

Költségvetés-kezelő mobilalkalmazás

Szegedi Tudományegyetem
Informatikai Intézet

SZAKDOLGOZAT

Taskovics Lőrinc Károly

2023

Költségvetés-kezelő mobilalkalmazás

Szegedi Tudományegyetem
Informatikai Intézet

Költségvetés-kezelő mobilalkalmazás

Szakdolgozat

Készítette:

Taskovics Lőrinc Károly

Gazdaságinformatika szakos
hallgató

Témavezető:

Dr. Bilicki Vilmos

egyetemi adjunktus

Szeged

2023

Feladatkiírás

A feladat egy olyan androidra fejlesztett mobilalkalmazás létrehozása, mellyel a felhasználó számon tudja tartani a kiadásait, illetve bevételeit. Az alkalmazásnak tudnia kell felhasználókat kezelni, adatbázisból adatokat lekérni, felvinni, illetve megjeleníteni.

Saját tapasztalataim alapján az emberek többnyire szeretik rendszerezni és vezetni a bevételeiket, kiadásait. Ez ebben a felgyorsult világban szerintem még fontosabb, ugyanis gyakorlatilag már akár egy órával is tudsz percek alatt elkölteni meghatározó összegeket. Nyilván ezek számontartására a bankok a maguk módján rendelkeznek ilyen szoftverekkel, de én úgy gondolom, hogy ezeket új funkciókkal, esetleg nem olyan fontos funkciókkal lehetne bővíteni, csökkenteni. Egy egyszerű példának felhoznám azt, hogy valami oknál fogva kártyás fizetés helyett a készpénzünkért kell nyúlnunk, ezt ezekben az alkalmazásokban nem tudjuk úgy követni, mint egy ilyen, manuális alapokon működő alkalmazásban, mint amilyen az én terveim szerint általam készülni fog. Az alkalmazás a Firebase felhő alapú szolgáltatását fogja használni.

Tartalmi összefoglaló

- **A téma megnevezése:** Költségvetés-kezelő mobilalkalmazás
- **A megadott feladat megfogalmazása:** A feladat az, hogy android rendszerre kell fejleszteni egy alkalmazást. Ennek az alkalmazásnak képesnek kell lennie arra, hogy a felhasználó nyomon tudja követni az esetleges pénzmozgásait. Ennek érdekében egy olyan felületet kell létrehoznom, amely képes a felhasználótól bekérni azokat az adatokat amelyeket a jövőben számon akar tartani.
- **A megoldási mód:** Ahogyan a feladat kiírása is szól, egy androidos alkalmazást csinállok. Ebben különböző funkciókat fogok megvalósítani, ezekkel növelve a felhasználói élményt és az alkalmazás logikáját. Az alkalmazás tud felhasználót kezelni illetve adatbázisok segítségével értékeket, adatokat tárolni. Ezek segítségével valósítom meg az applikáció funkcióit.
- **Alkalmazott eszközök, módszerek:** Az általam írt kód [Java](#) nyelven készült, a program elkészítéséhez az [Android Studio](#) fejlesztőkörnyezetet használtam. Az adatbázisok kezeléséhez a [Firebase](#) felhőalapú szolgáltatást vettem igénybe. A verziókövetést a [Git](#) segítségével oldottam meg, alkalmazásomat lokálisan illetve egy [GitHub](#) repo-ban tároltam. A dokumentációhoz készült ábrákat a [draw.io](#) segítségével készítettem el.
- **Elért eredmények:** Egy működő, a feladatkiírást kielégítő androidos applikáció létrehozása. Az applikáció képes felhasználókat kezelni, megvalósítja a CRUD műveleteket, az adatokból statisztikát tud kimutatni, alkalmas képfeltöltésre is.
- **Kulcsszavak:** android, Java, adatbázis, applikáció, alkalmazás, pénzügy, költségvetés, Firestore, felhasználó, lekérdezés, hozzáadás

Tartalomjegyzék

Tartalmi összefoglaló.....	1
Tartalomjegyzék.....	3
1. fejezet.....	5
Bevezetés.....	5
2. fejezet.....	6
Terület áttekintése.....	6
2.1 SettleUp.....	7
2.2 Banki mobil alkalmazások.....	7
2.3 Eltérések.....	7
3. fejezet.....	8
Funkcionális specifikáció.....	8
3.1 Alkalmazás működésének szemléltetése.....	9
3.2 Autentikáció.....	10
3.2.1 Bejelentkezés.....	10
3.2.2 Regisztráció.....	11
3.3 Főoldal(tranzakciók megjelenítése).....	11
3.3.1 Tranzakció szerkesztése.....	13
3.4 Tranzakció létrehozása.....	13
3.5 Nyugták megjelenítése.....	14
3.5.1 Nyugta létrehozása.....	15
3.6 Diagram.....	16
3.7 Egyes funkciók bemutatása szekvencia diagram segítségével.....	17
4. fejezet.....	18
Felhasznált technológiák.....	18
4.1 Android Studio.....	18
4.1.1. Java.....	19
4.1.2. XML.....	19
4.2 MPAndroidChart.....	19
4.3 Glide.....	20
4.4 Google Firebase.....	20
4.4.1. Firestore Database.....	20
4.4.2. Firebase Storage.....	21
4.5 GitHub.....	21
5. fejezet.....	22
Architektúra.....	22
6. fejezet.....	23
Activity-k és objektumosztályok.....	23
6.1 Activity-k.....	23
6.1.1 MainActivity.....	23
6.1.2 RegisterActivity.....	23
6.1.3 Tr_Retrieve.....	23

6.1.4 Tr_Recycle_Adapter.....	23
6.1.5 Tr_Create.....	24
6.1.6 Tr_Create_Receipt.....	24
6.1.7 Tr_Update.....	24
6.1.8 Tr_Stats.....	24
6.1.9 Tr_Receipt_Retrieve.....	24
6.1.10 Tr_Receipt_Adapter.....	25
6.1.11 DateComparator és RecComparator.....	25
6.2 Objektumosztályok.....	25
6.2.1 Tr_Object.....	25
6.2.2 Tr_Receipt.....	25
7. fejezet.....	26
Biztonság.....	26
8. fejezet.....	27
Adatmodell.....	27
8.1 Tr_Object.....	27
8.2 Tr_Receipt.....	28
9. fejezet.....	29
A rendszer magasszintű folyamatai, működése.....	29
10. fejezet.....	31
Fontosabb kód részletek, ezek ismertetése.....	31
10.1 Autentikáció.....	31
10.1.1 Regisztráció.....	31
10.1.2 Bejelentkezés.....	31
10.2 Tranzakció létrehozása.....	32
10.3 Bejelentkezett felhasználó tranzakcióinak megjelenítése.....	33
10.3.1 Tranzakció lekérő.....	33
10.3.2 Adapter.....	34
10.3.3 Comparator.....	34
10.3.4 Szűrő.....	35
10.3.5 Szerkesztés/törlés.....	35
10.4 Diagram.....	36
10.5 Számla létrehozása.....	37
10.6 Számla megjelenítése.....	38
11. fejezet.....	39
Tapasztalatok továbbfejlesztési lehetőségek.....	39
11.1 Nehézségek.....	39
11.2 Tapasztalatok, továbbfejlesztési lehetőségek.....	39
Irodalomjegyzék.....	41
Nyilatkozat.....	41
Köszönetnyilvánítás.....	43

1. fejezet

Bevezetés

Mindenek előtt a motivációmmal szeretném kezdeni, hiszen egy saját téma során szerintem ez egy nagyon fontos szempont, hogy kifejtthessem a projektem háttérét, indíttatását. Az egyetem alatt rengeteg programozási nyelvet ismerhettem meg, különböző stílusú projektekben vehettem részt, ezek közül úgy érzem, hogy hozzám az egyik legközelebb álló feladat a mobilalkalmazás fejlesztése volt. A kreativitásomat itt tudtam a legjobban kiélni, ennél éreztem azt, hogy teljesen beszippant az, hogy újabb és újabb dolgokat fedezhettem fel, illetve pakolhattam bele a fejlesztett alkalmazásomba. Értelemszerűen a motivációm nagy részét az is adja, hogy ezáltal még jobban el tudom sajátítani a mobilalkalmazások fejlesztését, illetve a Java nyelv rejtelseit.

Most rátérnék magának a témának a kifejtésére. Személy szerint amióta nagyjából önállóan kell gazdálkodnom, fontossá vált számomra és eléggé odafigyelek arra, hogy hova és mennyi pénz folyik ki a “kezeim közül”. Erre nyilván nagyon jó opciót tudnak nyújtani a különböző banki mobilalkalmazások, de például ezekben elég nehézkes a készpénzes pénzmozgásokat nyomon követni. Igaz, hogy ez a formája a pénzmozgásnak kezd eltűnni, de abban egyetérthetünk, hogy egyelőre továbbra is szükség van erre a fajtára is.

Ennek érdekében egy olyan ötletem támadt aminek megvannak a maga hátrányai, de ha az ember figyel arra, hogy hogyan költi a pénzét akkor, ez az alkalmazás megkönnyítheti annak követését. Hátrányként nyilván a manuális adatbevitelt kell megemlíteni, ez nyilván hátrányban van a banki alkalmazásokkal szemben, melyek egy-egy kártyás fizetés után ezeket egyből tudják automatán kezelni, eltárolni. Előnyként viszont, meg lehet említeni még egyszer a készpénzes pénzmozgásokat, emellett véleményem szerint hasznosabb statisztikai kimutatásokra képes az alkalmazásom és a számlákat is tárolni tudjuk a felhőben.

2. fejezet

Terület áttekintése

Célom egy olyan célra törő alkalmazás fejlesztése volt, ami egyaránt tartalmazza a piacon lévő banki illetve pénzügyi mobil alkalmazások legfontosabb funkcióit saját ötletekkel, funkciókkal kiegészítve. Alapul a már általam is ismert és használt applikációkat vettem, nevesítve az OTP Bank alkalmazását és a társasági köreimben használt SettleUp-ot.

Az alkalmazás véleményem szerint elég lényegretörő, az általam fontosnak tartott funkciókkal lett kibővítvé, de az ilyen típusú alkalmazások alapvető identitására épül. Munkám elején az általam használni kívánt funkciókat kigyűjtöttem a már piacon lévő alkalmazásokból, így elkészült egy piackutatási táblázat.

		Saját alkalmazás	SettleUp	Banki mobil alkalmazások
Funkciók	-Valuta testreszabása	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
	-Pénzforgás bevitele	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
	-Erdemények exportálása	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
	-Statisztikák a költségekről	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	-Adatok leírása	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
	-Diagram generálása adatokból	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	-Nyugta feltöltése	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	-Kategorizálás	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

2. ábra:Piackutatás

Ebben azt láthatjuk, hogy ez esetben a három alkalmazás funkciói hogyan viszonyulnak egymáshoz. Nyilvánvalóan a piacon jelen lévő alkalmazások számtalan plusz funkcióval is rendelkeznek, de ezeket úgy ítélt meg, hogy speciális esetekben használják a felhasználók. Most be szeretném mutatni külön-külön az összehasonlításhoz talált applikációk tulajdonságait.

2.1 SettleUp

Ezt az alkalmazást saját példámon keresztül szeretném bemutatni. Elsősorban én társasági szinten használom, tegyük fel, hogy egy kollégium légtérben lakunk többen. Nyilván ennek az életmódnak vannak közös költségei, ahhoz, hogy ezeket nyilván tudjuk tartani az adott ember aki mondjuk vesz egy konyhai papírtörülőt amelyet a szobában lakók ugyanúgy használnak, az alkalmazás segítségével be tudja írni, hogy a költségeket az adott emberek között hogyan osszuk fel. Az applikáció kismillió kisebb-nagyobb funkcióval rendelkezik, melyeket gyakran vagy akár még sohasem vettünk igénybe.

2.2 Banki mobil alkalmazások

Ahogy fentebb már említette, nyilvánvaló okok miatt tapasztalatom csak az OTP Bank applikációjával van. A koncepció egy kicsit más, nyilván itt a bank szolgáltat számunkra információt a számláinkról. Alapvetően ez az alkalmazás nagy részben csak “nézelődésre” szolgál, itt a hangsúly az adatok megjelenítésére van nem pedig új adatok létrehozásáról. Az alkalmazás tele van hasznos funkcióval, melyből ötleteket merítettem. Képes kategóriákba sorolni a felhasználó tranzakcióit, ebből pedig egyfajta kimutatást is biztosít. Nyomon Követhető minden olyan pénzmozgás amely az adott számlán történik. Képesek vagyunk utalást is végezni számlánkról.

2.3 Eltérések

Legfőbb eltérésnek azt említeném, hogy a felhasználó számon tudja tartani, hogy az adott tranzakció milyen típusú, lehet kártyás vagy készpénzes. A banki alkalmazásokból átvettem a kategorizálást, ennek mentén az appom képes diagramokat generálni a kategóriák mentén. A másik teljesen eltérő funkció az a nyugtakezelés. Lehetősége van a felhasználónak feltöltenie az adott tranzakcióhoz tartozó nyugtát, ez egy plusz ellenőrző faktor.

3. fejezet

Funkcionális specifikáció

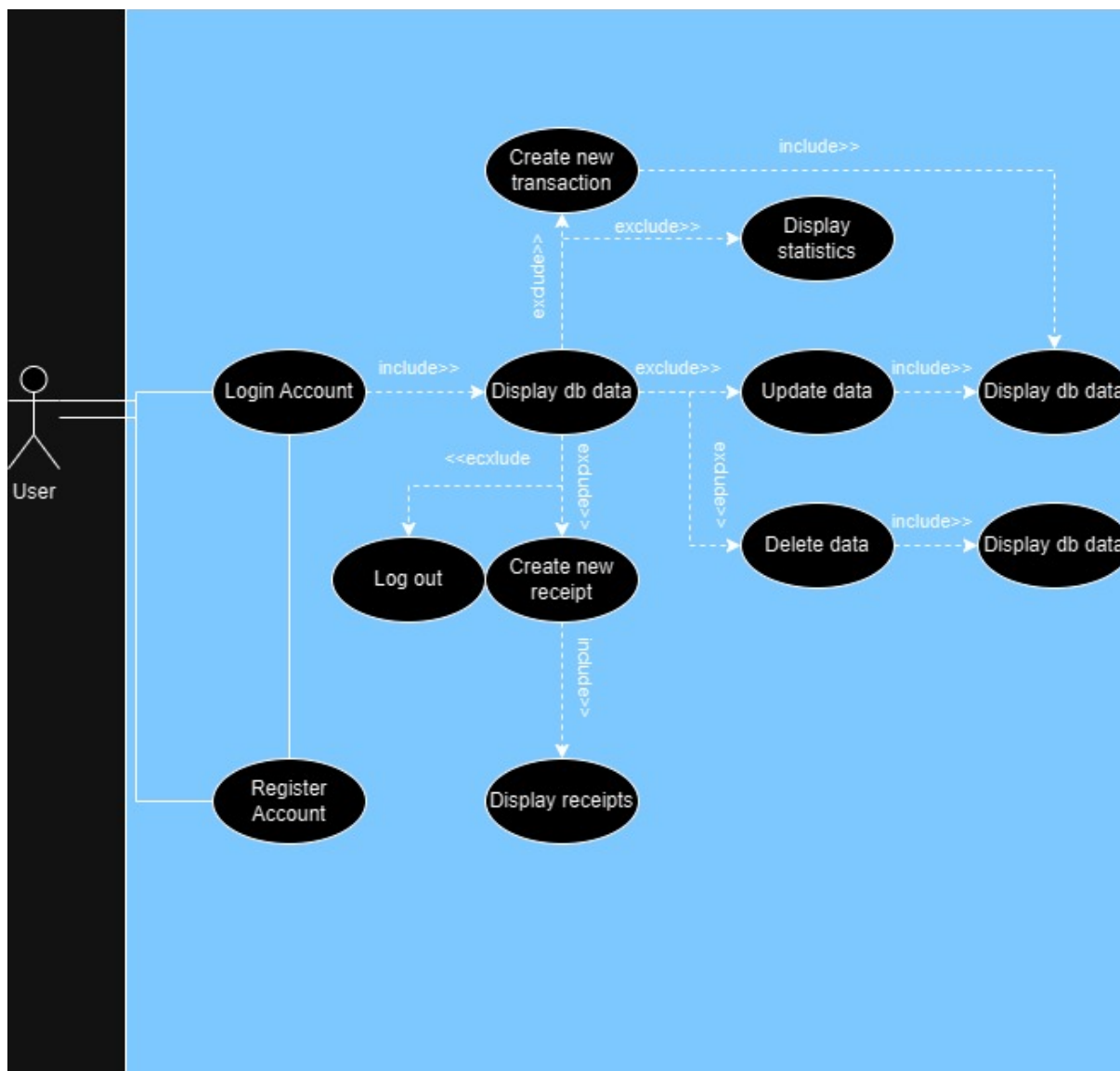
Most szeretném bemutatni az alkalmazás funkcióit. A kódomban a már tanult módszerekkel készítettem fejlesztői dokumentációt, ami megkönnyíti az esetleges továbbfejlesztést, a kód értelmezését. Az egyes funkciókhoz(ha elérhető) megmutatom a képernyő tervét és a végleges verzió kinézetét.

Részletesen végig fogok menni az alkalmazás összes funkcióján. Itt a már fentebb említett hasonlóságok és különbségek is kifejtésre kerülnek.

Röviden még ebben a bevezetésben beszélnék a technológiákról is, majd ezt hosszasan kifejtem egy későbbi fejezetben.

A fejlesztői környezetet az Android Studio biztosította, ezen belül Java nyelven történt a fejlesztés. Az adatbázis kezeléshez a Google Firebase felhőalapú szolgáltatását vettem igénybe. Ezen belül kétféle tárhelyet is használtam a Firestore Database-t illetve a Firebase Storage-et. Ezen kívül használtam még különböző Github-ra feltöltött dependency-ket, mint például a diagramok létrehozására vagy az adatok megjelenítésére.

3.1 Alkalmazás működésének szemléltetése



3.1. ábra: Use-case diagram

A szemléltetéshez egy use-case diagramot csináltam. Ezen azt láthatjuk, hogy a felhasználónak először két opciója van, bejelentkezés és regisztráció. Ha már regisztrált felhasználóról van szó akkor értelemszerűen be kell jelentkeznie, ellenkező esetben új felhasználóként regisztrálnia kell.

Bejelentkezés után, a tranzakcióit látja, erről az oldalról eléri az alapvető funkciókat. Képes új tranzakciót létrehozni, a diagramok megjelenítésére, az adott sor szerkesztésére vagy törlésére, a nyugták megjelenítésére, amely oldalon lehetőség van új nyugta hozzáadására. Emellett a főoldaltól ki tud jelentkezni a fiókjából.

3.2 Autentikáció

Ahogy már említettem, az alkalmazás egy felhasználói döntéssel indul, ahol választhat két opció, a bejelentkezés és a regisztráció között. Ezen az oldalon már adva van, hogyha regisztrált felhasználóval rendelkezik akkor a helyes mezőket kitöltve be tud jelentkezni.

3.2.1 Bejelentkezés

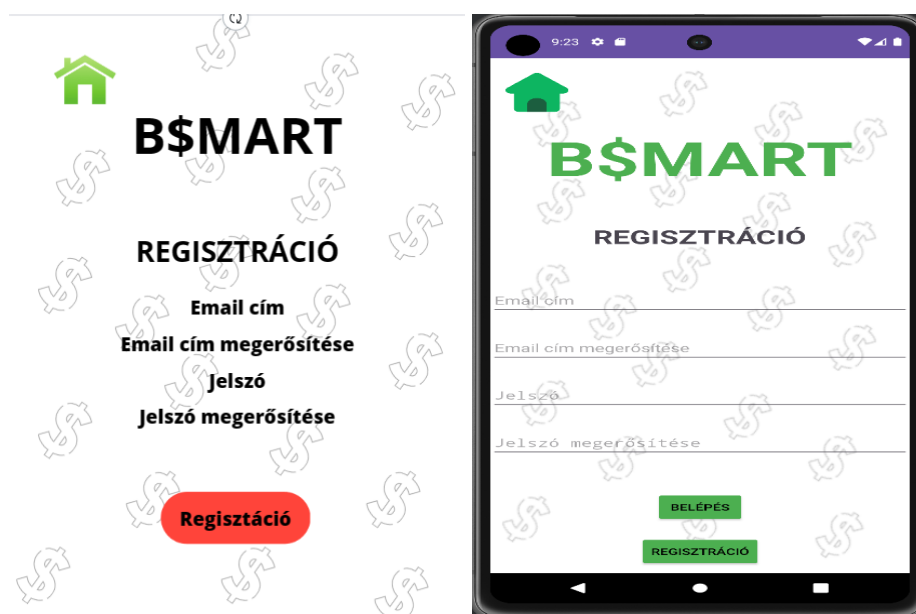
Az autentikációt a Firebase beépített authentication funkciójával valósítottam meg, erről részletesebben majd egy másik fejezetben beszélnék. Az alkalmazás a felhasználótól egy email címet és egy jelszót vár. Az xml beépített funkcióját használva az alkalmazás ellenőrzi, hogy a felhasználó jó típusú értéket adott-e meg az email mezőben. Ugyancsak az xml-t kihasználva a jelszó mezőbe írt karakterek csillagozva jelennek meg, ezzel is védve a felhasználót. A jelszónak minimum 6 karakter hosszúnak kell lennie. Ha bármelyik mező hibás, akkor arról egy Toast formájában értesíti a felhasználót az applikáció.



3.2.1. ábra: Login(képernyőterv vs megvalósított)

3.2.2 Regisztráció

A regisztráció gombra kattintva az app elnavigálja a felhasználót a regisztrációs oldalra, ahol 4 mezőt kell kitöltenie. Először is meg kell adnia az email címét, majd újra. Ha ezek nem egyeznek a felhasználó regisztrációja sikertelen lesz, erről értesül a felhasználó. A másik két mező is ugyanezen az elven alapul csak a jelszóval. Az oldalról továbbá elérjük a bejelentkezési oldalt, ezt a bal felső sarokban lévő ikonra való kattintással érhetjük el.



3.2.2. ábra: Regisztráció(képernyőterv vs megvalósított)

3.3 Főoldal(tranzakciók megjelenítése)

A sikeres bejelentkezés után eljutunk a főoldalra amely egyben a Firestore adatbázisban lévő tranzakciók lekérve és megjelenítve. Az alapötletem egy táblázatos megoldás volt, de ezen kicsit változtatnom kellett a fejlesztés során, a képernyőtervvel összehasonlítva láthatjuk a különbséget. Alapvetően azokat az adatokat jeleníti meg amelyeket a felhasználó töltött fel, a plusz attribútumokat, melyek ugyanúgy tárolva vannak(UserID, DocumentID, ImgURL) azokat nem.

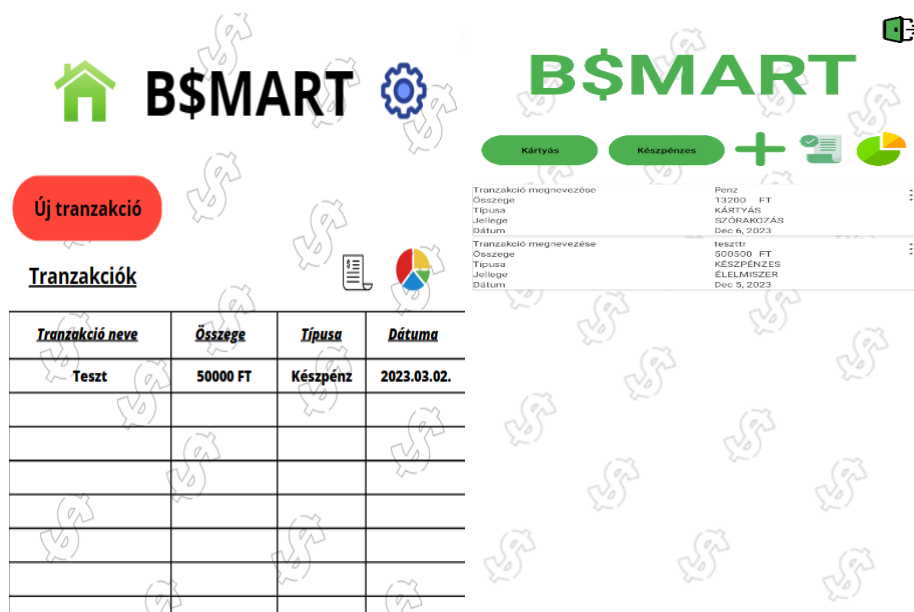
Minden sor egy dokumentumot jelöl a Firebase Database-ben, ha a három kis pontra kattint a felhasználó akkor lehetősége van az adott rekordot szerkeszteni vagy törölni, törlés esetén a megjelenítés automatikusan frissül.

Költségvetés-kezelő mobilalkalmazás

Erről az oldalról érhetjük el az új tranzakció létrehozása oldalt is, melyet egy kis ikon megnyomásával érhet el.

A nyugta megjelenítő lapot is innen érjük el, hasonlóan az előbbi metodikához.

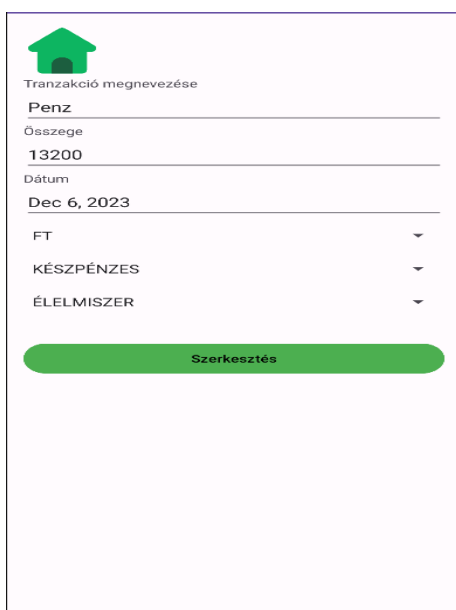
Ezek mellett még a statisztikai diagramokat illetve a felhasználó kijelentkeztetését hajthatjuk végre a főoldalon.



3.3. ábra: Főoldal(képernyőterv vs megvalósított)

3.3.1 Tranzakció szerkesztése

A szerkesztési oldalt a fentebb már említett három pontra való kattintással érhetjük el. Ezt követően az alkalmazás megnézi, hogy az adott rekordhoz milyen értékek tartoznak és ezt a szerkesztési oldalon a megfelelő beviteli mezőkbe beírja, hogy lássa a felhasználó, hogy milyen értékeket szeretne szerkeszteni. Ha mindent helyesen megadott, akkor a Szerkesztés gomb megnyomásával az adott rekord mind a felhőalapú adatbázisban, mind a főoldalon frissítésre kerül. A gomb megnyomása magával vonja a főoldalra való visszajutást is, ahol már a frissített adatok fogják várni a felhasználót.



The screenshot shows a mobile application interface for editing a transaction. At the top, there is a green house icon and the text "Tranzakció megnevezése". Below this, the word "Penz" is displayed. Further down, the "Összege" (Total) is set to "13200" and the "Dátum" (Date) is "Dec 6, 2023". There are three dropdown menus for "FT", "KÉSZPÉNZES", and "ÉLELMISZER", each with a downward arrow. At the bottom, there is a green button labeled "Szerkesztés" (Edit).

3.3.1. ábra: Tranzakció szerkesztése

3.4 Tranzakció létrehozása

A plusz jelre kattintva átkerül a felhasználó az új tranzakció létrehozása oldalra. Ezen az oldalon ki kell töltenie a megfelelő beviteli mezőket, lenyíló listákat, illetve a beépített dátumválasztó segítségével a dátumot. Mivel az összeg csak szám lehet, ezért itt történik egy ellenőrzés és ha a felhasználó nem megfelelő karaktert ír be akkor azt az alkalmazás jelzi a felhasználó felé. A tervezés során erre az oldalra tettem a nyugta létrehozását is de ezt különböző okok miatt, amit majd részletesebben kifejtek a megvalósításnál, egy külön oldalra helyeztem. Ha minden mezőt kitöltött a felhasználó, akkor a Tranzakció létrehozása gomb megnyomásával az alkalmazás feltölti és eltárolja az újonnan létrehozott dokumentumot a

Firebase Database-ben, ezzel párhuzamosan a felhasználó visszakérül a főoldalra, ahol már az újonnan feltöltött tranzakciója is a lekért rekordok között lesz látható.



3.4. ábra: Tranzakció létrehozása(képernyőterv vs megvalósított)

3.5 Nyugták megjelenítése

A kis “nyugta” ikonra kattintva elérhetjük a feltöltött nyugtáinkat. Ez egy nagyon egyszerű oldal, mivel maga a nyugta objektum is csak egy képet és a kép címét tartalmazza. Ezt azért csináltam meg így, mert úgy gondolom, hogy magán a nyugtán rajta van az az információ amelyre a felhasználónak szüksége van, ezt azért egészítettem ki egy cím adással, mert így a nyugta létrehozásánál mégis van egy lehetősége egy fajta megkülönböztetést alkalmazni, ezzel könnyebben azonosítani a keresett nyugtát a jövőben. Technikailag az oldal megjeleníti a nyugtákat és minden nyugta alá a hozzá tartozó címet. A bal felső ikon segítségével lehetőségünk van visszalépni a főoldalra.



3.5. ábra: Nyugták megjelenítése

3.5.1 Nyugta létrehozása

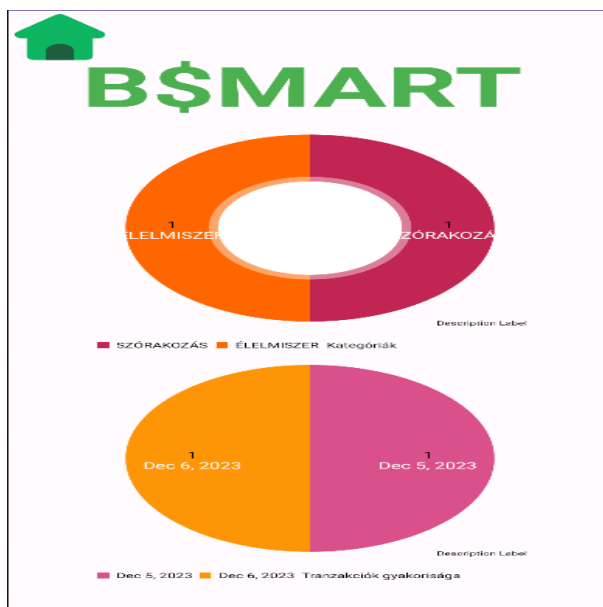
A nyugta listázása oldalról érhetjük el ezt a funkciót. Amilyen egyszerű a megjelenítés, olyan egyszerű a létrehozása is a nyugtáknak. A felhasználótól csupán egy, a galériából kiválasztott képet vár, ha ezt megtette, akkor a kiválasztott kép megjelenik, a szöveges beviteli mezőbe még meg kell adnia a kép címét, majd a Hozzáadás gombra kattintva az új adat felkerül a Firebase Storage-ba, és a navigációnak köszönhetően a felhasználó a nyugtak listázásánál találja magát ahol az újonnan feltöltött nyugta is látható lesz.

A screenshot of the form for creating a receipt in the B\$MART mobile application. At the top, there is a preview of the living room image. Below it, there is a label 'Blokk elnevezése' followed by an input field. Below that, there is a label 'Számra kiállító' followed by an input field. At the bottom, there are two green buttons: 'Fénykép feltöltése' and 'Blokk feltöltése'.

3.5.1. ábra: Nyugta létrehozása

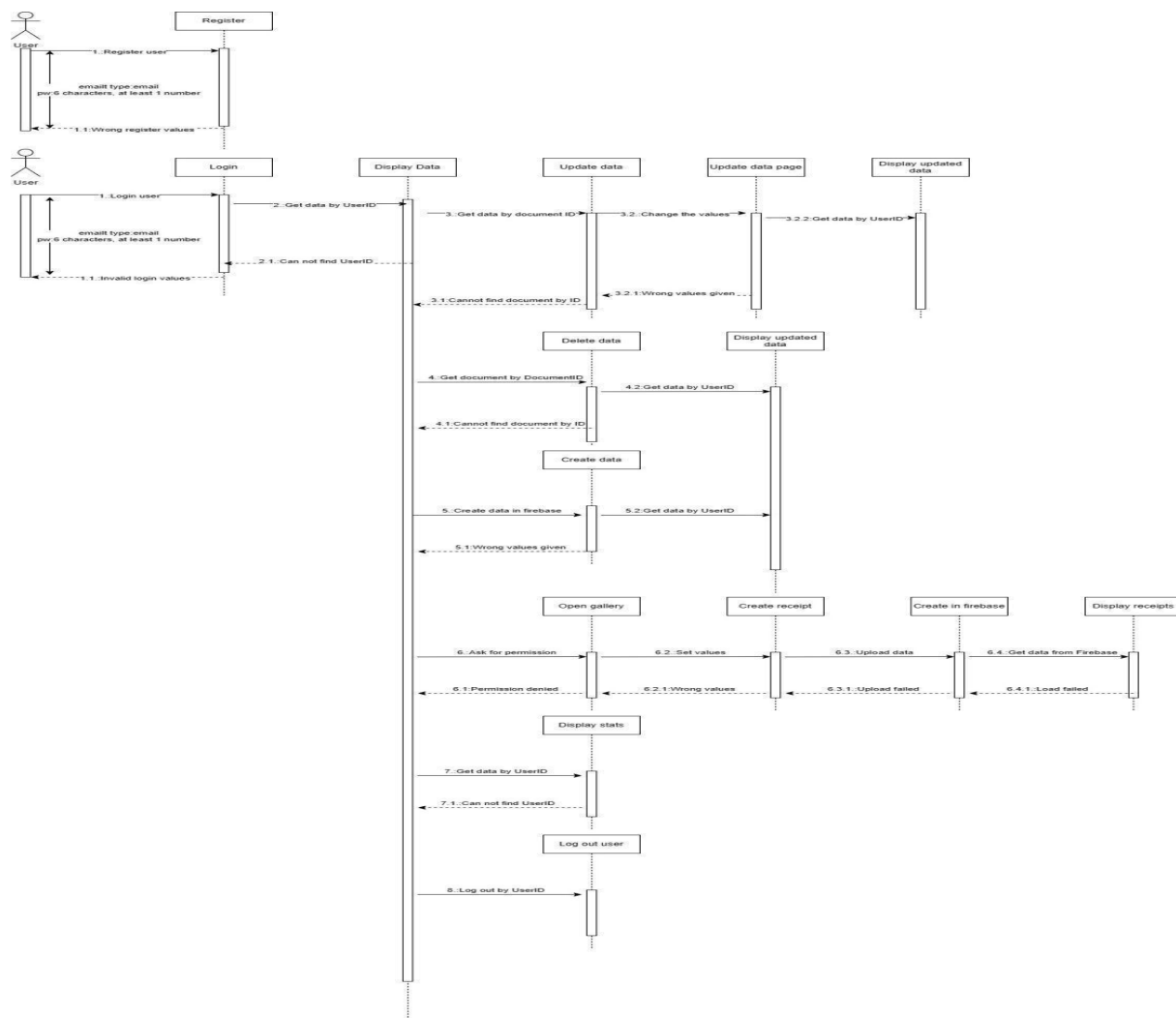
3.6 Diagram

Ezt a funkciót is a főoldalról érjük el. Ez csak egy megjelenítésre funkcionáló oldal, a felhasználótól semmilyen inputot nem vár. Két kör diagram kerül megjelenítésre.



3.6. ábra: Diagram

3.7 Egyes funkciók bemutatása szekvencia diagram segítségével



3.7. ábra: Szekvencia diagram

Az alábbi diagram szemléltetni szeretné a fentebb említett funkciókat tartalmazó osztályok hogyan is épülnek egymásra, illetve, hogy milyen üzenetváltások mennek végbe közöttük.

Szépen kivehető, hogy az alapvető autentikáción kívül, minden a főoldalként funkcionáló adatlekérés(Tr_Retrieve) osztályból érhető el különböző módokon. Az üzenetváltásokon láthatjuk, hogy gyakorlatilag ha a program sikeresen el tudja érni, az adott felhasználó által létrehozott különböző objektumokat, akkor ezeket probléma nélkül meg tudja jeleníteni, illetve a felhasználó ID használatával új objektumok létrehozására is képes. Mivel a főoldalból a felhasználó döntése alapján több irányba is mehet a navigáció, ezért ezeket egy életvonalra tettem, mivel nincs köztük alá-fölrendeltség.

4. fejezet

Felhasznált technológiák

Ebben a fejezetben rátérnék a feladat megvalósításához felhasznált és elengedhetetlen technológiák bemutatására. Szeretném bemutatni az adott technológiát először általánosan szemléltetni, majd specifikusan a projektre vonatkozóan.

4.1 Android Studio

Az Android platformra történő fejlesztéshez, a Google által fejlesztett Android Studio integrált fejlesztőkörnyezetét használtam. Ezt az IDE-t használhatjuk Linux-on, Windows-on és Mac OS-en egyaránt. A környezet biztosít a fejlesztő számára egyaránt frontend tervezésére alkalmas, illetve backend megvalósításra alkalmas lehetőségeket. Előbbit XML szerkesztő segítségével, utóbbinál több nyelven is lehetőségünk van a fejlesztésre. Választhatunk a Java illetve a Kotlin nyelvek közül, utóbbiról írnék röviden, előbbit részletesebben egy külön pontba szedve fogom ismertetni.

Fontosnak tartom bemutatni röviden azt a nyelvet is amit én személy szerint még nem használtam, de a jövőben jobban el szeretnék mélyülni benne. A Kotlin egy olyan nyílt forráskódú programozási nyelv, amely, nyilván szubjektív vélemények alapján, egy alternatívát nyújthat a fejlesztőknek. Ezen vélemények alapján, a kód rugalmasabb, könnyebb és olvashatóbb, mint a Java. Nyilván embere válogatja, én mindenképp adni fogok neki egy esélyt.

Az Android Studio továbbá biztosít a fejlesztőnek emulátoros megoldásokat is, amellyel még könnyebbé válhat a kód tesztelése, fejlesztése. Emellett fizikai eszközön is meg tudjuk jeleníteni az éppen fejlesztett programunkat. A felhasználói felület kreálásánál lehetőségünk van közvetlenül kódolni, illetve egy UI segítségével általánosabb módon, “drag and drop” technikával is létrehozni a kívánt felületet.

Összességében egy ma már alapvető erőforrásokkal rendelkező számítógép segítségével, könnyedén és kényelmesen lehet erre a platformra, ezen környezet segítségével alkalmazásokat fejleszteni. Továbbá rendelkezik minden olyan mai modern funkcióval, amely már szinte elengedhetetlen a fejlesztéshez.

4.1.1. Java

A Java egy objektumorientált programozási nyelv, rengetegféle területen használják, az egyik legnépszerűbb programozási nyelvek közé tartozik. A nyelv operációs rendszertől függetlenül futtatható, ehhez csak a futtatókörnyezetét szükséges telepíteni (JRE). A JVM felelős a kód fordításáért és futtatásáért. Ahhoz, hogy a kódot fel tudja dolgozni a JVM, a forráskódot bytecodeban kapja meg. Elsősorban az előismereteim miatt döntöttem a Java mellett.

4.1.2. XML

Ahogy már fentebb említettem, a fejlesztői környezet biztosítja a felhasználói felületek tervezését, megvalósítását is. Ehhez az XML strukturált adatleíró nyelvet veszi segítségül. Ennek a nyelvnek vannak sajátosságai, amelyekre szeretnék egy kicsit kitérni. Alapvetően elemeket tudunk létrehozni, melyeket tag-ek közé téve érhetünk el. Az elemeknek attribútumokat adhatunk, ezzel akár megkülönböztetve őket más elemektől. Rengetegféle módon testre szabhatjuk az elemeket, ezzel megkönnyítve a felhasználói felület létrehozását. Az elemeket egymásba tudjuk ágyazni, ezzel egy hierarchiát felállítva, segítve az adatok rendezettségét és struktúráltságát.

Beszélnék röviden a fejlesztői környezetnek köszönhető funkciókat az XML szerkesztőben. Többféle elrendezésű lapokat állíthatunk be, ezek mentén az elemeink elhelyezkedését tudjuk testreszabni.

Zárásul a rengetegféle elemtípus és ezek sokszínű testreszabása gyakorlatilag végtelen tárházat kínál azoknak akik szeretnek, igényes design-okat megvalósítani.

[4.2 MPAndroidChart](#)

Ezt a függőséget használtam azért, hogy a felhasználó egyes adatok felhasználásával különböző diagramokat láthasson. Ez egy nem beépített funkció az Android Studioban, egy fejlesztő által publikált megoldás, amit helyesen importálva használni lehet.

4.3 Glide

Ahogy az MPAndroidChart, a Glide is egy külsőleg fejlesztett nyílt forráskódú funkció, amit a nyugták megjelenítéséhez használtam. Ez segít abban, hogy a képeket meg lehessen jeleníteni a RecyclerView-ban. A Glide alkalmas még videók, illetve GIF-ek megjelenítésére is. Nagyon hasznosnak találom ha olyan feladatot kell megvalósítani ami képek megjelenítését végzi el.

4.4 Google Firebase

A Firebase egy felhőalapú szolgáltatás, amely nagyon megkönnyíti a mobil alkalmazások fejlesztését. Az én munkámat is nagyon megkönnyítette. Alapvetően a Firebase weboldalán létrehozhatunk többféle adatbázist is. Ezeket felhasználói guide segítségével könnyedén összekapcsolhatjuk a projektünkkel, sőt az Android Studio újításainak köszönhetően, már magában a fejlesztői környezetben is importálhatóak a szükséges függőségek.

Az adatbázisok nosql-re épülnek. Többféle adatbázis is van, ezek egy részét részletesebben is bemutatok. Van lehetőség úgynevezett real-time database-t is létrehozni, melynek előnye, hogy ha az adatbázisban frissülnek adatok, akkor azok azonnal frissülnek a kapcsolódott eszközökön is. Mivel az én alkalmazásom szempontjából ezt a funkciót nem tartottam annyira fontosnak, ezért én nem ezt a fajtát választottam.

Az egyik legnagyobb előnye a Firebase-nek, hogy szolgáltat a fejlesztő számára egy autentikációs megoldást. Az alapvető verzió email és jelszóval működik, de pár plusz függőség hozzáadásával implementálható a már jól ismert social médiás bejelentkezési opció is.

4.4.1. Firestore Database

A Firestore Database a Firebase adatbázisainak egyik fajtája. Alapvetően létre kell hozni egy kollekciót amelyben az objektumokat szeretnénk tárolni. Egy objektum létrehozásakor a kollekcióban egy új dokumentum jön létre, melyben az objektum értékei tárolódnak el. Tökéletes szövegek, számok tárolására. Minden dokumentum, tehát objektumnak a Firebase generál egy ID-t amit felhasználva műveleteket végezhetünk a dokumentumon.

4.4.2. Firebase Storage

Ennek a legfőbb funkciója, hogy képek, nagyobb fájlok, amelyeket a Firestore Databse már nem tud kezelni tárolni tudja. Megoldható továbbá, hogy az itt elérhető képek, videók, elérhetőek legyen egy kis csavarral a többi adatbázis szolgáltatásokban is, erről részletesen az implementáció bemutatásánál fogok beszélni.

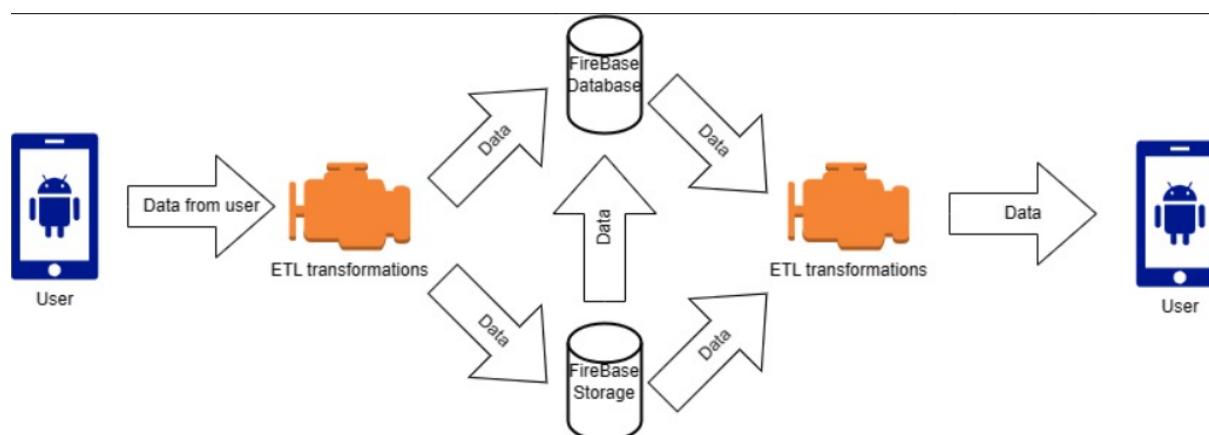
4.5 GitHub

A GitHub egy weben illetve már asztali alkalmazásként is elérhető verziókövető alkalmazás. A projekt átláthatóságát segíti, lehetőségünk van nyomon követni a fejlesztéseinket. Alapja, nevéből is adódik, a Git. Célszerű használni, ugyanis nagyon könnyű szinkronizálni a különböző fejlesztői környezetekkel, így egyszerre férhetünk hozzá a lokális verziókhoz és párhuzamosan a repository-ban tároltakhoz is. Minden változtatásnál úgynevezett commit-ot “csatolunk”, így szépen követhető, hogy mi történt az adott változtatásban.

Ipari környezetben is bevett és nagyon fontos dolog verziókövetők használata. Egy projekt során célszerű egyedi brancheket létrehozni, majd ha megbizonyosodtunk munkánkról, a merge funkcióval összeköthető egy vagy akár több branch-el is.

5. fejezet

Architektúra



5. ábra: Architektúra

Az alkalmazás elég egyszerűen és átláthatóan épül fel. A szerver oldalért és az adatbázissal való kommunikációért a Firebase felel. A már egyetemi kurzusokon is használatba vett eszköz, nagyon kényelmes és rugalmas munkát biztosít ilyesfajta projektekhez. Beépített autentikációja, adatbázis megoldásai nagyon megkönnyítik a fejlesztést. A Firebase többek között olyan szolgáltatást is nyújt melynek segítségével szabályok állíthatóak be az adatbázisunkra.

Ezt a két oldalt, a Java nyelven írt kód köti össze, melyben egyaránt hivatkozok az xml-ban készített objektumokra, illetve a Firebase adatbázisának beépített lekérdezéseire, dokumentumaira.

A program futtatását a fejlesztői környezet beépített emulátora segítségével hajtottam végre. Ehhez telepíteni kell az adott emulátor csomagját. Ezután a futtatás indításával a program elindítja az emulátort, feltelepíti rá az alkalmazást és azt meg is jeleníti, ezután úgy használhatjuk akár egy fizikai telefont.

6. fejezet

Activity-k és objektumosztályok

6.1 Activity-k

Az Activity-k felelnek gyakorlatilag mindenért. Ezek segítségével valósítható meg az adott xml-ek megjelenítése, ezek közötti navigáció. Általában minden activity-hez tartozik egy ún. layout fájl, ez az xml. Az activity-k java kiterjesztésűek. Ezekben a fájlokban alakítjuk ki a Firebase-zel való kommunikációt.

6.1.1 MainActivity

Ez az alkalmazás ún. “Launcher” activity-je, ami azt jelenti, hogy az alkalmazás indításakor az ehhez hozzárendelt layout-ot fogja megjeleníteni. Jelen esetben ez a bejelentkező felület.

6.1.2 RegisterActivity

A felhasználó ezen az oldalon tudja kitölteni a sikeres regisztrációhoz szükséges adatokat.

6.1.3 Tr_Retrieve

A Firebase kapcsolat létesítésével, az activity különböző funkciókkal lekéri és megjeleníti a dokumentumokat. Ez “alatt” egy ún. recyclerview típusú xml fájl van, amely lehetőséget biztosít, hogy egy adott objektum fajtát ugyanúgy jeleníthessünk meg. Azért alkalmaztam ezt, mert ha sok adatot kell megjeleníteni, akkor “görgetés” segítségével mindig ugyanannyit jelenít meg a kijelzőn a program.

6.1.4 Tr_Recycle_Adapter

Ez egy adapter az előbb említett recyclerview-hoz. Ehhez az az xml tartozik amelyben egy objektum megjelenítésének tulajdonságai vannak beállítva. Leegyszerűsítve ő felel azért,

hogy több objektum megjelenhessen a lekérő oldalon. Erről részletesebben egy későbbi fejezetben fogok írni.

6.1.5 Tr_Create

A bejelentkezés és regisztrációhoz hasonlóan ez az oldal is egy a felhasználótól várt értékekkel dolgozik. A megfelelő mezők kitöltése után az adott objektum feltöltésre kerül a Firestore Database-be, egy új dokumentumként. Ehhez szükséges volt létrehozni egy objektum osztályt is, erről részletesebben később.

6.1.6 Tr_Create_Receipt

Hasonlóan az előző activity-hez, ez is egy beviteli oldal, azonban itt nem csak szöveges mezőket kell kitöltenie a felhasználónak, hanem egy képet is fel kell töltenie a Firebase Storage-ba. Amint a feltöltés sikeres volt, a kép url-je másolásra kerül és eltárolódik egy Firestore Database kollekcióban is.

6.1.7 Tr_Update

A főoldalon lekért tranzakcióknál lehetőségünk van törlésre és szerkesztésre is. Ez az activity a szerkesztés funkciót valósítja meg. Kitöltve a már adott értékeket, melyeken a felhasználó változtathat, majd ezek frissülnek mind az adatbázisban mind a főoldalon.

6.1.8 Tr_Stats

Külső függőség importálásával létrehozhatóak különböző diagramok az Andorid Studioban. Ezek megjelenítésére szolgál ez az activity.

6.1.9 Tr_Receipt_Retrieve

Használatával a fentebb létrehozott számlákat tudjuk megjeleníteni, az előző megjelenítési módot használva, ugyancsak egy recyclerview segítségével.

6.1.10 Tr_Receipt_Adapter

A számla megjelenítő recyclerview adaptere.

6.1.11 DateComparator és RecComparator

Idősoros megjelenítésért felelős összehasonlító osztályok.

6.2 Objektumosztályok

Ahhoz, hogy a Firebase-ben dokumentumokat tudjunk feltölteni és tárolni, ehhez elengedhetetlen egy külön osztályt létrehozni az objektum típusnak. Az itt lévő konstruktor, illetve getterek, setterek segítségével érhetjük el mindkét oldalról az adott objektumot.

6.2.1 Tr_Object

A tranzakció tulajdonságait tartalmazza, értékei beállításával, majd ezek átadásával hozhatunk létre új dokumentumot a kollekcióban. A setterek segítségével pedig az objektum értékeinek lekérdezését végezzük el.

6.2.2 Tr_Receipt

A számla tulajdonságait tartalmazza, értékei beállításával, majd ezek átadásával hozhatunk létre új elemet a Firebase Storage-ban. A setterek segítségével pedig az objektum értékeinek lekérdezését végezzük el.

7. fejezet

Biztonság

Mivel a Firebase autentikációja már eleve ad egy egyfajta beépített biztonsági funkciót, illetve különböző kollekcióknál szabályokat is alkalmazhatunk, nem igazán foglalkoztam ezzel a ponttal.

Minden felhasználónak ugyanolyan jogai vannak, felhasználói fiók nélkül nem lát semmit a látogató. Az alkalmazás egy adat létrehozó és lekérő program, ezért nem láttam olyan egyértelmű jogokat amelyeket esetleg ki kellett volna osztanom vagy elvennem bizonyos esetekben. Az ilyen típusú alkalmazások kulcsfontosságú tulajdonsága mégis az és ezt a biztonsági funkciót bele kellett építenem az applikációba, hogy az adott felhasználó csakis azokat az adatokat láthassa, illetve kérhesse le amiket ő is töltött fel. Ezt szerver oldalon úgy valósítottam meg, hogy a Firebase generál egy userID-t amit egy objektumváltozóban eltároltam, majd az adatok lekérdezésénél ezt hasonlítom össze a kollekció dokumentumainak egyes userID-ivel. Ezzel orvosolva azt a problémát, hogy a felhasználók egymás adatait is láthassák. Emellett a Firebase konzoljában létre lehet hozni, ún. szabályokat, itt beállítottam azt, hogy csak a saját adatait láthassa az adott felhasználó.

```
match /users/{userId} {  
  allow read, update, delete: if request.auth != null && request.auth.uid == userId;  
  allow create: if request.auth != null;  
}
```

7. ábra: Firebase rule

8. fejezet

Adatmodell

8.1 Tr_Object

A tranzakciók tulajdonságait tárolom ebben az osztályban. Alapvetően alapjául azok a mezők szolgálnak amelyek a tranzakció létrehozása oldalon kitöltendőek. Emellett eltárolom benne a Firebase által generált userID-t és a documentID-t

- **key:**
Ez tartalmazza a Firebase által automatikusan létrehozott, minden dokumentum egyedi azonosítóját. Ezt string-ként tárolom, ennek segítségével tudom az adott tranzakciót törölni, illetve szerkeszteni.
- **userID:**
Ez az argumentum is generálva van a felhő alapú adatbázisból, minden felhasználónak van egy egyedi azonosítója. Nagyon fontos az alkalmazás és annak biztonsága szempontjából, ugyanis ennek a segítségével valósítottam meg azt, hogy az adott felhasználó csak a saját tranzakcióit lássa. Ez is string-ként tárolódik.
- **tr_name:**
Adott tranzakció neve, string.
- **tr_amount:**
Adott tranzakció összege, ez int típusú.
- **tr_date:**
A tranzakció dátuma, string-ként tárolódik, de ezt megelőzi egy típuskonverzió, ugyanis az xml-ből dátum típust kap, de, hogy a Firebase is kezelni tudja ezért átalakítom string-gé.
- **enumok:**
Egyes adatok bevitelére egyszerűbbnek láttam, hogy egy megadott listából választhasson a felhasználó. Nyilván így véges opciói vannak, de ezt bármikor lehet bővíteni, igény szerint.
 - **Currency:**
A tranzakció pénznemét választhatjuk ki. Értékei: FT, EUR, USD, EGYÉB.
 - **Type:**
A fizetés módját adhatjuk meg vele. Értékei: Kártyás, Készpénzes

- **Group:**
Megadott kategóriába sorolhatjuk a tranzakció jellegét az opciókból választva.
Értékei: ÉLELMISZER, SZÓRAKOZÁS, BEVÉTEL, EGYÉB_KIADÁS

8.2 Tr_Receipt

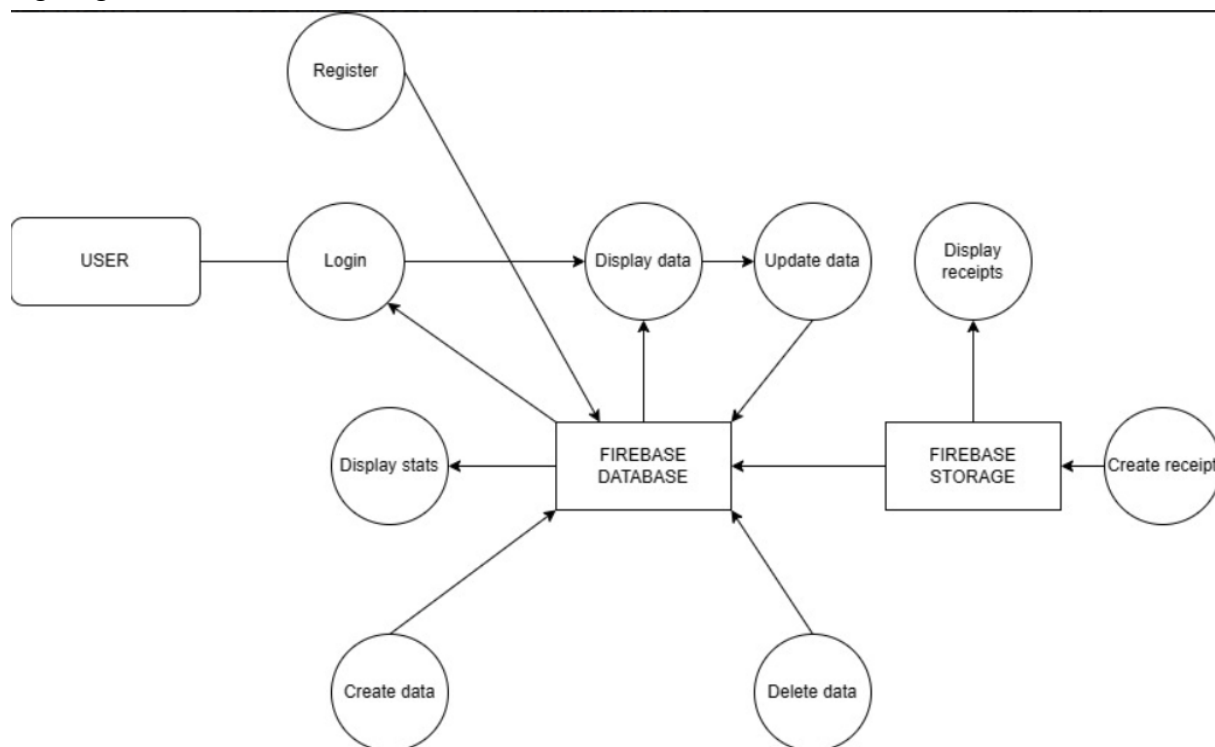
Ebben az osztályban kerülnek tárolásra a számla adatai.

- **userID:**
Megegyezik az Tr_Object argumentumával.
- **tr_rec_name:**
A számla nevét adhatjuk meg neki. String formátumú.
- **tr_rec_supplier:**
A számlát kiállító megnevezése. String.
- **tr_red_url:**
Ez is string típusú, még hozzá azért, hogy Storage-ba feltöltött kép url-jét el tudjuk tárolni Firestore Database-ben is.
- **tr_rec_date:**
Ezt az adatot a Firebase Storage metaadatai közül nyeri ki az applikáció, a hozzáadás dátumát kéri le. String típusú.

9. fejezet

A rendszer magasszintű folyamatai, működése

Az alábbiakban a rendszer folyamatait szeretném bemutatni egy adatfolyam diagram segítségével.



9. ábra: Adatfolyam diagram

Az alábbi diagrammal szeretném szemléltetni az alkalmazás folyamatait. Láthatjuk, hogy két nagy kapcsolódási pont van, az egyik a Firebase Database, ahol dokumentumok formájában vannak eltárolva a tranzakciók, mivel a funkciók nagy része a tranzakciókhoz kapcsolódik, ezért túlnyomó többségben ehhez az adatbázishoz kapcsolódnak nyilak.

Látható, hogy vannak olyan folyamatok amik egymást követően, vagy egymásból adódóan történnek. Ilyen például az “Update data”, ami a “Display data”-ból érhető el, logikailag így láttam helyesnek a jelölését, azonban gyakorlatilag ő is a az adatbázisból kapja az adatokat.

Megfigyelhető az is, hogy a “User”-nek mindennek előtt a “Login” funkción kell továbblépnie, hogy elérje az alkalmazás további funkcióit.

Nagyon fontos az, hogy a két adatbázis között is történik kommunikáció. Ez egy irányú, ami azt jelenti, hogy a Database csak fogad adatokat a Storage-ból. Amikor a felhasználó egy számlát szeretne létrehozni, akkor a tranzakciókkal ellentétben nem a Database-be tölti fel az adatot, mivel az, a méretből adódó limitációk miatt nem alkalmas rá. Azért, hogy ezt a problémát orvosoljam, létrehoztam egy Storage típusú adatbázist is, amely a nagy méretű, általában médiafájlok tárolására lett kitalálva(gondoljunk egy social media felületre, amely rengeteg ilyen típusú fájlt tartalmaz, erre tökéletes). Mivel a Storage főleg

Költségvetés-kezelő mobilalkalmazás

erre alkalmas, a kiegészítő információit a számlának továbbra is a Database-ben szerettem volna eltárolni. Ezt úgy oldottam meg, hogy miután a feltöltés a Storage-ba sikeres volt, a kiegészítő adatokat, illetve a már Storage-ban megtalálható kép url-jét áttöltöm a Database-be, ahol gyakorlatilag egy hivatkozás, illetve annak kvázi leírása fog eltárolódni.

10.fejezet

Fontosabb kód részletek, ezek ismertetése

10.1 Autentikáció

10.1.1 Regisztráció

```
mAuth.createUserWithEmailAndPassword(email, password).addOnCompleteListener( activity: this, new OnCompleteListener<AuthResult>() {
    @Override
    public void onComplete(@NonNull Task<AuthResult> task) {
        if(task.isSuccessful()) {
            Intent intent = new Intent( packageContext: RegisterActivity.this, MainActivity.class);
            startActivity(intent);
            Log.d(LOG_TAG, msg: "User created successfully");
        } else {
            Log.d(LOG_TAG, msg: "User was't created successfully:", task.getException());
            Toast.makeText( context: RegisterActivity.this, text: "User wasn't created successfully:", Toast.LENGTH_LONG).show();
        }
    }
});
```

10.1.1. ábra: Regisztráció(code)

Miután egy egyszerű if-else if logikai vizsgálattal meggyőződünk arról, hogy a felhasználó helyesen töltötte ki a regisztrációs felület, akkor a Firebase “createUserWithEmailAndPassword” funkciója hívódik meg, ahol az adott értékekkel létrehozza a felhasználót az adatbázisban. Egyuttal a felhasználó vissza kerül a bejelentkezési felületre, ahol már be tud lépni fiókjába.

10.1.2 Bejelentkezés

```
mAuth.signInWithEmailAndPassword(email, password).addOnCompleteListener( activity: this, new OnCompleteListener<AuthResult>() {
    @Override
    public void onComplete(@NonNull Task<AuthResult> task) {
        if(task.isSuccessful()){
            Create(view);
            Log.d(LOG_TAG, msg: "Login done!");
        } else {
            Toast.makeText( context: MainActivity.this, text: "Invalid email or password!", Toast.LENGTH_LONG).show();
        }
    }
});
```

10.1.2. ábra: Bejelentkezés(code)

Hasonlóan a regisztrációhoz, a bejelentkezés is a Firebase egyik autentikációs funkcióját hívja meg, mégpedig a “signInWithEmailAndPassword”-ot. Ha létező felhasználót adott meg, akkor sikeresen belépteti a rendszer az alkalmazás főoldalára. Ha az adatokkal amiket

megadott, nem talál felhasználót, akkor pedig egy Toast-tal jelzi a felhasználó felé, hogy probléma adódott.

10.2 Tranzakció létrehozása

```
db.collection(collectionName).collectionReference()
    .add(myObject).addOnSuccessListener(new OnSuccessListener<DocumentReference>() {
        @Override
        public void onSuccess(DocumentReference documentReference) {
            String ref = documentReference.getId();
            documentReference.update(field: "key", ref);
            Intent intent = new Intent(packageContext: Tr_Create.this, Tr_Retrieve.class);
            startActivity(intent);
        }
    })
    .addOnFailureListener(new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            Toast.makeText(context: Tr_Create.this, text: "Object wasn't created successfully!", Toast.LENGTH_LONG).show();
        }
    });
```

10.2. ábra: Tranzakció létrehozása(code)

Miután a felhasználó kitölti a létrehozó mezőket, ezek értékét elkérve, létrehozunk egy referenciát az adott kollekcióra a Firebase-ből. Ennek az “add” metódusát felhasználva, hozzáadjuk a tranzakció objektumunkat. Ha sikeres a művelet, akkor a létrehozott dokumentumnak a Firebase generál egy azonosítót, ezt az azonosítót eltárolom a “key” nevű argumentumba, ez később a szerkesztés és a törlés funkcióknál lesz fontos. A sikeres létrehozás után, a felhasználó visszakérül a főoldalra, ahol már látni is fogja a feltöltött tranzakciót. Itt is egy Toast üzenettel jelezzük, hogyha a feltöltés valamiért nem sikerült.

Fejlesztésem során az első nagyobb probléma a dátumok kezelése volt, végül a megoldás az lett, hogy a dátumválasztó ablakból megkapott értékből először egy “calendar” típusú változót generáltam, amit pedig String-gé formázva az adatbázishoz, illetve az objektumhoz lehet adni.

```
Calendar calendar = Calendar.getInstance();
calendar.set(datePicker.getYear(), datePicker.getMonth(), datePicker.getDayOfMonth());

// Convert the selected date to a Date object
Date selectedDate = calendar.getTime();

String tr_date = DateFormat.getDateInstance().format(selectedDate);
```

10.2. ábra: Dátum kezelés(code)

10.3 Bejelentkezett felhasználó tranzakcióinak megjelenítése

10.3.1 Tranzakció lekérő

Maga a megvalósítás nagyon hasonló magához a létrehozáshoz, először is le kell kérni, hogy éppen melyik felhasználó van bejelentkezve, neki el kell tárolni az egyedi azonosítóját. Ha ez megvan, akkor létrehozunk itt is egy referenciát a kollekcióra, ahonnan az adatokat szeretnénk lekérni. Miután ez megvan, egy feltételhez kell kötni, hogy csak azokat a kollekciókat vegye figyelembe ahol, a felhasználói azonosító megegyezik a kollekcióban, a létrehozás alkalmával eltárolt userID-vel. Ezzel biztosítva azt, hogy csak az adott felhasználó adatai jelenjenek majd meg.

Ebben az esetben meghívásra kerül a Firebase “get” metódusa, amiben az adott dokumentum adatait eltárolhatjuk a “Tr_Object” típusú objektumunkban, majd ezt az Adapter osztályunk “addData” metódusának adjuk át.

```
private void loadDataFromFirestore(String userID) {
    FirebaseFirestore db = FirebaseFirestore.getInstance();
    db.collection( collectionPath: "tr10") CollectionReference
        .whereEqualTo( field: "userID", userID) Query
        .get() Task<QuerySnapshot>
        .addOnSuccessListener(queryDocumentSnapshots -> {
            for (DocumentSnapshot document : queryDocumentSnapshots) {
                Tr_Object object = document.toObject(Tr_Object.class);
                if (object.getKey() != null){
                    adapter.addData(object);
                }
            }
            adapter.sort();
        })
        .addOnFailureListener(e -> {
            Log.d(LOG_TAG, msg: "Something is not working");
            // Handle any errors while fetching data
        });
}
```

10.3.1. ábra: Tranzakció lekérő(code)

10.3.2 Adapter

A recyclerview adapterében, először is definiáltam egy olyan listát, amely tranzakciós objektumokat tartalmaz. Az előzőekben említett “addData” metódus feltölti ezt a listát a kapott tranzakciókkal az adatbázisból. Majd az előre definiált TextViewk-ba a getterek és setterek segítségével betöltjük az adott értékeket.

```
public void addData(Tr_Object newData) {
    TrList.add(newData);
    notifyDataSetChanged(); // Notify RecyclerView about the data change
}
```

10.3.2. ábra: Adapter-addData(code)

```
holder.nameTextView.setText(object.getTr_name());
```

10.3.2. ábra: Adapter-setText(code)

10.3.3 Comparator

Ahhoz, hogy ne a Firebase default rendezésével jelenjenek meg a tranzakciók, kicsit kutakodnom kellett, ugyanis először magát a dokumentumon belüli kollekciókat próbáltam rendezni a Firebase beépített funkcióival, de ezek nem működtek. Ezután arra gondoltam, hogy magát a listát fogom rendezni, de ez objektumok esetén nem triviális. Ekkor találtam rá arra, hogy a listákat amelyek objektumokat tartalmaznak ún. Comparator segítségével lehet rendezni. Ehhez létre kell hozni egy Comparator osztályt, melyben a listában szereplő objektumot hasonlítja össze egy másikkal a megadott argumentumok mentén. Ez alapján sikerült megvalósítani úgy, hogy a megjelenítés során az adatok időrendi sorrendben legyenek.

```
public class DateComparator implements Comparator<Tr_Object>{
    public int compare(Tr_Object o1, Tr_Object o2)
    {
        SimpleDateFormat dateFormat = new SimpleDateFormat( pattern: "MMM d, yyyy");
        try {
            Date d1 = dateFormat.parse(o1.getTr_date());
            Date d2 = dateFormat.parse(o2.getTr_date());
            return d2.compareTo(d1); // Compare in descending order
        } catch (ParseException e) {
            e.printStackTrace();
            return 0; // Default comparison if parsing fails
        }
    }
}
```

10.3.3. ábra: Comparator(code)

10.3.4 Szűrő

Szűrő megvalósítására ötlet hiányából adódóan, egy elég fapados megvalósítást alkalmaztam, de talán egyszerűségében rejlik szépsége és ahsznossága. Alapvetően egy gombbal értem el azt, hogy az oldalon csak kártyás, csak készpénzes vagy az összes tranzakciót jelenítse meg. A gombra hivatkozásokat helyeztem, melyek állapotát vizsgálva változik a szövegük, illetve funkciójuk. Szűrésnél a nem “jó” értékeket kitörlöm a listából, majd ha újra az összeset szeretné látni a felhasználó akkor ezeket újra betöltöm a listába.

```
public void filter(View view) {
    final int status =(Integer) view.getTag();
    type = "KÁRTYÁS";
    if (status == 1){
        adapter.deleteData();
        filter(userID, type);
        filterbtn.setText("Összes");
        view.setTag(0);
    } else if (status == 0) {
        adapter.deleteData();
        loadDataFromFirestore(userID);
        filterbtn.setText("Kártyás");
        view.setTag(1);
    }
}
```

10.3.4. ábra: Szűrő(code)

10.3.5 Szerkesztés/törlés

Köszönhetően annak, hogy a dokumentumokban eltárolásra kerültek a dokumentum azonosítók, így ezekre könnyedén lehet hivatkozni és meghívni azok beépített “update” illetve “delete” funkcióit. Mivel mindkét funkciót egy menüből érjük el, ezért mindenképp ezekre az eseményekre egy elágazást kellett létrehozni az adapterben.

A törlés funkcióval kezdeném, egyszerűsége miatt. Miután a helyes menü id azonosítása sikerült, bele lépünk a törlés ágba, ahol elkérjük az adott objektum kulcs argumentumát, amit átadva a “delete” metódusnak, a dokumentum törlésre kerül. Ha ez sikeres, akkor a listából is töröljük, így elérve azt, hogy automatikusan eltűnjön az oldalról is.

```
documentReference = db.collection( collectionPath: "tr10").document(key);
documentReference.delete().addOnSuccessListener(suc ->
{
    Toast.makeText(context, text: "Record is removed", Toast.LENGTH_SHORT).show();
    notifyItemRemoved(position);
    TrList.remove(object);
}
```

10.3.5. ábra: Szerkesztés/törlés(code)

A szerkesztés egy sokkal bonyolultabb, főleg azért, mert a szerkesztő oldalra be kell tölteni a szerkeszteni kívánt adatokat. Ezt az “Intent” változó “putExtra” metódusával érhetjük el, meglehet adni egy kulcsszót, illetve a vizsgált objektumot, majd a Tr_Update osztályban a “getSerializableExtra” funkcióval egyeztetve, átadhatjuk egy változóba a vizsgált objektumot. Ezt felhasználva a getterek segítségével az EditText mezőket feltölthetjük az objektum helyes argumentumaival.

```
Intent intent = new Intent(context,Tr_Update.class);
intent.putExtra( name: "SZERKESZTÉS",object);
```

10.3.5. ábra: putExtre(code)

```
tr_edit = (Tr_Object) getIntent().getSerializableExtra( name: "SZERKESZTÉS");
```

10.3.5. ábra: getSerializableExtre(code)

Miután átadtuk az értékeket, a felhasználó szerkeszti a kívánt mezőket és ezek értékét felhasználva, illetve már a törlésnél is használt “key” argumentumot használva, meghívjuk a referenciára az “update” metódust az új értékekkel.

```
documentReference.update( field: "tr_name",edit_name.getText().toString(), ...moreFieldsAndValues: "tr_amount",
    Integer.parseInt(edit_amount.getText().toString()),"tr_date",edit_date.getText().toString(),"currency",
    selectedCurrency,"type",selectedType,"group",selectedGroup).addOnSuccessListener(suc ->
{
```

10.3.5. ábra: update(code)

10.4 Diagram

A projekt kitalálása során mindenképpen szerettem volna valamiféle statisztikát készíteni a tranzakciókból. A fejlesztés során amikor már ehhez a ponthoz értem felmértem a lehetőségeimet, és arra jutottam, hogy egy olyan tool-t fogok használni ami már elég elterjedt, illetve annyira nem is bonyolult implementálni, látványos. Tehát a választásom az MPAndroidCharts-ra esett.

Nagyon nagy pozitívumként kell felhoznom, hogy nem kell az XML-ben gyakorlatilag semmit sem csinálni, csak megadni, hogy milyen diagramtípust szeretnél implementálni.

Maga a diagram megvalósítása egy külön activity-ben található. Az adatok a már jól ismert módon kerülnek az applikáció “kezébe”, annyi kiegészítéssel, hogy a két diagram megfelelő szűréseinek felelnek meg.

Mivel ez egy elég hosszú kódrészlet, illetve nem is bonyolult, csak sok egymásra épülő “modul”-ból áll, ezért csak egy részletet mutatnék be belőle.

```
Map<String, Integer> typeCountMap = new HashMap<>();
for (String type : typeList) {
    if (typeCountMap.containsKey(type)) {
        int count = typeCountMap.get(type);
        typeCountMap.put(type, v: count + 1);
    } else {
        typeCountMap.put(type, v: 1);
    }
}
```

10.4. Diagram-részlet(code)

Itt azt láthatjuk, hogy a typeList tartalma kerül feldolgozásra, az egyes “type”-ok előfordulását vizsgálja, ez fog megjelenni a diagramon.

10.5 Számla létrehozása

A fő szempont ami megkülönbözteti a számla feltöltést a sima tranzakciók létrehozásától, hogy maga a kép a Storage-ba kerül, illetve a dátumot is a Storage-ból nyerjük ki.

Először is bemutatnám azt, hogy hogyan valósítottam meg a galéria megnyitását, illetve a fénykép kiválasztását. A galéria megnyitását egy intent segítségével érhetjük el. Ennek különböző funkcióit meghívva megnyitásra kerül a galéria és választhatunk egy képet. Ezt a képet átadom egy imageView-nak, így a kiválasztott kép meg is jelenik a képernyőn, illetve majd ez fog segíteni nekünk abban is, hogy feltölthessük az adatbázisba.

```
private void selectImage() {

    Intent intent = new Intent();
    intent.setType("image/*");
    intent.setAction(Intent.ACTION_GET_CONTENT);
    startActivityForResult(intent, requestCode: 100);

}
```

10.5. ábra: Galériából választás(code)

```
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if (requestCode == 100 && data != null && data.getData() != null){
        imageUrl = data.getData();
        capturedImageView.setImageURI(imageUri);
    }
}
```

10.5. ábra: Megjelenítés(code)

Magáról a fájl feltöltése az ugyanúgy történik, mint a tranzakciónál, csak itt maga a nyers kép az a Storage-be töltődik be. Ahhoz, hogy ezt egy hivatkozás formájában át tudjam tölteni a Database-be, először is le kellett kérnem a kép Url-jét majd ezt Stringgé alakítva már fel tudtam tölteni a kiegészítő információkkal(név, kiállító) a Database-be.

```
storageReference.getDownloadUrl().addOnSuccessListener(new OnSuccessListener<Uri>() {
    @Override
    public void onSuccess(Uri uri) {
        Tr_Receipt receipt = new Tr_Receipt(userID, fileName, supplier, formattedDate, uri.toString());
        db.collection(collectionPath: "tr_receipt") // Replace "images" with your desired collection name
            .add(receipt)
    }
})
```

10.5. ábra: uri.toString(code)

A másik izgalmasabb dolog még az volt, hogy szerettem volna az időpontot is eltárolni, de most valahogy máshogy, automatikusan. Ehhez sikerült lekérnem a Storage-ból az adott kép ún. “metadata”-i közül a létrehozás időpontját. Ezen egy kicsit formázni kellett, de egy változóban eltárolva ezt is ugyanúgy át tudtam tölteni a Firebase Database-be.

```
storageReference.getMetadata().addOnSuccessListener(new OnSuccessListener<StorageMetadata>() {
    @Override
    public void onSuccess(StorageMetadata storageMetadata) {

        Date creationDate = new Date(storageMetadata.getCreationTimeMillis());
        SimpleDateFormat sdf = new SimpleDateFormat(pattern: "yyyy-MM-dd", Locale.getDefault());
        formattedDate = sdf.format(creationDate); // Format the date
    }
})
```

10.5. ábra: Dátum formázás(code)

10.6 Számla megjelenítése

Mivel a számlák összes tulajdonsága el lett tárolva a Database-ben ezért, a lekérésében semmi újdonáság nincs.

11. fejezet

Tapasztalatok továbbfejlesztési lehetőségek

11.1 Nehézségek

A fejlesztés során adódtak folyamatosan kisebb, nagyobb akadályok de ezek túlnyomó részét sikerült megoldani, vagy egy új megoldást találni rá.

Az első probléma a dátumokkal volt. Először úgy indultam neki, hogy a felhasználó begépelhette a dátumot, de ezzel nem voltam megelégedve, ez nem egy elegáns megoldás. Ezért elkezdtem utánanézni, hogy hogyan lehetne először is dátum formátumú beviteli mezőt létrehozni, majd ezt hogyan lehet eltárolni az adatbázisban. Kezdetben megnéztem, hogy milyen megoldásai vannak az XML-nek, ekkor találtam rá a “DatePicker” tag-re. Örültem, hogy van egy ilyen opció, de elég igénytelenül néz ki, ezért megpróbáltam a formátumán változtatni, de sajnos ezt a feladatot nem tudtam megugrani. Miután már volt egy dátumom, szerettem volna feltölteni az adatbázisba, nagyon sok próbálkozás és szenvedés után, sem sikerült dátum formátumban eltárolnom, ezért a feltöltés előtt egy string-gé konvertálom, így egy szöveges értéként kerül tárolásra.

Ezen kívül voltak kisebb problémák, de ezek általában valami figyelmetlenségből adódtak, szerencsére a projekt előrehaladtával, már profin értelmeztem a dobott hibaüzeneteket, így egyre könnyebb volt a hibákat is megkeresni, azokra megoldást találni.

Néhány akadály a frontend oldalon is adódott, sokszor a beépített XML tervező felület, maga a kód és a buildelés után futó alkalmazás három különféle verziót mutatott. Emellett, sajnos erőforrás problémáim is voltak, mivel emulátorral dolgoztam ezért nagyon sokszor kifagyott, leállt, stb. Mivel ez nem a kódból adódó probléma volt, újraindításon kívül nem tudtam sokkal többet tenni érte.

11.2 Tapasztalatok, továbbfejlesztési lehetőségek

Szakdolgozatom elkészítése során rengeteg új dolgot próbáltam ki. Mivel ez egy önálló ötletű projekt, ezért nyilván olyan területet választottam ami az egyetemi oktatásom alatt felkeltette az érdeklődésem. Ezt, úgy gondolom, hogy a fejlesztés még jobban elmélyítette. Nem tartom magam közel sem jó programozónak, de ez az alkalmazás segített, hogy olykor kiléphessek a komfortzónából és új dolgokat próbáljak ki. Hasznos tudásra és még inkább tapasztalatra tettem szert.

Zárásként szeretnék még beszélni arról, hogy milyen irányba lehetne bővíteni az alkalmazásom:

- Igényesebb, profibb GUI
- Biztonsági bővítés
- Adatokból egy fajta riport készítő funkció

Költségvetés-kezelő mobilalkalmazás

- Fénykép feltöltése mellett, valamilyen OCR megoldás
- Részletesebb szűrési funkció
- Autentikáció bővítése

Irodalomjegyzék

1. Java: [Java | Oracle](#) (Utolsó megtekintés: 2023.12.13.)
2. Java guide: [Introduction to Java - GeeksforGeeks](#) (Utolsó megtekintés: 2023.12.13.)
3. Android Studio: [Meet Android Studio | Android Developers](#) (Utolsó megtekintés: 2023.12.13.)
4. Firebase: [Firebase | Google's Mobile and Web App Development Platform](#) (Utolsó megtekintés: 2023.12.13.)
5. MPAndroidChart: [GitHub - PhilJay/MPAndroidChart: A powerful !\[\]\(746d018fdf6ab02bf5fb7681133e8b29_img.jpg\) Android chart view / graph view library, supporting line- bar- pie- radar- bubble- and candlestick charts as well as scaling, panning and animations.](#) (Utolsó megtekintés: 2023.12.13.)
6. Glide: [GitHub - bumptech/glide: An image loading and caching library for Android focused on smooth scrolling](#) (Utolsó megtekintés: 2023.12.13.)
7. Draw.io: [draw.io \(diagrams.net\)](#) (Utolsó megtekintés: 2023.12.13.)
8. Comparator: [java - Sorting a collection of objects - Stack Overflow](#) (Utolsó megtekintés: 2023.12.13.)
9. Git: [Git \(git-scm.com\)](#) (Utolsó megtekintés: 2023.12.13.)
10. GitHub: [GitHub: Let's build from here · GitHub](#) (Utolsó megtekintés: 2023.12.13.)

Nyilatkozat

Alulírott Taskovics Lőrinc Károly Gazdaságinformatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztés Tanszékén készítettem, BSc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat / diplomamunkámat a Szegedi Tudományegyetem Diplomamunka Repozitóriumban tárolja.

Dátum 2023.12.15.

Aláírás

Köszönetnyilvánítás

Szeretnék köszönetet mondani Dr. Bilicki Vilmos tanár úrnak az elmúlt egy évben nyújtott segítségéért, tanácsaiért, útmutatásaiért. Nagyon megkönnyítette a munkánkat a tanár úr felkészültsége.