Szegedi Tudományegyetem

Informatikai Intézet

SZAKDOLGOZAT

Horváth Máté

2022

Szegedi Tudományegyetem
Informatikai Intézet

Saját költségvetést kezelő alkalmazás fejlesztése Ionic keretrendszer segítségével

Szakdolgozat

Készítette:
Horváth Máté
informatika szakos hallgató

Témavezető:
Dr. Bilicki Vilmos
egyetemi adjunktus

Szeged
2022

Feladatkiírás

A szakdolgozatom célja egy olyan mobil és webalkalmazás létrehozása Angular és Ionic keretrendszerek felhasználásával, amelynek használatával a felhasználó monitorozni és menedzselni tudja a saját pénzügyeit. Az alkalmazáson belül láthatja a kiadásait és bevételeit és rögzíthet újakat a rendszerbe, valamint számláit tudja kezelni az applikáció segítségével.

Tartalmi összefoglaló

- Téma megnevezése

Cross-platform mobil alkalmazása fejlesztése saját pénzügyi költségvetés kezelésére.

- Megadott feladat megfogalmazása

A feladatom egy, olyan alkalmazása fejlesztése, mely lehetővé teszi a felhasználó számára, hogy könnyebben nyomon tudja követni bevételeit és kiadásait.

- Megoldási mód

Elkészítettem egy Ionic mobil alkalmazást minimális dizájnnal. Létrehoztam a hozzá tartozó adatbázist, majd implementáltam a felhasználó számára elérhető funkciókat, amelyet a dolgozatomban részletesen bemutatok.

- Alkalmazott eszközök, módszerek

A szoftver elkészítéséhez Angular és Ionic keretrendszer használtam, melyek segítségével könnyen és gyorsan tudtam elkészíteni az alkalmazást. Az Ionic funkcionalitásának köszönhetően a szoftver használható Android és iOs operációs rendszert futtató készülékeken is. Az alkalmazás elérhető weben keresztül is, ennek a hostolására Firebase Hosting-ot használtam. Az adatok tárolását Firebase Firestore segítségével oldottam meg, így gyorsan és valós időben elérhetővé váltak a tárolt adatok.

- Elért eredmények

A dolgozat elkészítése során sikerült elsajátítanom a mobil alkalmazásfejlesztésekhez használt modern technológiákat. A munka során sikerült létrehoznom egy felhasználóbarát, könnyen kezelhető és egyszerűen értelmezhető felépítésű cross-platform alkalmazást.

- Kulcsszavak

Ionic, Angular, Firebase, Capacitor, Cordova

Tartalomjegyzék

Motiváció

Az emberek többsége szereti nyomon követni pénzügyeit. Legtöbbjük ezt valamilyen egyszerű adatbázis táblában próbálja meg vezetni, mint pl. Excel. Ennek a módszernek vannak hátrányai, egy egyszerű tábla megszerkesztéséhez, amin nyomon tudja követni a kiadásokat és bevételek is, olyan tudásra van szükség, amivel az átlag felhasználó nem biztos, hogy rendelkezik. Ezek az adatbázisok könnyen bonyolulttá és átláthatatlanná válhatnak. Az adatok szűréséhez, pedig mélyebb ismeretek szükségesek az adott táblázatkezelő programról. Egy másik problémát a számlák befizetési határidejének betartása okozza. Az általam fejlesztett alkalmazásban, a felhasználó ezeket egyszerű módon tudja egyetlen programban nyomon követni. Nem szükséges semmilyen plusz tudás, hogy tudja használni az applikációt, a bevitel és lekérdezések is intuitívan végig vezetik a felhasználót a lépéseken.

Több hasonló alkalmazás már elérhető a piacon, a dolgozatom elején, ezeknek a funkcióiról lenne egy összehasonlítás, beleértve a magam által fejlesztett funkciókkal. A dolgozat többi részében az alkalmazás főbb funkcióinak megvalósítását és működését mutatom be. Kitérek a rendszer működéséért felelős folyamatokra és bemutatom a használt technológiákat.

1. Meglévő költségvetést kezelő alkalmazások

Már magyar nyelven is elérhető több olyan alkalmazás, amelyek lehetőséget biztosítanak, hogy nyomon kövessük kiadásainkat és bevételeinket. Ezek között van pár olyan fejlettebb alkalmazás is, amikben lehetőséged van a bankszámláidat hozzákapcsolni a fiókodhoz, így valós időben jelennek meg a pénzügyeid és nem szükséges manuálisan rögzítened ezeket. A dolgozatomban csak azokat az alkalmazásokat vizsgáltam, ahol elérhető volt a magyar nyelv.

Ebben a részben szeretném pár szóban bemutatni a vizsgált szoftvereket. A feltűntetett applikációk mindegyikét kipróbáltam és a bemutatásukban leírom az általam tapasztalt előnyöket és hiányosságokat.

1.1. Koin

Az egyik legnépszerűbb magyar nyelven elérhető költségvetést kezelő alkalmazás a piacon. A magyar fejlesztésű szoftver elérhető mind Android, mind iOS operációs rendszerekre, valamint webes felületet is biztosít, így asztali számítógépünkről vagy laptopunkról is hozzáférhetünk adatainkhoz. Az alkalmazás használatához regisztráció szükséges, ezzel biztosítva, hogy a pénzügyi adatainkat biztoságban legyenek és csak mi férjünk hozzá.

A program lehetőséget biztosít, hogy a fiókodat összekapcsold a saját bankszámláddal, így a költéseid és bevételeid valós időben, automatikus megjelennek az alkalmazásban is. Ez nagy előny a manuális bevitellel szemben, hiszen előfordulhat, hogy elfelejted nyilvántartásba venni valamelyik pénzügyi tranzakciódat. Amennyiben a tranzakció bankszámládról történt, ami össze van kapcsolva a fiókoddal, akkor az alkalmazás is könyvelni fogja azt. Komplex alkalmazás, amely a legfontosabb funkciókat tartalmazza, hogy a költségvetésünket minél pontosabban és részletesebben nyomon követhessük.

1.2. Money Pro

Több platformra is elérhető, lehetőség van Android, iOs vagy akár Windows és macOs operációs rendszerekre is telepíteni az alkalmazást. Az ingyenes verzióban csak manuálisan tudjuk rögzíteni pénzügyi tranzakcióinkat, de van lehetőség nagyobb csomag megvásárlására is, ahol a fiókunkat szinkronizálhatjuk a bankszámlákkal. Egyik előnye, hogy tudunk rögzíteni számlákat, amelyekhez lehetőségünk van határidőt és emlékeztetőt beállítani. További előnye, hogy a felhasználói felület letisztult és modern megjelenésű, a felhasználó könnyen elér minden lényeges funkciót.

Hátránya, hogy nem lehet a megtakarításainkat vezetni az applikáción belül. Nincs olyan felület, ahol lehetőségünk van egy megadott célra gyűjtést elindítani és ehhez különböző összegeket hozzáadni vagy kivonni. Szinte az összes kipróbált alkalmazásból hiányoltam ezt a funkcionalitást, ezért az általam fejlesztett szoftverben implementálva lett ez a funkció is.

1.3. Pénzügyeim

Szintén magyar nyelven is elérhető a Pénzügyeim nevű alkalmazás, de csak Android operációs rendszerre. A rendszer jól átlátható, az ikonok között könnyedén el lehet igazodni, bár a megjelenése nem túl modern. Webes felület nem érhető el az alkalmazáshoz és saját profil létrehozására sincs lehetőségünk. Az alkalmazáshoz viszont be lehet állítani jelszót, ezzel is védve az pénzügyi adatainkat. Adatok exportálására az alkalmazás előfizetése után van lehetőség, de az ingyenes verzióban ezt nem tudjuk használni.

Hátrányai még a szoftvernek, hogy nem támogatja a számlák bevitelét a rendszerbe és a figyelmeztetést sem. Megtakarításainkat sem tudjuk benne vezetni. Összességében az alkalmazás egy rendkívül letisztult, ám a mai UI standardoktól egy kicsit elmaradó felületet biztosít a felhasználók számára. Fő szempont volt a saját applikációm fejlesztése során ezeknek a hiányosságoknak a megoldása és implementálása.

1.4. Wallet

A Wallet a cseh székhelyű BudgetBakers cég által fejlesztett alkalmazás. Az applikáció elérhető Android és iOs operációs rendszerekre is, valamint webes applikációként is megtalálható. Talán az egyik legjobb költségvetést kezelő alkalmazás a piacon. Támogatja a magyar nyelvet, felhasználói felülete modern és letisztult, a legtöbb felhasználó számára egyszerűen kezelhető. A böngészőből történő elérhetőségének köszönhetően bárhonnan el tudjuk érni az adatainkat.

Komplex és robosztus alkalmazás, amely tartalmaz minden, olyan funkciót, amelyekkel tisztán és alaposan átláthatod a pénzügyeidet.

2. Felhasznált technológiák

Ebben a részben a dolgozat alapját képző költségvetés kezelő szoftver fejlesztéséhez milyen technológiákat használtam. A továbbiakban ezeket szeretném bemutatni és ismertetni fontosabb tulajdonságaikat.

2.1. Angular

Az Angular keretrendszer segítségével gyorsan és hatékonyan készíthetünk úgynevezett single-page, magyarul egyoldalas applikációkat. Az egyoldalas alkalmazások lényege, hogy a böngésző dinamikusan tölti be a tartalmat egyetlen oldalon belül. A teljes oldal újra töltése helyett csak a szükséges komponenseket tölti be a szerverről. Az Angular a Typescript programozási nyelvet használja. A Typescript egy erősen típusos nyelv, ami Javascript-en alapul és ezt egészíti ki, ezáltal lehetővé teszi, hogy hamarabb, már fordítási időben kiderüljenek a hibák vagy kivételek.

Egy Angular alkalmazás különböző elemekből épülhet fel és különböző elemeket használhat. Ilyenek a komponensek, modulok, szolgáltatások (service). A komponensek a képernyő egy részét képezik, egy Angular alkalmazásnak mindig rendelkeznie kell egy gyökér komponenssel, amely összeköti a többi alkomponenst. Ez a hierarchia adja meg az alkalmazás vázát. A nézet és a nézethez kötődő alkalmazás logika külön-külön fájlokba vannak szervezve, hogy az egyes kódrészletek ne keveredjenek egymással. A navigációról a Routing gondoskodik, amely a komponensekhez kötődő utakkal mindig a megfelelő nézetet tölti be a képernyőn a megfelelő adatokkal.

2.2. Ionic

Az Ionic egy nyílt forráskódú UI eszközkészlet, melynek használatával modern és jó minőségű mobil és asztali alkalmazásokat készíthetünk. HTML, CSS és Javascript-en alapul, ezért a legtöbb fejlesztő számára könnyen hozzáférhető. A régebbi verziók inkább Angular specifikusak voltak, de a legújabb verziók már könnyen integrálhatók más keretrendszerekkel vagy könyvtárakkal, mint például a népszerűbb React vagy Vue frontend keretrendszerekkel.

Egyetlen kódbázissal tudunk olyan szoftvereket fejleszteni, amelyek több platformon is képesek működni. Az ilyen alkalmazások egyaránt futnak Android vagy iOs operációs rendszereket használó eszközökön, de lehetőségünk van webes környezetben is használni. A rengeteg előre implementált UI komponensek, mint a gombok, kártyák, animációk, ablakok stb. segítségével egyszerűen és gyorsan kialakíthatjuk az applikációnk felhasználói felületét. A

Capacitor és Cordova projektek segítségével képes az alkalmazásunk használni az eszköz natív funkcióit is. Mindkét kiegészítő egy könnyen használható interfészt biztosít a fejlesztőknek, amellyel hozzáférhetnek a natív API-okhoz. Ez azt jelenti, hogy egyetlen közös kóddal tudunk hozzáférni mondjuk a készülék kamera funkciójához, függetlenül attól, hogy Android vagy iOs operációs rendszert használ az eszközünk.

2.2.1. Capacitor

A Capacitor egy open-source projekt, melyet 2018-ban indítottak el, azzal a céllal, hogy egy új modernebb megközelítésbe helyezze a cross-platform mobil fejlesztést. Ez a kiegészítő a legújabb API-okat használja, segítségével teljesen hozzá tudunk férni az eszközünk natív funkcionalitásaihoz. Saját fejlesztői csapata van, ennek köszönhetően egy jól karbantartott eszközt kapunk. A biztosági és funkcionalitás béli problémák gyorsan és hatékonyan javítva vannak.

2.2.2. Cordova

A Cordova szintén open-source projekt, ám ez régebbi, mint a Capacitor. A Cordova közösségi fejlesztésként indult 2009-ben és azóta sincs neki hivatalos fejlesztői csapata, ezért a karbantartás és a hibajavítás lassabb, mint Capacitor esetén. Használatával szintén lehetőségünk van hozzáférni a natív funkcionalitásokhoz, de előfordulhat, hogy egy-két plugin nem működik megfelelően. A két projekt kompatibilis egymással, ezáltal egy Cordova kiegészítőt probléma nélkül tudunk használni egy Capacitor projektben is.

2.3. Firebase

A Firebase egy Google által fejlesztett szolgáltatást, amely komplett platformot biztosít az alkalmazásaink számára. Ennek köszönhetően egyszerűbb vagy akár robosztusabb alkalmazásoknál is szinte teljesen elhagyhatjuk a szerver oldali implementációt, helyette a Firebase által biztosított szolgáltatás csoportokat használhatjuk ennek kiváltására.

Az egyik leggyakrabban használt szolgáltatás a valós idejű adatbázis. Ezzel a szolgáltatással nem kell http request-eket küldenünk a backend felé az adatok lekérdezésére vagy módosítására. Helyette úgynevezett websocket-et használ, melynek segítségével azonnal értesülhetünk az adatbázisban történt változásokról. NoSql adatbázisokat tudunk csak kezelni a szolgáltatással, az adatainkat dokumentumokban tudjuk tárolni kulcs-érték párokként. Ezeket

a dokumentumokat tudjuk kollekcióba rendezni, minden dokumentumnak valamilyen kollekcióba kell tartoznia.

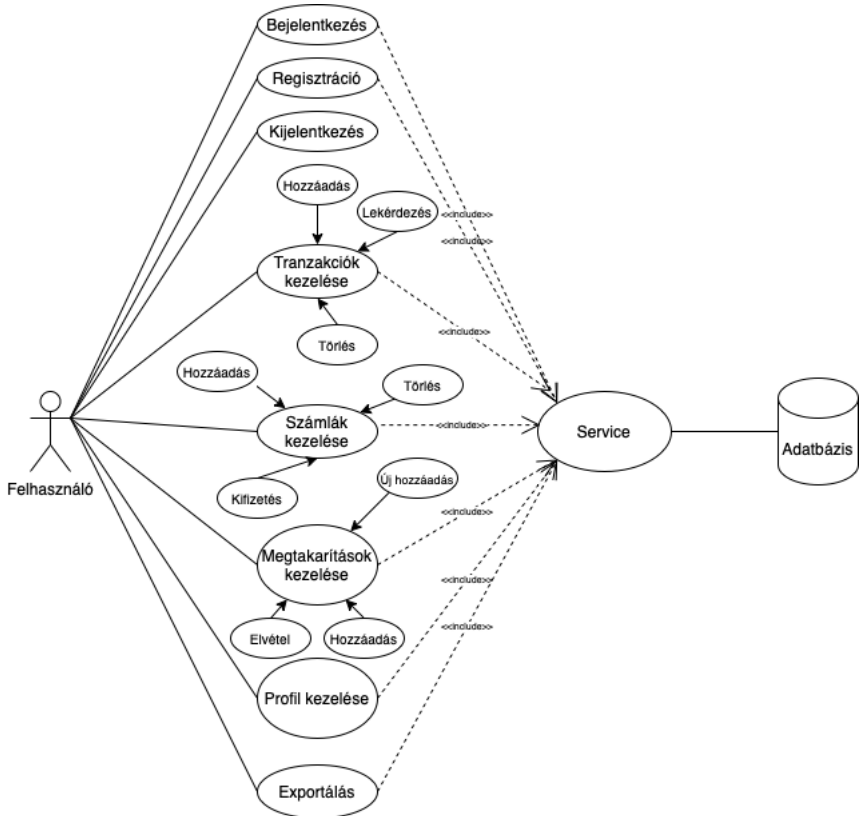
További szolgáltatásai közül lehetőségünk van weboldalaink hostolására is, amely rendkívül gyors elérést tud biztosítani. Ezeken felül a Storage szolgáltatásnak köszönhetően könnyedén tudunk különböző fájlokat tárolni az alkalmazásunk számára. A Cloud functions segítségével a felhőben tudunk folyamatokat futtatni, amelyeket különböző eseményekhez tudunk kapcsolni. Egy másik fontos és sokat használt modulja a Firebase-nek az Authentication szolgáltatás, amellyel könnyedén megvalósítható a felhasználó kezelés az alkalmazásunkhoz.

3. Funkcionális specifikáció

Ebben a fejezetben a rendszer elvárt funkcionalitásait szeretném bemutatni ábrák és magyarázat segítségével. Pár szóban írok a követelményekről, amit az elkészült rendszernek tudnia kell megvalósítani, ezenkívül kitérek a fontosabb rendszerfolyamatokra, amelyek képernyőtervekkel tervezek bemutatni.

3.1. Használati összefoglaló

A felhasználó a regisztráció és bejelentkezés folyamata mellett számos műveletet hajthat végre a rendszerrel. Egy use-case ábrán szeretném bemutatni, hogy melyek ezek a műveletek. A diagram segítségével könnyen meg lehet érteni az alkalmazás funkcionalitásait és egy átfogó képet ad a felhasználónak és a fejlesztőnek is.



3.1. ábra – Use-case diagram

Jól látható az ábrán, hogy a felhasználó milyen műveletek képes végrehajtani a rendszerben és hogy mely műveletek igényelnek adatbázis műveleteket. Az alkalmazás használatához folyamatos internetelérés szükséges, hiszen minden működéshez szükséges információ az online adatbázisban van eltárolva. A kijelentkezéshez nem kell az adatbázis módosítani vagy lekérdezni adatot. Ez a folyamat csak kliens oldalon megy végbe. A bejelentkezési folyamathoz a felhasználó adatait le kell kérdezni az adatbázisból, ugyanez igaz a regisztráció során is, ahol pedig azért van szükség az adatbázis elérésére, hogy a regisztráció során megadott adatokat el tudjuk tárolni.

A felhasználónak tudnia kell az alkalmazásba a kiadásait és bevételeit vezetni. Több funkció áll a rendelkezésére, hogy a tranzakcióit könyvelhesse. A tranzakció hozzáadása funkcióval a felhasználó létre tud hozni egy új tranzakciót, amely lehet bevétel vagy kiadás. Ezután megfelelő adatstruktúrában bekerül az adatbázisba. A felvitt adatokat lekérdezheti az adatbázisból, amelyek listaként jelennek meg az alkalmazásban, majd a listából lehetősége van a már felvitt tranzakció törlésére. Az exportálás funkció segítségével automatikusan generált adatfájlba lehet exportálni a pénzügyi tranzakciókat.

Fontos része a szoftvernek, hogy a számlákat fel lehessen tűntetni a rendszerben. Ezeket a számla hozzáadás funkcióval teheti meg a felhasználó. Miután a számlát sikerült eltárolni az adatbázisban egy emlékeztető kerül beállításra, amely figyelmezteti a felhasználót a számla

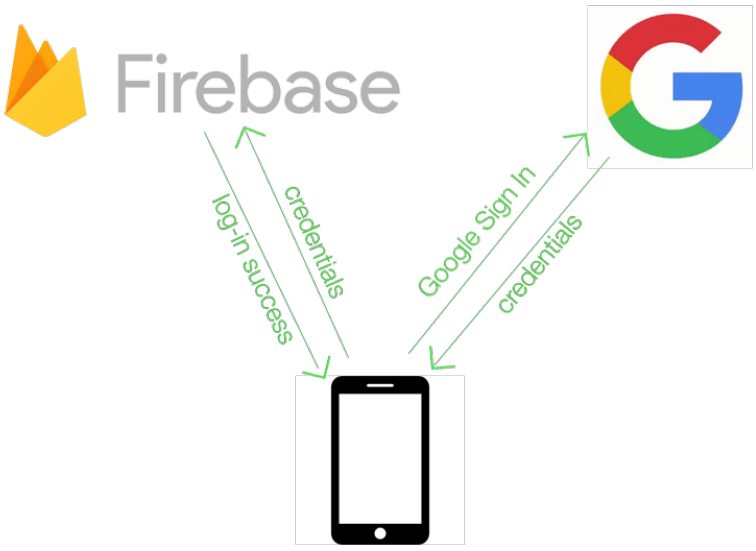
befizetési határidejének közeledtéről. A számla beviteléhez és a számla törléséhez is szükség van adatbázis kommunikációra, ezek az ábrán is megjelennek. Miután egy számla befizetésre kerül, az alkalmazásban is tudnunk kell ezt feltűntetni. Erre szolgál a számla befizetés funkció, amelyhez szintén módosítanunk kell az adatbázisunkat. Miután a módosítás megtörtént és megjelöltük befizetettként a beállított figyelmeztetést is törölni kell.

Beállíthatunk egy célt, hogy egyszerűbben kezelhessük megtakarításainkat. Bármikor tudunk új gyűjtést létrehozni a már meglévőkön kívül. Egy megtakarításhoz két fő funkció tartozik. Tudunk egy megadott összeget hozzáadni, ezzel közelebb kerülve a beállított célösszeg eléréséhez. Valamint el is tudunk a már összegyűjtött összegből.

Mindegyik regisztrációhoz a felhasználók képesek egyéni profilt létrehozni. A profilok kezeléséhez elengedhetetlen a kommunikáció az adatbázissal. A regisztrált felhasználó az alkalmazáson belül tud profilképet feltölteni és képes a megjelenített adatokat pl. név módosítani, ezzel jobban egyénre szabhatóvá válik az alkalmazás.

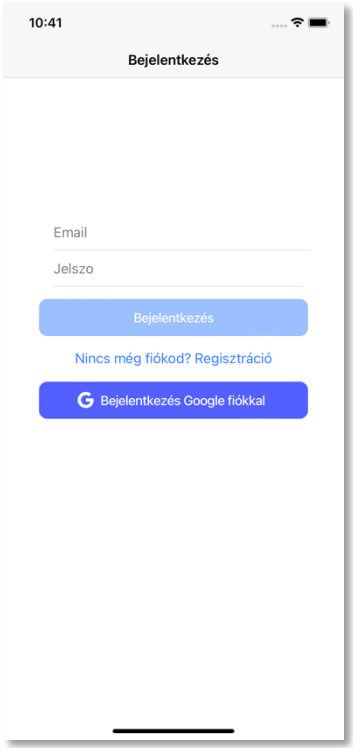
3.2. Felhasználókezelés, autentikáció

A felhasználói autentikációhoz a Firebase Authentication szolgáltatását építettem be a szoftverbe. Ezzel könnyen és gyorsan meg tudtam valósítani a bejelentkezés és regisztráció folyamatát. A szolgáltatás lehetőséget biztosít az email és jelszó páros alapján történő felhasználó kezelésre, ezeken felül a modul még támogatja Google fiókkal történő bejelentkezést is, amely szintén implementálva lett a kész alkalmazásban.

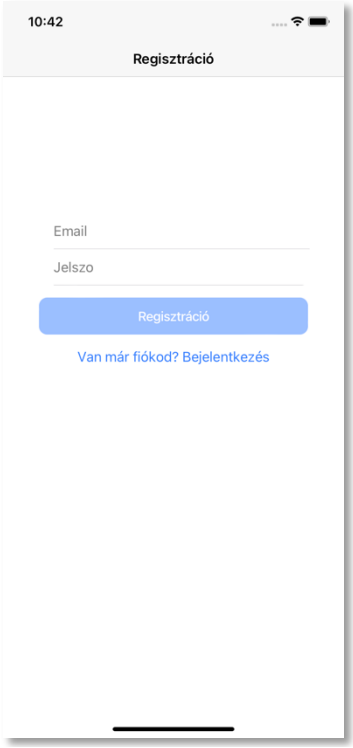


3.2 ábra – Google autentikáció folyamata

A program használatához rendelkeznünk kell egy regisztrált felhasználói fiókkal, hiszen a pénzügyi adatok, ehhez a fiókhoz tudjuk kapcsolni az adatbázisban. Az applikáció első elindításakor egy bejelentkezés képernyő fogad minket, ahol az adatok megadásával be tudunk jelentkezni a fiókunkban, ha még nem rendelkezünk fiókkal, akkor a regisztráció gombra kattintva át kerülünk a regisztrációs felületre. A regisztrációhoz egy email és jelszó párost kell megadni a felhasználónak. Ezután a megadott email címre egy megerősítő levél kerül kiküldésre, amelyben található linkre kattintva lesz teljes a regisztráció. Sikeres regisztráció után a bejelentkezés képernyőn az email és jelszó adatokat megadva vagy a Google fiókunkkal be tudunk jelentkezni az alkalmazásba és hozzáférhetünk a funkciókhoz.



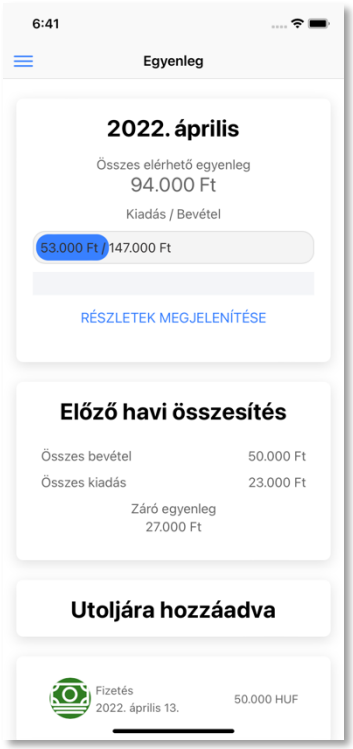
3.3. ábra – Bejelentkezés képernyő



3.4. ábra – Regisztráció képernyő

3.3. Összefoglaló oldal

Sikeres bejelentkezés az alkalmazás fő oldalára kerülünk, ahol egy összefoglalót láthatunk az egyenlegünkről.



3.5. ábra – Alkalmazás főoldala

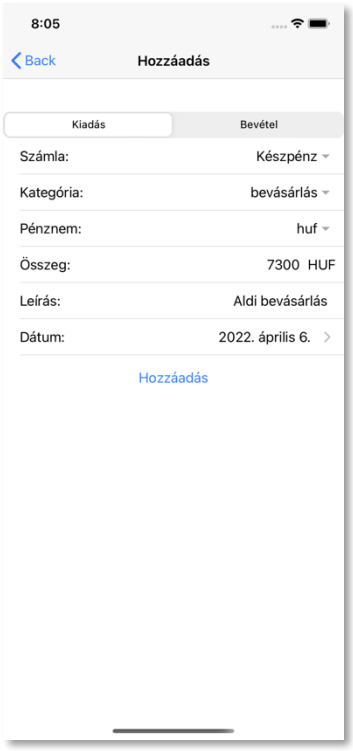
A főoldalon található első kártyán az aktuális hónapról jelenik meg egy kisebb statisztika, hogy a felhasználó gyorsan informálódni tudjon a fontos pénzügyi információiról. Az „Összes elérhető egyenleg” alatt a bankszámlán és készpénzben tárolt egyenlegei kerülnek összeadásra, tehát azt mutatja meg, hogy mennyi pénzzel tud még gazdálkodni a felhasználó. Az alatta található részben a bevételek és kiadások aránya van vizuálisan és összegekkel is feltűntetve. A „Részletek megjelenítése” gombra kattintva a részletes egyenleg oldalra kerülünk, ahol már tételek lebontva láthatjuk a tranzakciókat.

A második kártyán az előző havi kiadások és bevételek vannak feltűntetve, valamint a záró egyenlegünk. Ez azért hasznos mert gyors információt biztosít az alkalmazás használójának, hogy az előző hónaphoz képest most milyen irányba mozogtak el a bevételek és kiadások, ezáltal jobban meg tudja tervezni a pénzügyi tranzakcióit.

Az utoljára hozzáadott részben midig az 5 legutoljára hozzáadott, vagyis a legfrissebb kiadásai jellenek meg a felhasználónak.

3.4. Tranzakció hozzáadás folyamata

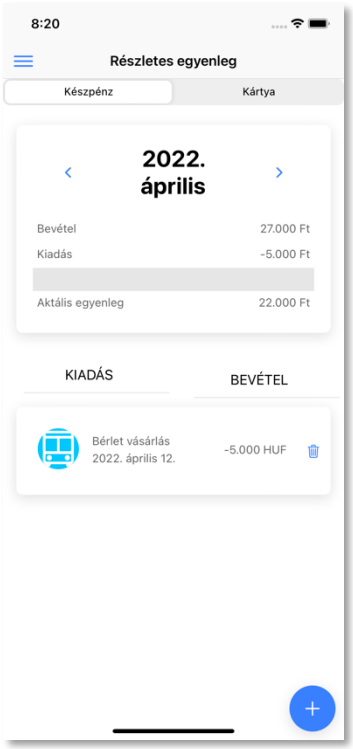
Bevételek vagy kiadások rögzítésére a részletes egyenleg oldalon van lehetősége a felhasználónak. A felület az oldalsó menüből vagy a főoldalról is elérhető. Miután az oldal betöltődött a lap jobb alsó sarkában található lebegő gombra kattintva kerülünk át arra a lapra, ahol rögzíteni tudjuk a tranzakciót.



3.6. ábra – Tranzakció hozzáadása

A hozzáadás oldalon meg kell adni az űrlapon kért adatokat, hogy rögzíthesse az adatokat. Minden mezőt ki kell tölteni. Legelőször is ki kell választani a legfelső csúszkán, hogy bevételt vagy kiadást szeretnénk felvinni az adatbázisba. A következő opciónál kiválasztjuk, hogy a készpénzünkből vagy bankszámlákról történt a tranzakció. Az alkalmazás képes több pénznemet kezelni, ezeket a hozzáadáskor tudjuk kiválasztani. Ezeket az alkalmazás a háttérben átváltja forintra és így kerül elmentésre az adatbázisban. Az átváltás folyamatát és megvalósítását a későbbiekben bemutatom. Az összeg, leírás és dátum megadása után, ha minden mezőt helyesen töltöttünk ki, akkor a „Hozzáadás” gomb aktívvá válik és rögzíthetjük a tranzakciót.

3.4. Részletes egyenleg megjelenítése



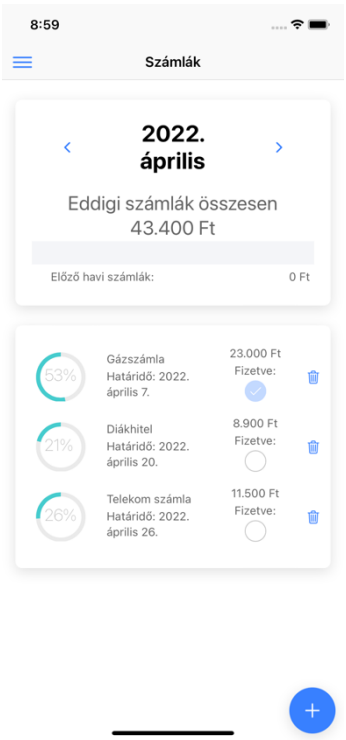
3.7. ábra – Részletes egyenleg oldal

A részletes egyenleg oldalon a bejelentkezett felhasználó által rögzített kiadás vagy bevétel adatok jelennek meg havi lebontásban. A képernyő tetején lévő szegmens navigáción lehet kiválasztani, hogy éppen a készpénzes tranzakcióinkat vagy a bankszámlánkat használó tranzakcióinkat jelenítse meg az alkalmazás. Az első kártya mindig az aktuális hónap jelenik meg, de a jobbra és balra nyilakra kattintva lehetőség van a hónapok között váltogatni. Az adatok dinamikusan frissülnek a kiválasztott hónapnak megfelelően. A kártyán még szerepel az adott hónapban rögzített összkiadásunk és összbevételünk, valamint az aktuálisan elérhető egyenlegünk. Ezek az adatok mindig forintban jelennek meg függetlenül attól, hogy milyen pénznemben kerültek rögzítésre a tranzakciók.

Az alatta található részben kiválaszthatjuk, hogy a kiadásokat vagy a bevételeket listázza nekünk a rendszer. Mindegyik tranzakció külön-külön kis kártyán jelenik meg, amelynek a jobb oldalán található „kuka” ikonra kattintva törölhetjük azt az adatbázisból.

3.5. Számlák kezelése

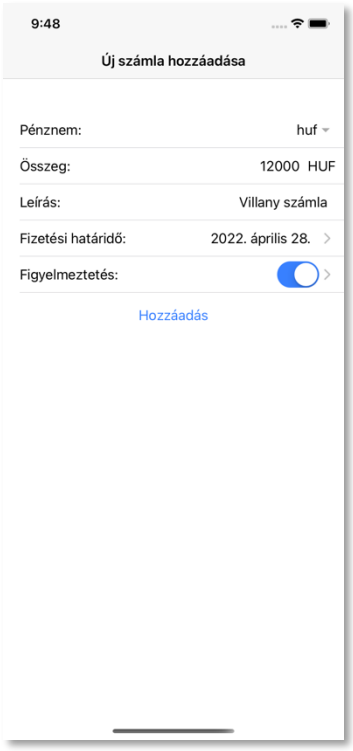
Az oldalsó menü panel „Számlák” opcióját választva az alkalmazás átnavigál bennünket, arra az oldalra, ahol a felhasználó megtekintheti a számláit és lehetősége van új számla bevitelére.



3.8. ábra – Számla oldal

Felépítése nagyon hasonló a „Részletes egyenleg” oldalhoz, a képernyő felső részén található kártyán itt is megjelenik a dátumválasztó, ahol a nyilak segítségével tudunk váltani a hónapok között. Alatta egy összegző rész található, ahol a bevitt számlák összege jelenik meg, hogy a felhasználó könnyen értesülni tudjon a számlák összértékéről. Valamint az előző havi számlák összértéke is látható az oldalon, ezáltal a felhasználó képes viszonyítani a múltbéli értékekhez.

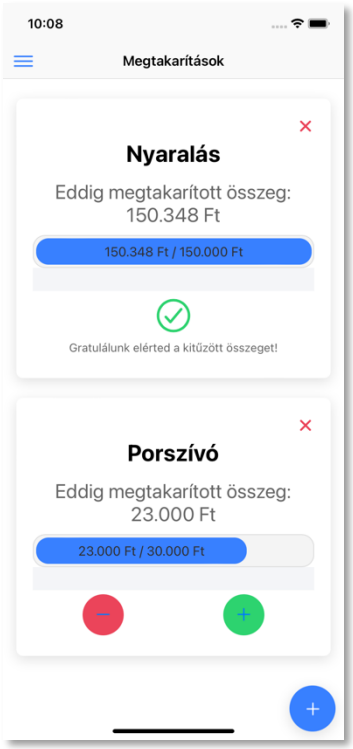
A másik kártyán az egyes számlák jellenek meg listában megjelenítve, a listaelemek elején található egy kör diagram, amely azt mutatja, hogy az adott számla összege mekkora része az összes számla értékének. Ezt az arányt a szoftver százalékosan jeleníti meg, így vizuálisan is képet kap a felhasználó, hogy mely számlák teszik ki a kiadások nagy részét. Egy checkbox kijelölésével megjelölhetjük a számlánkat kifizetettként, miután ezt bejelöltük a rendszer nem küld figyelmeztetést a határidő lejártával kapcsolatban. A „kuka” ikonra kattintva törölhetjük az adatbázisból a számlát és ezáltal a határidő figyelmeztetés is megszűnik.



3.9. ábra – Számla hozzáadása

A lebegő gombra kattintva a jobb alsó sarokban a rendszer átnavigál az „Új számla hozzáadás” oldalra, ahol a felhasználó a szükséges adatok megadása után új számlát menthet el az adatbázisban. Többféle pénznemű számla rögzítésére is van lehetőség, amelyet az űrlap elején található legördülő listából választhatunk ki. Az összeg és leírás megadása után meg kell adni a számla befizetési határidejét, majd a felhasználó dönthet, hogy szeretné-e, hogy a rendszer automatikusan figyelmeztesse a határidő lejártáról. Amennyiben a figyelmeztetést kéri a felhasználó az alkalmazás push notification-ben fogja értesíteni a felhasználót a lejárat előtt egy nappal.

3.6. Megtakarítások nyilvántartása



3.10. ábra – Megtakarítások oldal

Szerettem volna egy olyan egyszerű felépítésű és könnyen kezelhető funkciót létrehozni az alkalmazásban, melynek segítségével az adott felhasználó képes gyűjtést létrehozni saját maga számára. A beállított célhoz hozzárendelhet egy célösszeget, amelyhez képes megadott összeget hozzáadni vagy elvenni belőle.

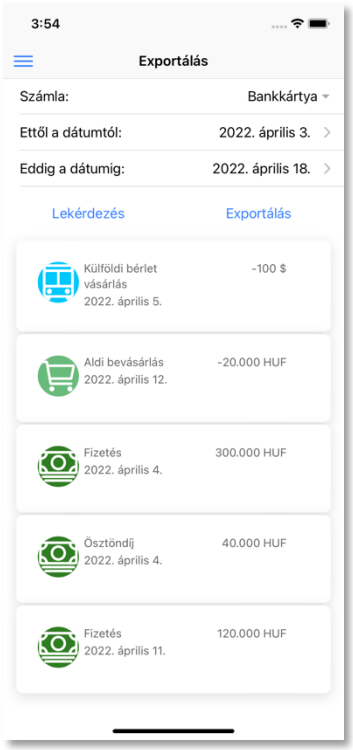
A „Megtakarítások” oldalon a felhasználó által létrehozott megtakarítási célok jelennek meg kártyát formájában. Egy kártyán megtalálható a leírás, amelyre a megtakarítást elindítottuk, az eddig megtakarított összeg, amelyet a plusz és kivonás gombokkal tudunk módosítani. A zöld plusz gombra kattintva tudunk hozzáadni a megtakarításhoz, a piros kivonás gombbal pedig a már megtakarított összegből tudunk visszavenni. Az alatta lévő sávban vizuálisan és összegekkel is meg van jelenítve az előrehaladás, amely dinamikusan változik a célösszeg és a jelenleg megtakarított összeg arányában.

A kártyák jobb felső sarkában található kereszt ikonra kattintva bármikor ki tudjuk törölni az elindított megtakarítási folyamatot függetlenül attól, hogy elértük a kitűzött célt vagy sem. Új megtakarítási folyamat elindítása rendkívül egyszerűen megoldható, a jobb alsó sarokban látható lebegő gombra kattintva, csak két adatot kell megadni, egy leírást, hogy mire szeretnénk

félretenni a pénzünket és egy célösszeget. Ezután bekerül az adatbázisba és dinamikusan valós időben tudjuk kezelni a megtakarításainkat.

3.7. Exportálás

Az alkalmazás használatához állandó internet kapcsolat szükséges a felhasználó pénzügyi adatai a felhőalapú adatbázisban vannak eltárolva. Ahhoz, hogy ezekhez hozzáférhessünk kapcsolatot kell létesíteni az adatbázis szolgáltatással, ezért nincs offline használatra lehetőség. Ugyanakkor előfordulhat, hogy az alkalmazás használója szeretné, ha internet kapcsolat nélkül is bármikor hozzáférhetne az adatokhoz, vagy csak egy megadott intervallumban megtörtént tranzakciókat szeretné lekérdezni. Ezért a szoftverhez kifejlesztésre került egy exportálást végző funkció, amellyel a lekérdezett adatokat .csv fájlként képes a készülékre menteni a felhasználó.



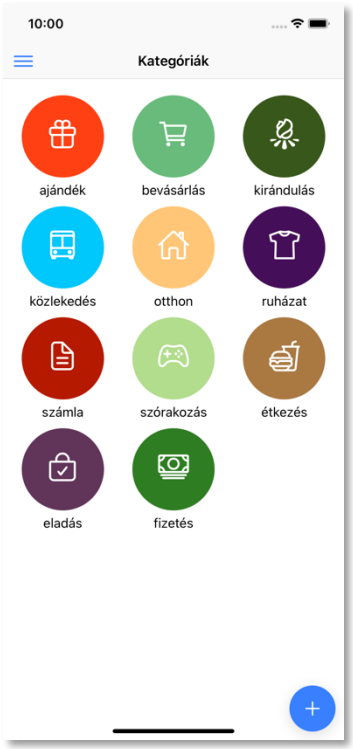
3.11. ábra – Lekérdezés megjelenítése

Az exportáláshoz először is ki kell töltenünk a felületen található űrlapot, ahol ki kell választanunk, hogy a bakszámlán történt tranzakciókat vagy a készpénzben rögzített kiadások és bevételeket szeretnénk menteni. Kitöltés után aktívvá válik a lekérdezés gomb, megnyomást követően, ha a lekérdezés sikerrel járt és van adat, amit ki lehet exportálni külső fájlba, akkor az felsorolásra kerül a lapon és aktívvá válik az exportálás gomb. Ha lekérdezés nem adott vissza semmilyen adatot, akkor „Nincs adat” jelenik meg a lapon és az exportálás gomb inaktív

marad, hiszen így nincs értelme a külső fájlba mentésnek. Az exportálás követően egy felugró ablak jelenik meg, ahol látható lesz az művelet sikeressége és a fájl helye az eszközön.

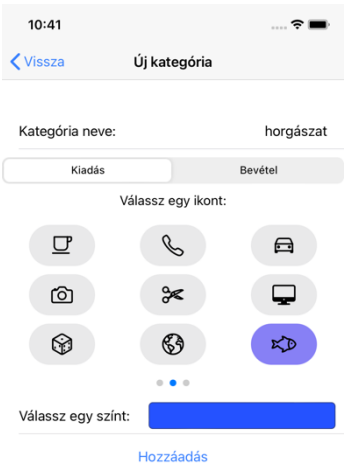
3.8. Kategóriák

Az alkalmazás rendelkezik beépített kategóriákkal, hogy a felhasználó még rendszerezettebben tudja kezelni a pénzügyeit. A „Kategóriák” oldalon a beépített és a felhasználó által létrehozott kategóriák jelennek meg felsorolás szerűen.



3.11. ábra – Kategóriák listája

A kategóriák oldalra navigálva a rendszerben eltárolt összes kategória megjelenik a 3.11 ábrán látható rácsos elrendezésben. Minden kategória rendelkezik egy saját színnel és ikonnal a könnyebb megkülönböztetés miatt. A kategóriára kattintva egy másik képernyőre kerülünk, ahol a kiválasztott kategóriába tartozó összes tranzakció megjelenik egyszerűen felsorolva az oldalon. A lebegő gombra kattintva a szoftver átnavigál egy másik képernyőre, ahol új egyedi kategóriákat tudunk hozzáadni az alap kategóriákhoz.



3.12. ábra – Új kategória felvétele

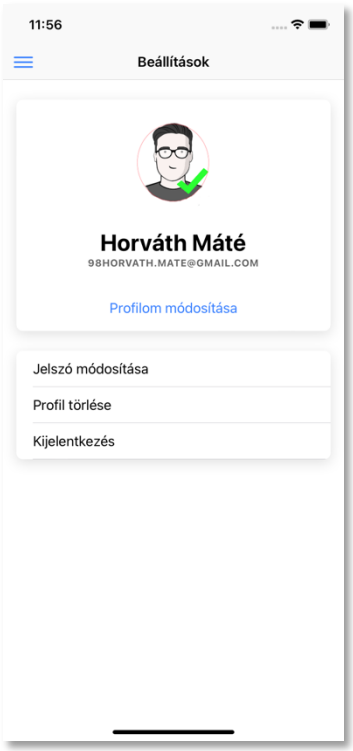
Ha a felhasználó szeretne új kategóriát elmenteni a rendszerben az „Új kategória” oldalon ki kell töltenie az űrlapon kért adatokat. Meg kell adnia az új kategória nevét és ki kell választania, hogy a kiadásokhoz vagy a bevételekhez tartozó kategóriáknál jelenjen meg. Választanunk kell egy ikont, amely a kategóriát hivatott vizuálisan szimbolizálni, valamint ez fog megjelenni amikor a tranzakciók felsorolásra kerülnek. A megjelenéshez tartozik egy szín is. Az ikon és a megadott szín alapján könnyebb beazonosítani, hogy az adott tranzakció melyik kategóriába lett sorolva, amikor rögzítettük azokat az adatbázisban.

3.9. Profil oldal

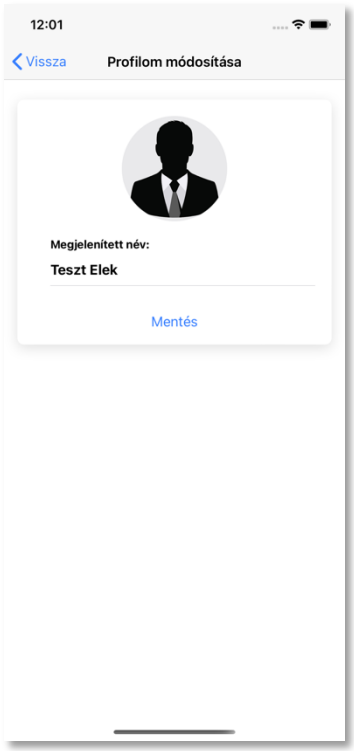
A profilunkat kétféleképpen tekinthetjük meg, az oldalsó menü panelt kinyitva az opciók felett megjelenik a felhasználó által beállított profilkép (3.13. ábra), a beállított név és az e-mail cím. Itt nincs lehetőségünk a módosításra, csak miután átnavigáltunk a profil oldalra.

Az oldal tetején található kártyán is megjelenítésre kerültek a felhasználói profil adatai, egy kör alakú területen látható a beállított profil kép, alatta a megjelenített név és az regisztrált e-mail cím. A „Profil módosítása gombra kattintva tudja a felhasználó szerkeszteni ezeket az adatokat. Átkerül egy másik oldalra, ahol a szoftver futtató eszköz fotói közül tud választani profilképet és feltölteni azt az adatbázisba, valamint az alkalmazásban megjelenő nevet is képes módosítani.

Ezen az oldalon (3.13. ábra) lehetősége van még a felhasználónak a jelszava módosítását kérni és a felhasználói fiokját is törölheti véglegesen az adatbázisból, ilyenkor helyreállításra már nincs lehetőség. Megtalálható még egy kijelentkezés gomb is, amely kijelentkezteti a felhasználót az applikációból, újbóli indítást követően a megfelelő adatokkal ismét be kell jelentkeznie.



2.13. ábra – Profil oldal



3.14. ábra – Profil módosítása

4. Az alkalmazás szerkezeti felépítése

Ebben a fejezetben azt szeretném bemutatni, hogy az általam készített alkalmazás milyen főbb szerkezeti elemei vannak. Egy Angular projektnek egyedi felépítése van, de megpróbálja követni a MVC mintát. Ezek az alkalmazások moduláris felépítésűek, a kialakított hierarchiának a legalapvetőbb építőkockái a komponensek. A komponensekből létrejött hierarchikus struktúrából jön létre az Angular alkalmazás.

4.1. Modulok

Az Angular modulok nagy szerepük van a kód strukturálásában, funkció és terület alapján rendszerezhetővé teszi a komponenseket, direktívákat, szolgáltatásokat stb.. Az NgModule egy @NgModul dekorátorral ellátott osztály, az @NgModule dekorátor metainformációk alapján

adja meg a sablon fordításának módját és az injector futásidejű működését. Ezen információk alapján azonosítja saját komponenseit, csöveit és egyéb szerkezeti elemeit, illetve lehető teszi ezek publikálását. A szolgáltatások injektálását és szkópját is it tudjuk megadni. Minden alkalmazás rendelkezik legalább egy gyökér modullal, amelyet általában AppModule-nak nevezünk.

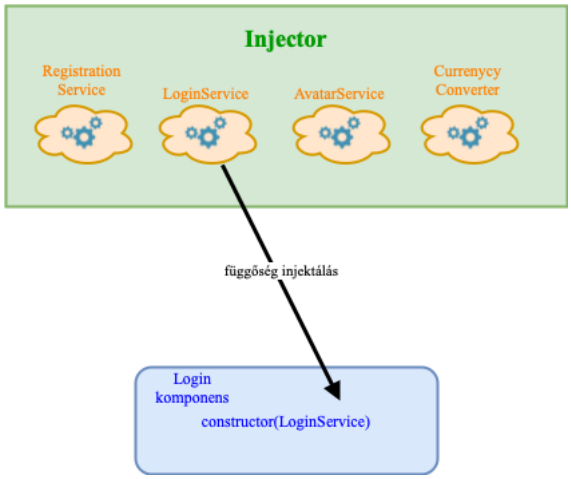
Az alkalmazásban található modulok:

- AppModule
- LoginPageModule
- RegisterPageModule
- HomePageModule
- DetailedBalancePageModule
- InvoicesPageModule
- SavingsPageModule
- SettingsPageModule
- CategoriesPageModule
- ExportPageModule
- ProfilePageModule

4.2. Szolgáltatások (Services)

A szolgáltatások segítséget nyújtanak abban, hogy alkalmazásunk még strukturáltabb és átláthatóbb legyen. A „Szolgáltatás” egy széles kategória bármi lehet benne, ami az alkalmazás számára szükséges lehet. Fő célja, hogy az üzleti logikát, a metódusokat és különféle adatokat ki lehessen szervezni egy külön egységbe, amelyet az alkalmazás többi egységei el tudnak érni és használni tudják az implementált funkciókat. A szolgálatásokat érdemes úgy létrehozni, hogy azok egy jól meghatározott szűk funkció listával rendelkezzenek és egy szolgáltatás csak egy egységhez tartozó folyamatokat implementáljon.

A szolgáltatások is alapelemei minden Angular alkalmazásnak, a létrehozott komponensek ezekre épülnek. Ebben történhet pl. a szerverrel történő kommunikáció, felhasználói adat validáció, naplózás stb.. Minden szolgáltatás az alkalmazás élettartama alatt csak egy inicializálódik és futás során bármikor elérhetőek az alkalmazás bármely részében.



4.1. ábra – Szolgáltatás injektálás

A következőkben szeretném bemutatni az alkalmazásom főbb szolgáltatásait és azok fontosabb metódusait. A szolgáltatások külön mappába lettek rendezve a kód könnyebb átláthatósága miatt.

Authentication Service

A felhasználókezelésért felelős szolgáltatás. Az ebben szereplő metódusok valósítják meg a bejelentkezés és regisztráció funkciót.

- RegisterUser(): A paraméterben átadott email és jelszó párossal regisztrálja a felhasználót a rendszerben. A metódus a Firebase Authentication folyamatait használja.
- SignIn(), SignOut(): A felhasználó be- és kijelentkezését valósítja meg. A SignIn() metódus paraméterben megkapja az e-mail cím és jelszó párost. Szintén a Firebase Authentication folyamatait használja.
- SendVerificationEmail(): Regisztráció után egy megerősítő e-mailt küld a regisztrált felhasználónak.
- PasswordRecover(): Jelszó módosításhoz használt metódus. A megadott e-mail címre kerül kiküldésre egy link, ahol a jelszó megváltoztatható.
- GoogleAuth(): Google fiókkal történő bejelentkezést valósítja meg. A Firebase Authentication signInWithPopup() folyamatot implementálja.

Avatar Service

A regisztrált felhasználóknak lehetőségük van a profiljukat testre szabni és saját profilképet feltölteni, valamint az alkalmazásban megjelenített nevet megváltoztatni.

- uploadImage(): Paraméterként egy Capacitor Photo objektumok vár, amelyet a készülék fotói közül tudunk kiválasztani. A kapott képet feltölti a Firebase Storage szolgáltatásába és innen lesz elérhető.
- updateUserProfile(): Frissíti az adatbázisban a felhasználói adatokat, miután a kép feltöltésre került a folyamat végén visszakapott elérési útvonalat elmenti a felhasználó többi adatához.

Currency Converter Service

Az alkalmazás képes kezelni több pénznemet is, ezek az eredeti pénznemben kerülnek mentésre az adatbázisban. Viszont a szoftverben több helyen szükséges, hogy ezeket az összegeket forintra átváltva is tudjuk használni, ezért lett implementálva egy valutaváltó funkció.

- convertCurrencyToHuf(): Ez a metódus valósítja meg a valuta forintba történő átváltását. Paraméterben kapja meg az eredeti pénznem kódját és az átváltani kívánt összeget és visszatér a forintra váltott összeggel.

Transaction Service

Ebben a szolgáltatásban találhatóak a tranzakciók mentéséért és lekérdezéséért felelős metódusok.

- addNewExpense(), addNewIncome(): A metódusok segítségével a kiadások és bevételek rögzíthetők az adatbázisba.
- deleteTransaction(): A kiválasztott tranzakció törlését teszi lehetővé az adatbázisból. Paraméterben kapja meg, hogy melyik elem kerüljön törlésre.
- getCardExpenses(), getCardIncomes(): Azokat a tranzakciókat kérdezhetjük le az adatbázisból, amelyek a bankszámlánkhoz kapcsolódnak.
- getCashExpenses(), getCashIncomes(): A készpénzes ügyleteinket kaphatjuk vissza a szervertől a metódusok segítségével.

Data Export Service

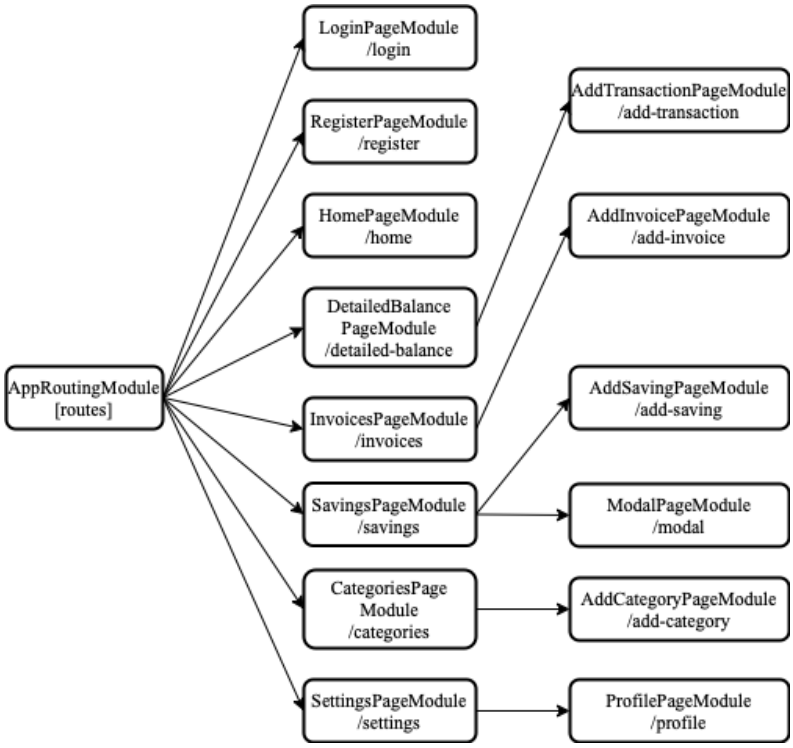
Az adatok exportálásának megvalósítása is külön szolgáltatásba lett kiszervezve. A service a kívánt adatok átalakításáért felel, hogy azok megfelelően menthetők lehegyen külső adatfájlba. A szolgáltatásban került implementálásra az elkészített adatfájlnak eszközre mentésének folyamata is.

- convertToCSV(): Paraméterben adhatjuk át a metódusnak az adatokat, amelyeket szeretnénk külső fájlba menteni. A folyamat végén megkapjuk az átkonvertált adatot, amelyből már tudunk .csv fájlt készíteni.
- downloadCSV(): Az elkészített fájl tudjuk az eszközünkre menteni a metódus segítségével.

4.2. Routing

A Routing tölti be az alkalmazásunkban azt a szerepet, hogy képesek legyünk navigálni az oldalak között. Az Angular keretrendszer használatával single-page alkalmazásokat hozhatunk létre, melynek lényege egy oldalon belül kerülnek betöltésre a különböző nézetek. Azt, hogy mikor melyik nézet legyen a képernyőn megjelenítve a Routing segítségével szabályozhatjuk.

Az oldalak közti navigáció URL-ek segítségével történik, ezekhez az URL-ekhez tartozik egy nézet. Ha az alkalmazásban ez az útvonal megváltozik a router az útvonalhoz tartozó komponenst tölti be és jeleníti meg a képernyőn.



4.1. ábra – Routing

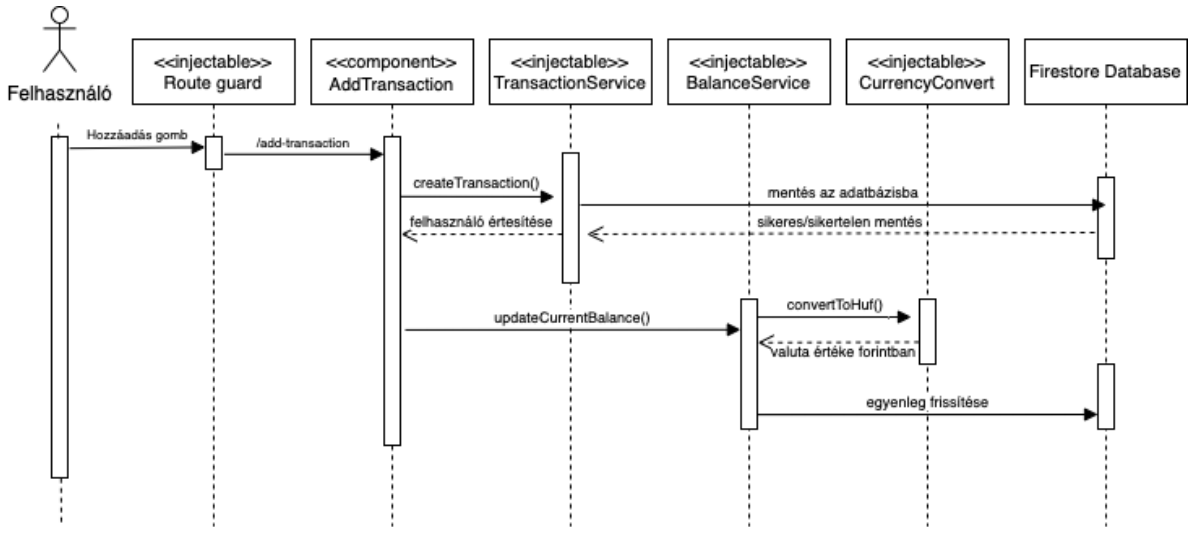
A fenti 4.1. ábra szemlélteti az alkalmazásban használt routing felépítését. Az alkalmazást elindítva az alapértelmezett root útvonal töltődik be, amely a HomePageModule-t tartalmazza. Az útvonalak Route Guard-al vannak védve, amennyiben a felhasználó nincs bejelentkezve csak a /login és a /register útvonalak érhetőek el számára. Ezt az ellenőrzést a szoftver

automatikusan elvégzi, ha a felhasználó azonostás nélkül próbál meg elérni bármilyen útvonalat a router a /login-hoz tartozó komponens fogja betölteni. Sikeres bejelentkezést követően az ábra második oszlopában lévő modulok az alkalmazás oldalsó menüjének segítségével érhetőek el. A további útvonalak csak az egyes oldalakon megjelenő navigációs gombok és linkek segítségével érhetőek el.

5. Az alkalmazás fontosabb folyamatai és kódrészei

Ebben a fejezetben a kész alkalmazás pár fontosabb folyamatát szeretném bemutatni és a legfontosabb kódrészeket pár szóban ismertetni. Az általam bemutatott részekhez, ahol szükséges illusztrálom egy szekvencia diagrammal is, hogy átláthatóbb legyen az egyes folyamatok, funkciók működése.

5.1. Tranzakció hozzáadás folyamata



5.1. ábra – Tranzakció hozzáadás szekvencia diagram

A folyamat elindításához mindenképpen szükséges, hogy be legyünk jelentkezve, ugyanis enélkül nem tudjuk elérni az oldalt, ahol új bevételt vagy kiadást tudunk létrehozni. Miután az ellenőrzés megtörtént a router betölti a „Tranzakció hozzáadása” oldalt, ahol egy űrlapot kell kitöltenie a felhasználónak. A „Hozzáadás” gomb csak akkor válik elérhetővé, ha minden szükséges adatot megadtunk. A gomb megnyomásakor elindul a folyamat, először el kell dönteni, hogy bevételt vagy kiadást rögzítünk. Ezután a TransactionService megfelelő metódusa fog lefutni és elmenteni az adatbázisba az adatokat, majd a felhasználót értesítjük egy toast üzenet segítségével a folyamat állapotáról.

A bevétel vagy kiadás rögzítésének folyamatával párhuzamosan meghívódik egy másik metódus is, amelynek segítségével az éppen elérhető egyenlegünket és a kiadások és bevételek arányát tudjuk naprakészen tartani. Az alkalmazásban tudunk többféle valutában is tranzakciókat rögzíteni, viszont a megfelelő működéshez szükséges ezeket forintra átváltva is tárolni. Ezért az egyenleg frissítésekor felhasználjuk a CurrencyConvert szolgáltatást is, amely visszaadja az adott összegű valuta értékét forintban. Ezután már lehetséges az aktuális elérhető egyenlegünket frissíteni az adatbázisban.

```
public addNewExpense(transaction: Transaction, option: string) {
  return this.userRef.collection(option).doc(this.user.uid).collection('expenses').add(transaction);
}

public addNewIncome(transaction: Transaction, option: string) {
  return this.userRef.collection(option).doc(this.user.uid).collection('incomes').add(transaction);
}
```

5.2. ábra – Bevétel és kiadás mentése

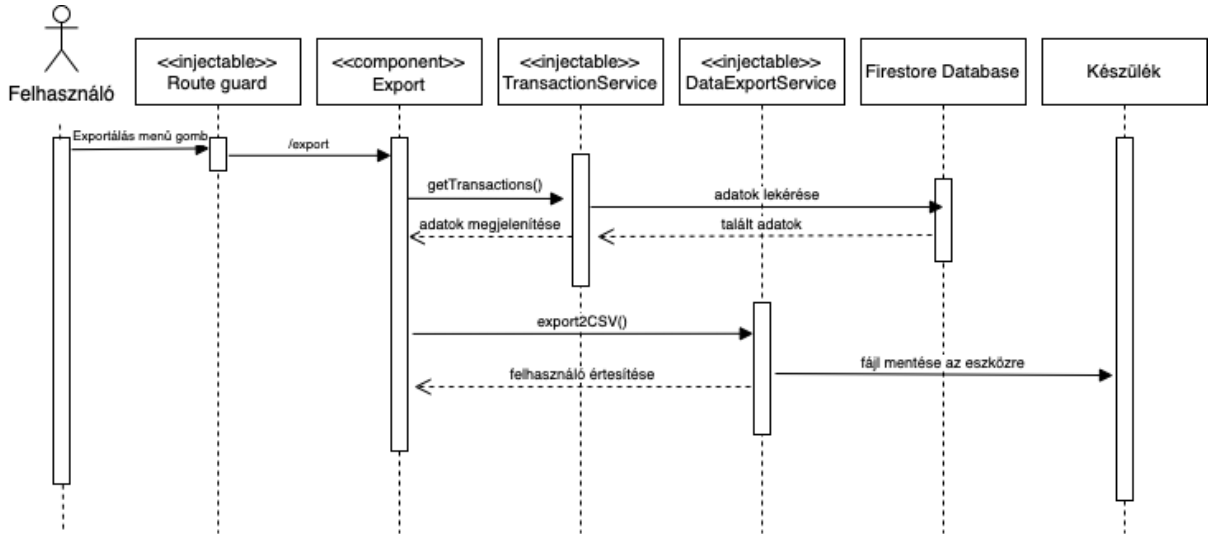
A fenti ábra az bevételek és kiadások adatbázisba történő mentését valósítja meg. A metódusok paraméterként kapják meg az adatot, amely a szintén paraméterként kapott kollekcióban kerül eltárolásra. A *userRef* egy AngularFirestoreDocument típusú változó, amely az aktuálisan bejelentkezett felhasználóhoz tartozó referenciát tartalmazza. Mindkét metódus visszatérési értéke egy Promise, ezáltal az AddTransaction komponensben a *.then()* és a *.catch()* metódusok segítségével tudjuk kezelni a sikeres és sikertelen adatbázis műveleteket is.

Sikeres művelet esetén az egyenleg frissítéséért felelős metódus is meghívásra kerül, amelyet a lenti (5.2.) ábra szemléltet. Az aktuális egyenleg havi bontásban van tárolva az adatbázisban, ezért a folyamat elején a paraméterben kapott tranzakcióból kerül elkészítésre a módosítandó dokumentumhoz tartozó referencia. Mivel előfordulhat, hogy az adott tranzakció nem forintban szeretnénk rögzíteni, ezért minden esetben átváltásra kerül az összeg. Ezután a Firestore increment() metódusa segítségével tudjuk módosítani a mező értékét. Ha nem találtunk a referenciához megfelelő dokumentumot, akkor új dokumentumként kerül mentésre.

```
async updateTotal(transaction: Transaction, option: string) {
  const docId = this.dateFormat(transaction.createdAt);
  if(transaction.value > 0) {
    this.userRef.collection(option).doc(this.user.uid).collection('currentBalance').doc(docId).update({
      balanceHUF: increment(await this.currenyConverter.convertCurrencyToHuf(transaction.currency, transaction.value)),
      incomeAmountHUF: increment(await this.currenyConverter.convertCurrencyToHuf(transaction.currency, transaction.value))
    }).catch(async error => {
      if (error.code == "not-found") {
        this.userRef.collection(option).doc(this.user.uid).collection('currentBalance').doc(docId).set({
          balanceHUF: await this.currenyConverter.convertCurrencyToHuf(transaction.currency, transaction.value),
          incomeAmountHUF: increment(await this.currenyConverter.convertCurrencyToHuf(transaction.currency, transaction.value)),
          expenseAmountHUF: 0
        });
      } else {
        console.log(error);
      }
    });
  } else if(transaction.value <= 0) {
    this.userRef.collection(option).doc(this.user.uid).collection('currentBalance').doc(docId).update({
      balanceHUF: increment(await this.currenyConverter.convertCurrencyToHuf(transaction.currency, transaction.value)),
      expenseAmountHUF: increment(await this.currenyConverter.convertCurrencyToHuf(transaction.currency, transaction.value))
    }).catch(async error => {
      if (error.code == "not-found") {
        this.userRef.collection(option).doc(this.user.uid).collection('currentBalance').doc(docId).set({
          balanceHUF: await this.currenyConverter.convertCurrencyToHuf(transaction.currency, transaction.value),
          expenseAmountHUF: await this.currenyConverter.convertCurrencyToHuf(transaction.currency, transaction.value),
          incomeAmountHUF: 0
        });
      } else {
        console.log(error);
      }
    });
  }
}
```

5.2. ábra – Egyenleg frissítéséért felelős metódus

5.2. Adatok exportálása



5.3. ábra – Exportálás szekvencia diagram

Az adatok exportálásához először az „Exportálás” oldalra kell navigálnunk, amelyet az oldalsó menü exportálás gombjára kattintva tehetünk meg. Csak bejelentkezett felhasználó tudja elérni az oldalt ez mindig ellenőrizve van. Az oldalon egy űrlap jelenik meg, ahol meg

tudjuk adni azt az időintervallumot, amelynek összes tranzakcióját szeretnénk egy külső fájlba elmenteni.

A folyamat első lépéseként az adatbázisból lekérdezzük az intervallumnak megfelelő adatokat és megjelenítjük a képernyőn. Amennyiben a lekérdezés adott vissza eredményt, akkor az oldalon lévő „Exportálás” gomb elérhetővé válik, amelyre kattintva a fájl mentésre kerül az eszközre. Sikeres mentést követően a kijelzőn egy felugró ablakban értesítjük a felhasználót, hogy a folyamat sikeresen lezárult és megjelenítjük számára a mentett fájl útvonalát, ahol elérheti azt.

```
let path = null;
if (this.platform.is('ios')) {
  path = this.file.documentsDirectory;
} else if (this.platform.is('android')) {
  path = this.file.externalRootDirectory;
}
this.file.writeFile(path, filename, csvdata).then( async () => {
  const alertMsg = await this.alertCtrl.create({
    header: 'Fáj mentése sikeres volt',
    subHeader: 'A fájl itt található: ' + path + filename,
    buttons: ['Ok']
  });
  await alertMsg.present();
}).catch(async (error) => {
  const alertMsg = await this.alertCtrl.create({
    header: 'Valami hiba történt a mentéskor. Kérlek próbáld meg újra.',
    buttons: ['Ok']
  });
  await alertMsg.present();
  console.log(error);
});
```

5.4. ábra – Fájl mentésért felelős kódrészlet

A fenti kódrészlet implementálja a csv formátumra konvertált adat külső fájlba történő mentését. A megoldáshoz a Cordova File plugint használtam, ez a plugin implementálja az összes szükséges folyamatot, hogy képesek legyünk fájlokat olvasni és írni az adott eszközön. Az exportált fájlunk elérési útvonal a futtatott operációs rendszer függvényében változik, ezért ellenőrizve van, hogy épp milyen platformon van az alkalmazásunk használva és ennek alapján állítjuk be a mentés helyét. Csv formátumba kerül exportálásra a fájl, amelyben az adat szövegesen van eltárolva és az egyes értékek vesszővel vannak elválasztva egymástól. A lenti (5.5.) ábrán egy ilyen exportált fájl Microsoft Excel-be importálva látható. A generált fájl első sorában az alkalmazáson belül megadott és lekérdezett időszak jelenik meg. A többi sorban a fejlécnek megfelelően a lekérdezett tranzakciók azonosítója, a bevitelkor megadott leírás és dátum, a tranzakció kategóriája, valamint az összeg és a bevitelkor használt valuta.

B27

	A	B	C	D	E	F	G
1	2022-04-3.-2022-04-19.						
2	ID	Leírás	Összeg	Valuta	Dátum	Kategória	
3	pclweBc7nTLISvnSEZBb	Külföldi bérlet vásárlás	-100	$	2022. április 5.	KÖZLEKEDÉS	
4	JwHqQo5lr9aiRLNkM4cH	Aldi bevásárlás	-20 000	HUF	2022. április 12.	BEVÁSÁRLÁS	
5	pWivZFKNqPRk2tjcuWGk	Fizetés	300 000	HUF	2022. április 4.	FIZETÉS	
6	ubV6PhjTa2fsB4Ep8chW	Ösztöndíj	40 000	HUF	2022. április 4.	FIZETÉS	
7	gvbz8h848bpZ6f7mNmlp	Fizetés	120 000	HUF	2022. április 11.	FIZETÉS	
8							
9							

5.5. ábra – Exportált fájl felépítése

5.3. Emlékeztető beállítása

A felhasználó az alkalmazáson keresztül nyomon tudja követni a számláit egy egyszerű felületen keresztül. Az emlékesztetőhöz először is fel kell töltenie a felhasználónak a számlát az adatbázisba. Ennek elindításához a „Számlák” oldalon a jobb alsó sarokban található plusz gombra kattintva átkerülünk arra az oldalra, ahol megadhatjuk az adatokat. Ebben az űrlapban meg kell adnunk a számlának a befizetési határidejét. A szoftver úgy készült, hogy a befizetési határidő előtti napon egy push üzenet segítségével figyelmezteti a felhasználót a lejáró számláról, amennyiben nem lett megjelölve már korábban befizetettként. Az emlékeztető a számla bevitelével automatikusan létrejön a felhasználónak nem kell külön beállítani semmit.

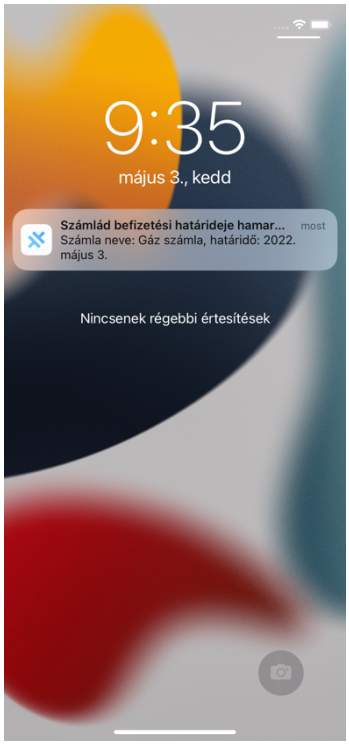
Az első számla feltöltésénél az alkalmazás engedélyt kér, hogy push üzeneteket küldhessen a felhasználó számára. Amennyiben a felhasználó nem engedélyezi ezt, akkor a funkció nem lesz működőképes, de a számlát attól függetlenül rögzítheti az alkalmazásban. Az engedélyeket az operációs rendszer beállításai között később bármikor tudjuk módosítani, így ha korábban engedélyeztük az értesítéseket, de már nem akarjuk használni, akkor le tudjuk tiltani.

```
pushDate.setDate(pushDate.getDate() - 1);
pushDate.setHours(12, 0, 0);
const textStr = "Számla neve: " + invoice.name + ", határidő: " + this.dateChange(invoice.deadline);
this.localNotification.schedule({
  title: 'Számlád befizetési határideje hamarosan lejár',
  text: textStr,
  trigger: { at: pushDate}
});
```

5.6. ábra – Push notification beállítása

A funkció megvalósításához több megoldást is kerestem és próbáltam, végül a Capacitor Local Notification plugin használatával oldottam meg a feladatot. A plugin segítségével könnyen kezelhetőek a push üzenetek, nem szükséges hozzá internet kapcsolat és tudjuk időzíteni, hogy az üzenetek mikor jelenjenek meg. A plugin schedule() metódusának

használatával kerül beállításra az üzenet. Ebben megadjuk az üzenet címét és az megjelenítendő üzenetet, ami ebben az esetben a számla neve és a befizetési határideje. A *trigger* tagban pedig meg van adva a már előre beállított dátum.



5.7. ábra – Az emlékeztető üzenet megjelenése

6. Adatmodell és adat tárolás

Dolgozatom ezen részében be szeretném mutatni a fejlesztett szoftver adattárolási módját és felépítését. Az alkalmazás működéséhez a manapság legjobban elterjedt felhő alapú adattárolást használtam fel, amelyet szeretnék bemutatni és képekkel illusztrálni.

6.1. Felhőalapú adattárolás

A felhő alapvetően kiszolgálók egy globális hálózatot jelenti, amelyben mindegyik kiszolgálónak megvan a saját feladata. A felhőt nem kezeljük entitásként, hanem egy olyan hatalmas hálózatként, amelyek részei össze vannak kapcsolva, ezáltal egyetlen egységes rendszerként működik. Az egyes kiszolgálók speciális feladatra vannak tervezve, vannak amelyek adatokat tárolnak vagy vannak amelyek alkalmazásokat futtatnak. Ezek a szolgáltatások adatközpontokban találhatóak a világ minden táján.

Ahelyett, hogy a fájlokat és az adatokat valamilyen saját fizikai szerverről próbálnánk elérni, azokat online hozzáféréssel az internethez kapcsoló bármilyen eszközről tudjuk kezelni, így az

információ bárhol és bármikor elérhető a felhasználó számára. További nagy előnye a felhő technológiának, hogy nem nekünk kell foglalkoznunk a karbantartással és hibák elhárítása sem a mi feladatunk lesz.

6.2. Firebase Firestore

A Firebase Firestore egy könnyen használható felhőalapú NoSQL adatbázis, amely a Google Cloud Platform fejlesztői csapatával szoros együttműködésben készült. Ennek megfelelően egy teljesen menedzselt szolgáltatásról van szó, a cég elosztott adatbázis rendszere miatt a folyamatos adat elérés biztosítva van az alkalmazás számára. A Firestore valós időben szinkronizálja az adatokat az eszközök között. Képes gyorsítótárazni az adatokat, így ezek offline állapotban is elérhetőek és amint újra online állapotba kerül automatikusan feltölti ezeket az adatbázisba.

A Firestore a Firebase többi szolgáltatásával is együttműködik, például a Firebase Authentication-nel, ezáltal elősegítve a felhasználókezelés gyors és hatékony megvalósítását. Az adatok strukturálásához és lekérdezéséhez a szolgáltatás dokumentumokban és gyűjteményekben tárolja az információkat.

6.3. Mi az a NoSQL?

A NoSQL adatbázisokra „nemrelációs” adatbázisként hivatkozhatunk, hiszen az SQL adatbázisokkal szemben itt nem kell előre definiált sémákat megadnunk, ezért jól használható a nagy mennyiségű strukturálatlan adatokhoz. A NoSQL adatbázisok azzal a céllal jöttek létre, hogy a fejlesztők gyorsan hozhassanak létre adatbázis rendszereket az adatok tárolására és elérhetővé tételére. A relációs adatbázisokban az adatok táblákban vannak eltárolva, a tábla egy sora reprezentál egy adat példányt. Ezzel szemben a legelterjedtebb NoSQL adatbázisok dokumentumokként tárolnak adatokat, ezeknek nem szükséges egységes sémával rendelkezniük, ezért akár több különböző struktúrájú dokumentumot is tudunk egy helyen tárolni. A nemrelációs adatbázisok legelterjedtebb típusai a kulcs-érték páron alapúló tárolás, a dokumentum orientált, az oszlopalapú vagy a gráf típusú adatbázisok.

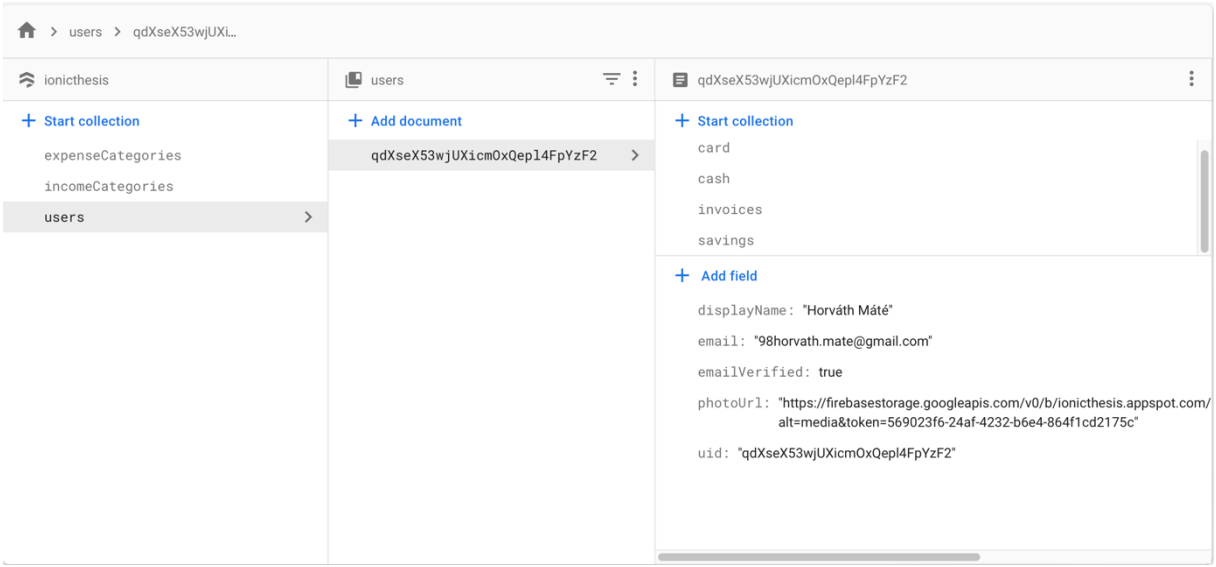
A különböző adattárolási módok mellett skálázhatóság szempontjából is van eltérés az SQL és NoSQL adatbázisok között. A skálázhatóságnak nincs általánosan elfogadott definíciója. Informálisan egy rendszert, akkor nevezünk skálázhatónak, ha az erőforrásait növelve, a rendszer teljesítménye a hozzáadott erőforrásokkal arányosan javul. A skálázhatóságot két fő típusra tudjuk bontani, a vertikális skálázhatóságra és horizontális skálázhatóságra. A

nemrelációs adatbázisok nagy előnye, hogy horizontálisan skálázhatók, ami azt jelenti, hogy a rendszer bővítése érdekében nem egy már meglévő elemét fejlesztjük, hanem egy új fizikai számítógépet adunk hozzá a rendszerhez.

6.4. Adatmodell

A NoSQL adatmodell mintájára a Firestore-ban is dokumentumokban vannak eltárolva az adatok. Ezeket a dokumentumokat a szolgáltatás különböző kollekciókba rendezi, amelynek segítségével a különböző dokumentumokat külön egységekbe tudjuk szervezni, ezáltal létre tudunk hozni lekérdezéseket az adatok szűrésére. Minden dokumentum valamilyen gyűjteménybe kell, hogy legyen szervezve. A kollekciók és a dokumentumok is egymásba ágyazhatók, így tudjuk kialakítani a megfelelő hierarchikus struktúrát az adattároláshoz.

A dolgozatomhoz fejlesztett alkalmazáshoz az adatbázis struktúra a következőképpen lett kialakítva:



6.1. ábra – Az alkalmazás adat struktúrája

A 6.1. ábra szemlélteti az alkalmazás által használt adatbázis strukturális felépítését. A regisztrált felhasználók adatai a *users* kollekcióban vannak eltárolva, mindegyik dokumentum egyedi azonosítója a regisztrációkor generált *uid*-val egyezik meg. Adatmódosításkor vagy törléskor erre az azonosítóra hivatkozva tudjuk módosítani az adatbázist. A dokumentumban kulcs-érték párokként vannak eltárolva a felhasználó profiljához tartozó adatok, itt kerül eltárolásra a felhasználó által feltöltött profilkép elérési útvonala is.

Ezen felül a dokumentumban beágyazásra került négy kollekció is azért, hogy a bankkártyás adatokat, a készpénzes adatokat, a számlákat és a megtakarításokat hatékonyan tudjam tárolni és, hogy a lekérdezések hatékonyságát növeljem. A *card* és a *cash* kollekciókban a felhasználó

által mentett tranzakciók adatai vannak külön-külön dokumentumként eltárolva, az *invoices* kollekcióban találhatóak a rendszerbe felvitt számlák adatai, itt is minden egyes számla külön dokumentumban van tárolva, ezen kívül a *savings* gyűjtemény a felhasználó aktuális megtakarítási céljait tárolja.

A *users* kollekció oszlopában látható az *expenseCategories* és az *incomeCategories* nevű gyűjtemények. Ezekben a rendszerben alapból megjelenő, illetve a felhasználó által feltöltött kategóriák szerepelnek. Mivel a felhasználó egyedi kategóriákat is tud hozzáadni a rendszerhez, ezért a legjobb mód ezeknek az adatoknak az eltárolására, az volt, ha külön kollekcióként kezelem ezeket.

7. Összefoglaló

A szakdolgozatom készítésének és az alkalmazás fejlesztésének kezdetekor még nem ismertem az Ionic keretrendszert, viszont nagyon érdekelt és a mai napig is foglalkoztat a mobil alkalmazásfejlesztés területe, ezért szerettem volna a szakdolgozatomban is egy ilyen témával foglalkozni. A dolgozatomban igyekeztem az általam készített alkalmazást és az általa használt módszereket bemutatni. Mivel a korábbi években az egyetemen is és a szabadidőmben is foglalkoztam már Angular-al, így már volt némi alap tudásom, hogy könnyebben elsajátíthassam az Ionic keretrendszert. Egyre nagyobb teret hódítanak az IT világában a különböző keretrendszerek, ezért az informatikusoknak is elengedhetetlen ezek valamelyikének az ismerete, hogy jól tudjanak érvényesülni a szektorban. Úgy gondolom, hogy a mobil alkalmazásfejlesztés hatalmas növekedésen és fejlődésen ment keresztül az elmúlt években és véleményem szerint ez a növekedés nem fog lelassulni, éppen ezért szerettem volna megismerni az Ionic keretrendszert. A dolgozatommal és az elkészült alkalmazással sikerült egy olyan plusz tudásra szert tennem, amelyet később tudok hasznosítani a munkáim során.

A tényleges fejlesztés megkezdése előtt részletesen tanulmányoztam a keretrendszer dokumentációit, felmértem, hogy milyen lehetőségeket kínál a fejlesztők számára az Ionic és én azt, hogyan tudom használni a készülő alkalmazásomban. Rengeteg videó megnézése és több teszt projekt létrehozása után kezdtem csak neki az alkalmazás fejlesztésének. Az alkalmazás készítésének célja az volt, hogy a szoftvert használó személyek egy olyan termékhez jussanak, amelyben a legfontosabb funkcionalitások megtalálhatóak. A fejlesztés során törekedtem az alkalmazásnak egy egyszerű és letisztult dizájnt létrehozni, hogy minden funkciót a lehető legkönnyebben el lehessen érni. Legnagyobb hangsúlyt a funkciók elkészítése

kapta a munka során, ezért az alkalmazás megjelenése nem túl egyedi, de sikerült egy felhasználóbarát megoldást elkészítenem.

Mivel a fejlesztés megkezdése előtt még nem rendelkeztem megfelelő technikai tudással, ezért a munka során voltak bonyolultabb és összetetteb részek, amelyek megoldása komplikált volt számomra. A készítés során több plugin-t is használtam a funkciók implementálásához, ezért muszáj volt mélyebben beleásni magam ezeknek a dokumentációjába és a kódokba. A fejlesztés elején nagy nehézséget okozott számomra, ezeknek a használata, hiszen mindegyikhez külön dokumentációt kellett tanulmányozni és sokszor a megfelelő használatuk sem volt egyértelmű, de számos fórumot átolvasva mindig sikerült elsajátítanom a megfelelő ismereteket. Idővel egyre könnyebben és gyorsabban kezdtem el megoldani a problémákat és egyre jobb minőségű kódrészeket készítettem.

A program működése szempontjából stabilnak és megbízhatónak mondhat, ám a használat során jelentkezhetnek nem várt hibák, melyek csökkenthetik a felhasználói élményt. A rövid tesztelés során megtalált hibákat sikerült javítani az alkalmazásban, de még maradhattak benne észre nem vett problémák. Véleményem szerint a szoftver jó felhasználói élményt nyújt a használója számára és a legszükségesebb funkciók jelenleg is megtalálhatóak benne, ezért szeretném a jövőben folytatni a projektet. Tervek közt szerepel új funkciókkal bővíteni az alkalmazást és egy egyedi megjelenést létrehozni a még maradandóbb felhasználói élmény érdekében, valamint a későbbiekben esetlegesen felmerülő hibák folyamatos javítása is a célok közt szerepel.

Felhasznált források

1. Angular documentation - https://angular.io/docs
2. Ionic framework - https://ionicframework.com/docs
3. Cloud Firestore - https://firebase.google.com/docs/firestore
4. Felhőalapú számítástechnika: Microsoft Azure - https://azure.microsoft.com/hu-hu/overview/what-is-cloud-computing/
5. Cordova documentation - https://cordova.apache.org/docs/en/latest/
6. Firebase Authentication - https://github.com/angular/angularfire/blob/master/docs/auth/getting-started.md
7. Currency Exchange Rates API - https://github.com/fawazahmed0/currency-api
8. Capacitor documentation - https://capacitorjs.com/docs
9. Angular University: The Firebase Jumpstart Ebook - https://angular-university.io/ebook/EBOOK_FIREBASE
10. Angular Routing - https://angular.io/guide/router
11. Gajdos Sándor: Adatbázisok - 2. függelék: NoSQL adatbázis-kezelők - https://inf.mit.bme.hu/sites/default/files/publications/nosql-konyvfejezet.pdf
12. Koin - https://www.koin.hu
13. Money Pro - https://money.pro/

Nyilatkozat

Alulírott Horváth Máté programtervező informatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztés Tanszékén készítettem, programtervező informatikus 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 Informatikai Intézet könyvtárában, a helyben olvasható könyvek között helyezik el.

2022. május 04. Horváth Máté

.....

aláírás