlő mértékben tartozik vagy fizet.

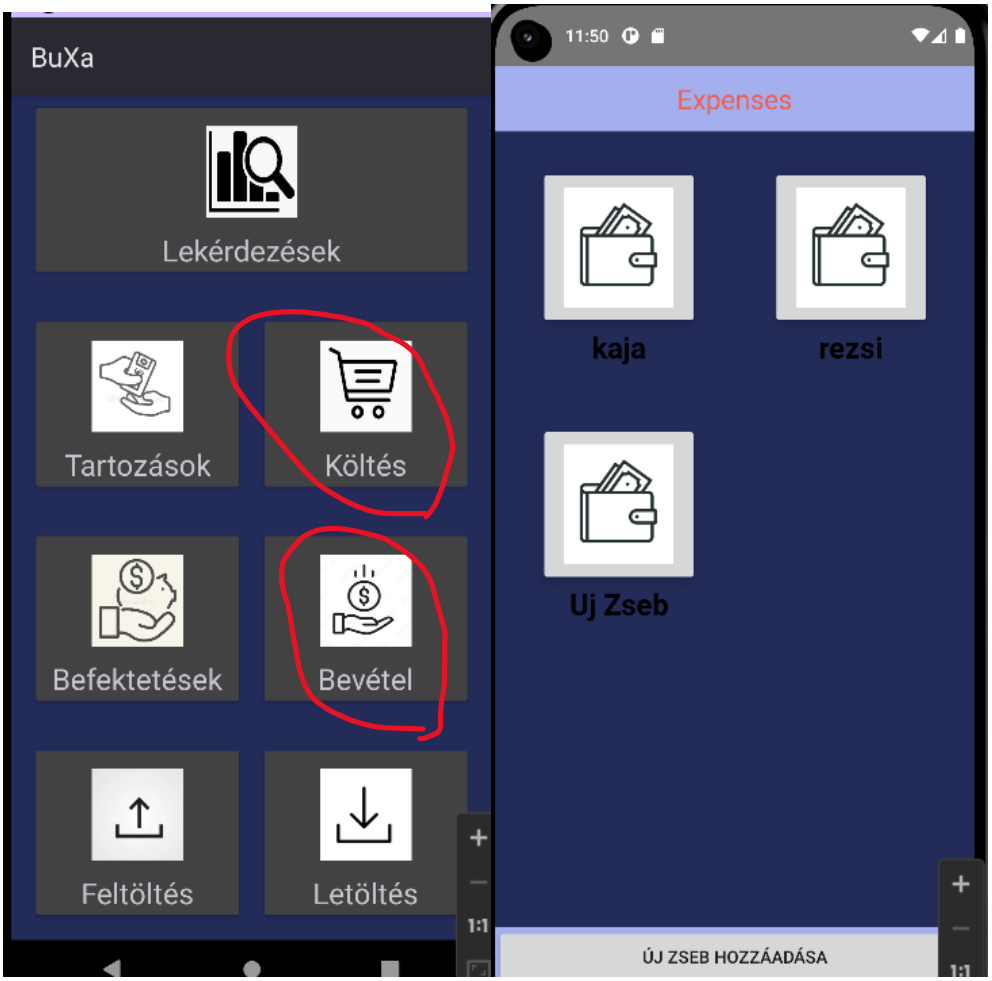
Költés/Bevétel

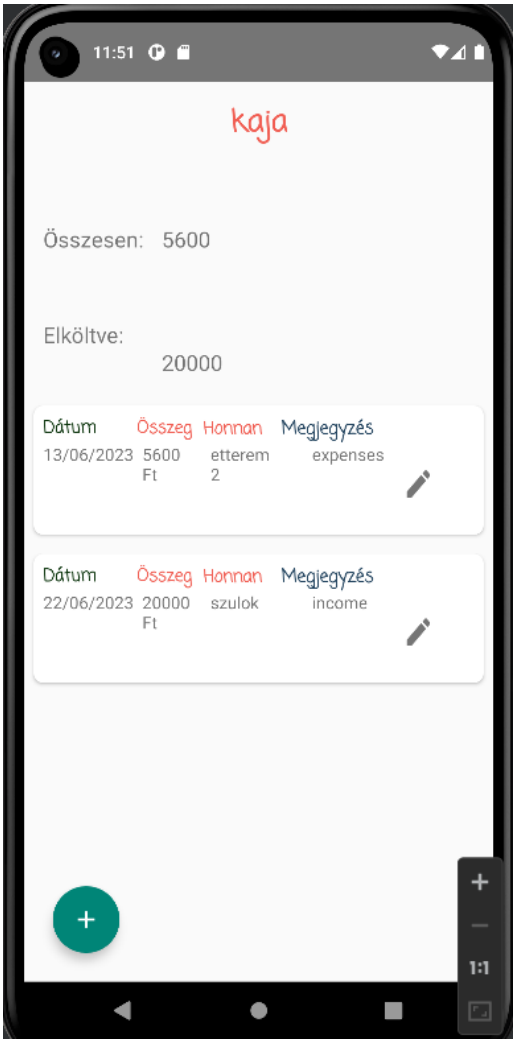
Amikor a felhasználó szeretne egy költést rögzíteni az alkalmazásban, az "Költés" gombra kattint. Ezen az oldalon létrehozhat egy új zsebet, amelyben a költési adatokat fogja megadni. Az "Új zseb hozzáadása" gomb segítségével lehetősége van új zsebet létrehozni, és meg kell adnia a zseb nevét. Ezenkívül lehetőség van egy kép hozzárendelésére is a zsebbe. Ha hosszan nyomja a zseb nevét, megnyílik a galéria, ahol kiválaszthatja a képet, amelyet szeretne beállítani a zsebhez. Fontos megjegyezni, hogy jelenleg a kilépés után a kép még nem mentődik el.

Miután belépünk a zsebbe, láthatjuk az összes költést és bevételt, amit ebbe a zsebbe rögzítettünk. A zseb neve, az abban lévő pénzmennyiség, valamint a költött összeg is megjelenik a tetején. A "Költések" menüből csak költést lehet rögzíteni a plusz gomb segítségével. Ha rákattint a gombra, megjelenik egy dialódusablak, amelyet a felhasználó kitöltve létrehozhatja az új költési bejegyzést. Ha a bejegyzésen a kis ceruza ikonra kattintunk, módosíthatjuk vagy törölhetjük azt. A dátumot egyszerűen kiválaszthatjuk a DatePicker segítségével. A megjegyzés részén jelenleg csak a "bevétel" vagy az "kiadás" kulcsszó jelenik meg, attól függően, hogy egy költési vagy bevételi bejegyzést hoztunk létre. Ez azért van jelen jelenleg, hogy szemléltessem a bejegyzések típusát.

A bevétel (income) létrehozása ugyanígy működik, csak a "Bevétel" menüre kell az elején rákattintani, és ott lehet rögzíteni az új bejegyzést.

Az alkalmazás Költés/Bevétel funkciójának megvalósításához a következő megoldási módszereket alkalmaztam:


BuXa
Lekérdezések
Tartozások
Költés
Befektetések
Bevétel
Feltöltés
Letöltés
11:50
Expenses
kaja
rezsi
Uj Zseb
ÚJ ZSEB HOZZÁADÁSA



ActivityPocket: Ez az osztály felelős a zsebaktivity megjelenítéséért és kezeléséért. Az onCreate metódusban inicializálja a szükséges komponenseket, beállítja az adaptert és a ViewModelt. A gombokra kattintásokat kezeli, például az új zseb hozzáadását és a kép kiválasztását. A AdapterInterface interfész metódusait implementálja, amelyek a zsebekre kattintásokat és a kép kiválasztását kezelik.

AdapterPocket: Ez az osztály felelős a zsebek megjelenítéséért és kezeléséért a RecyclerView segítségével. A ViewHolder osztály a RecyclerView elemek megjelenítését végzi. Az adapterben definiált metódusok kezelik az elemek hozzáadását, frissítését és a zsebekre való kattintásokat.

ViewModelPocket: Ez az osztály felelős a zsebmodellek kezeléséért és az adatok betöltéséért a RepositoryExpenses segítségével. A getPocketsList metódus visszaadja a zsebek listáját. A addPocketToList metódus hozzáad egy új zsebet a listához. A loadItemsInBackground metódus háttérszálon betölti az adatokat és frissíti a zsebek listáját.

ActivityImagePicker: Ez az osztály felelős a kép kiválasztásáért a galériából. Az onCreate metódusban inicializálja a szükséges komponenseket, például az ImageView-t és a gombot. A onActivityResult metódus kezeli a kép kiválasztása utáni eredményt és beállítja az ImageView-ba a kiválasztott képet.

ActivityInPocket osztály: Ez az osztály felelős az Activity kezeléséért, amely az alkalmazás egyik képernyőjét reprezentálja. Az osztály inicializálja és kezeli a ViewModel-t, az adatkötést (binding), az adaptert és a felhasználói felülettel kapcsolatos eseményeket. A onCreate metódusban történik az inicializáció, és beállításra kerül a felhasználói felület.

AdapterInPocket osztály: Ez az osztály egy RecyclerView adaptert valósít meg, amely az elemek megjelenítését és kezelését végzi a listanézetben.

Felelős az elemek létrehozásáért, megjelenítéséért és az események kezeléséért.

Számításokat végez a kiadások és bevételek összegének kiszámításához, ezek később e képernyőre kerülnek, hogy melyik zsebben mennyit költöttek és mennyi pénz van még ott.

FragmentExpenses osztály: Egy párbeszédpanel (dialog), amely a kiadás hozzáadásáért vagy módosításáért felelős.

Tartalmazza az új kiadás adatainak bevitele és a dátum kiválasztását.

FragmentExpensesModify osztály: Egy párbeszédpanel (dialog), amely a kiadás módosításáért felelős.

Tartalmazza az adott kiadás adatainak módosítását és törlését.

ViewModelInPocket osztály: A ViewModel osztály, amely az üzleti logikát valósítja meg a kiadások kezeléséhez.

Tartalmazza a kiadások listáját, és kommunikál az adatbázisréteggel.

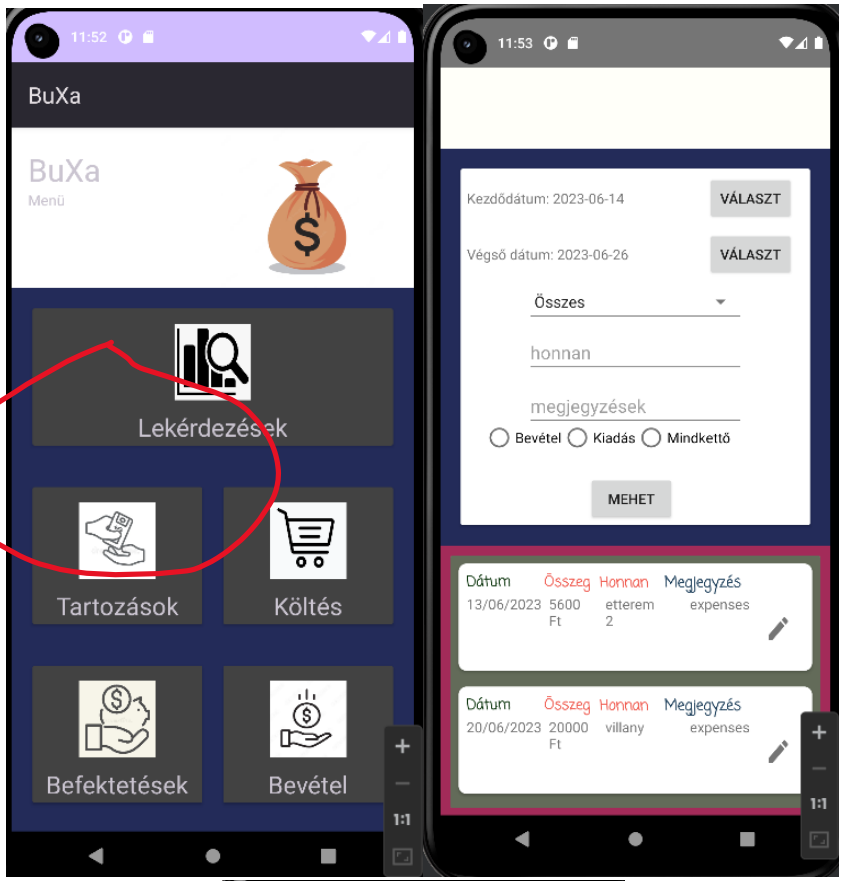
Végzi az adatok betöltését, hozzáadását, törlését és az összesítési számításokat.

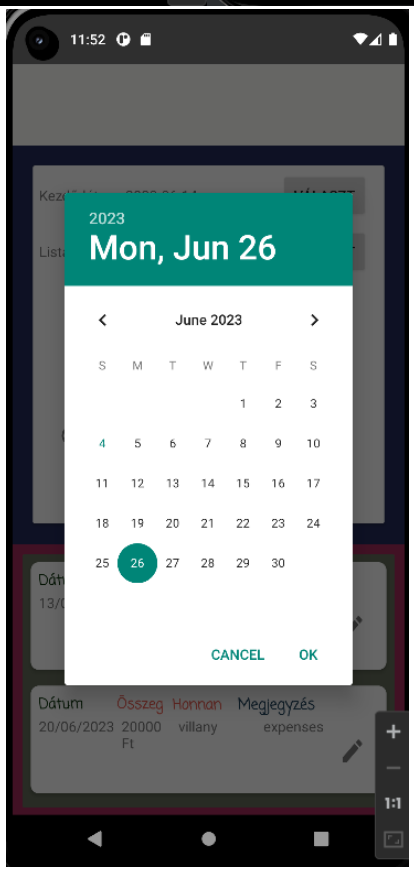
Lekérdezések

Ebben a menüpontban a felhasználó ki tudja választani a listázás kezdeti és végdátumát egy egyszerű DatePicker segítségével. Ez lehetővé teszi a dátumok pontos beállítását. Ezenkívül egy lenyíló kombóBox-ban megtalálja az összes létező zsebet, amelyek közül választhat, és csak az adott zsebben található bejegyzések jelennek meg. A "honnan" és "megjegyzések" szövegdobozokba kulcsszavakat lehet írni. Ha ezek a kulcsszavak megtalálhatók bármely rekord megfelelő mezőjében, akkor azok listázódni fognak.

Az alsó részen, a rádiógombok segítségével, a felhasználó kiválaszthatja, hogy a bevételt, a kiadást vagy mindkettőt szeretné listázni. Miután a felhasználó megadta a szűrési feltételeket, a "Mehet" gomb lenyomása után a rendszer listázza az összes olyan bejegyzést, amely a megadott szűrési kritériumoknak megfelel. Fontos megjegyezni, hogy a szűrési funkciók közül a dátumválasztás, a kulcsszavas keresés a "honnan" mezőben, valamint a zseb kiválasztása már működik és szépen be vannak kötve.

- **Az alkalmazás lekérdezés funkciójának megvalósításához a következő megoldási módszereket alkalmaztam:**


11:52
BuXa
BuXa
Menü
Lekérdezések
Tartozások
Költés
Befektetések
Bevétel
11:53
Kezdődátum: 2023-06-14
VÁLASZT
Végső dátum: 2023-06-26
VÁLASZT
Összes
honnan
megjegyzések
Bevétel
Kiadás
Mindkettő
MEHET
Dátum
Összeg
Honnan
Megjegyzés
13/06/2023
5600 Ft
etterem 2
expenses
Dátum
Összeg
Honnan
Megjegyzés
20/06/2023
20000 Ft
villany
expenses


11:52
2023
Mon, Jun 26
June 2023
S M T W T F S
1 2 3
4 5 6 7 8 9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30
CANCEL
OK
Dátum
Összeg
Honnan
Megjegyzés
20/06/2023
20000 Ft
villany
expenses

ActivityQuery: Ez az osztály egy Activity, amely a felhasználói felületet kezeli a lekérdezések végrehajtásához. Itt találhatók az adatbázis inicializálása, az adapter beállítása, a gombok eseménykezelői, valamint a lekérdezések végrehajtása és eredményeinek megjelenítése a RecyclerView-ben.

AdapterQuery: Ez az osztály egy RecyclerView adapter, amely a lekérdezések eredményeinek megjelenítését végzi. A payments listában tárolja az elemeket, amelyeket a RecyclerView-ben megjelenít. Az adapter felelős az elemek megjelenítéséért és frissítéséért.

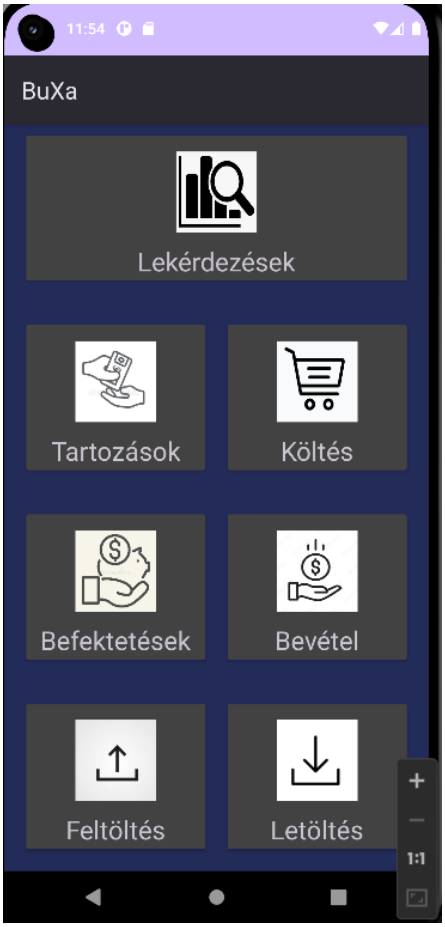
ViewModelQuery: Ez egy ViewModel osztály, amely az ActivityQuery osztállyal kommunikál, és az adatok tárolásáért és feldolgozásáért felelős. Az osztályban találhatók metódusok a lekérdezések szűrésére és a RecyclerView adapterének frissítésére.

Feltöltés Firebase database-ba, Letöltés Firebase-ról

Amennyiben már vannak rekordok a Tartozás menüpontban, lehetőségünk van azokat feltölteni a Firebase szerverre a "Feltöltés" gomb segítségével. Ezáltal az adatok biztonságosan el lesznek tárolva a szerveren. Ha később szeretnénk ezeket a rekordokat törölni a telefonról, akkor a "Letöltés" gombra kattintva újra letölthetjük és megjeleníthetjük azokat.

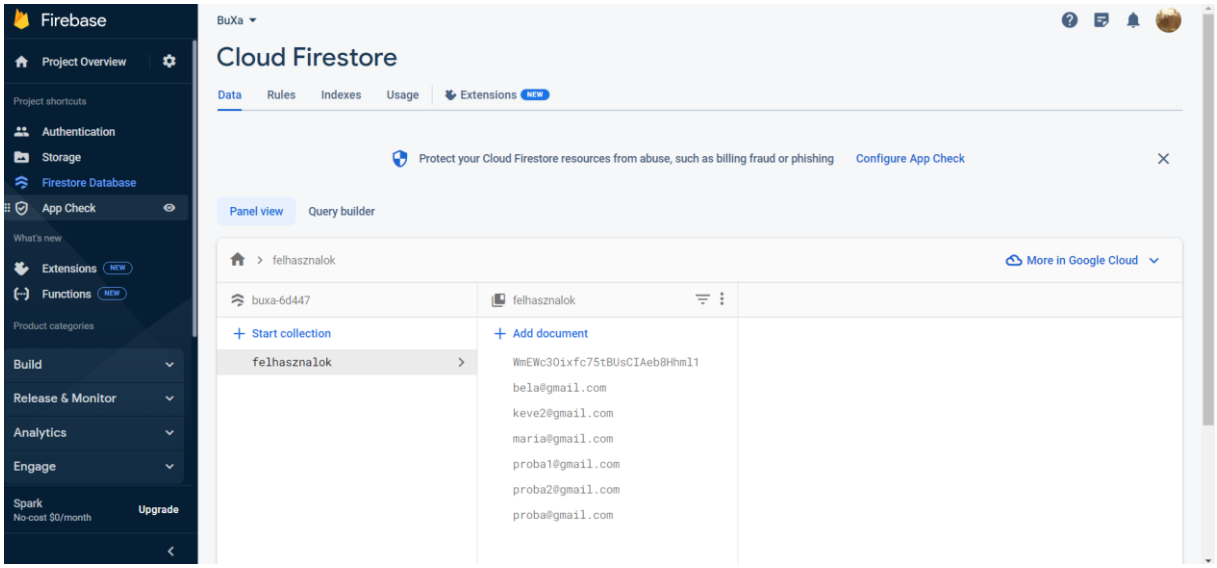
Ez a folyamat lehetővé teszi a rekordok biztonsági mentését a Firebase szerveren, így ha töröljük azokat a telefonról, még mindig visszaállíthatjuk őket az alkalmazásba a letöltés gomb használatával. Ezáltal nem veszítünk el semmilyen fontos információt és könnyedén visszaállíthatjuk a korábbi rekordokat az alkalmazásban.

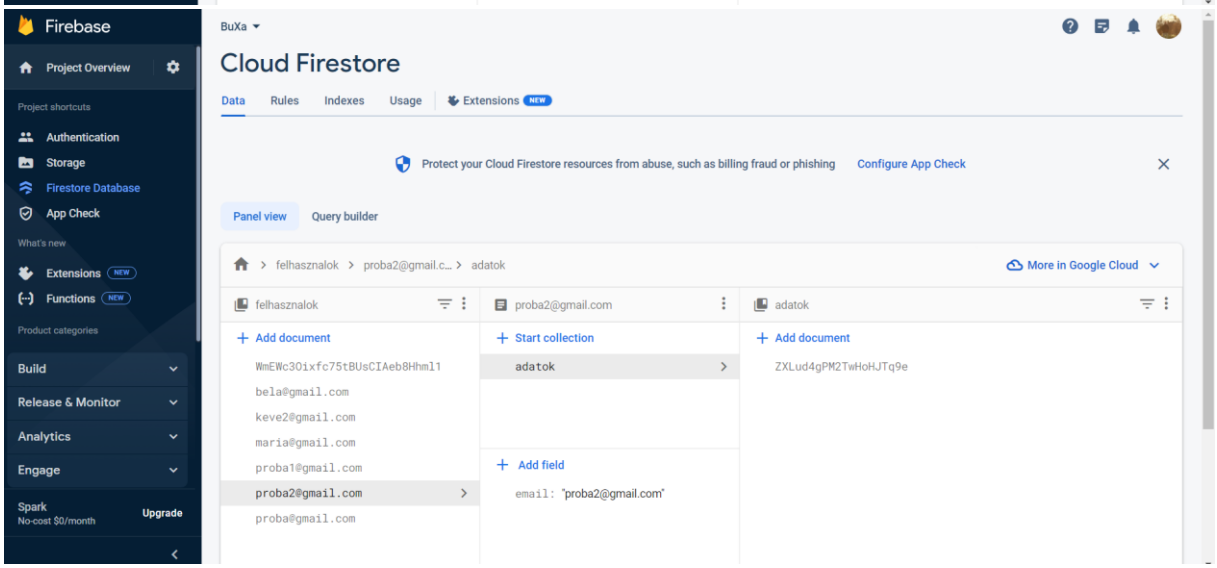
- **Az alkalmazás Menü, Feltöltés Firebase database-ba, Letöltés Firebase-ról funkciójának megvalósításához a következő megoldási módszereket alkalmaztam:**



MenuActivity: Ez az osztály az alkalmazás főmenüjét kezeli. Itt találhatók a különböző gombok eseménykezelői, amelyek más Activity-ket hívnak meg a megfelelő funkciókhoz. Emellett itt találhatók a mentési és betöltési funkciók, amelyek az adatokat a Firestore adatbázisba mentik és onnan töltik be.

Feltöltés, Letöltés: A Firebase adatbázisra való feltöltés és onnan való letöltés jelenleg csak a tartozások lokális adatbázisára, azaz a "debtList"-re vonatkozik. A kapcsolódó függvények a "debt" modul osztályaiban találhatók. Mivel a Firebase egy NoSQL adatbázis, amelyben kollekciók és dokumentumok vannak, létrehoztam egy "felhasznalok" nevű kollekciót, amelyben dokumentumok találhatók. Annak érdekében, hogy minden dokumentumnak egyedi neve legyen, az e-mail címet használom dokumentum nevként, ezáltal biztosítva az egyediséget és a keresést. Ezekben a dokumentumokban található egy "adatok" nevű gyűjtemény, amely akkor jön létre, ha még nem létezik, és ebbe kerülnek a felhasználó által feltöltött tartozások.





Az alkalmazás jelenlegi formája ezen a GitHub linken érhető el:
https://github.com/KeveXD/Onlab

Balla Keve
BE1L51
Önálló Laboratórium
2023.06.05.