Szegedi Tudományegyetem
Informatikai Intézet

SZAKDOLGOZAT

Kovács Alex

2023

Szegedi Tudományegyetem
Informatikai Intézet

Multiplatform Mobilalkalmazás Fejlesztése Közösségi Árkövetést Támogató Adatbázissal

Szakdolgozat

Készítette:
Kovács Alex
programtervező informatikus szakos hallgató

Témavezető:
Kiss-Vetráb Mercedes
PhD Hallgató

Szeged
2023

Témakiírás

A szakdolgozat témája a multiplatform mobilalkalmazás fejlesztés. A dolgozat célja, hogy a hallgató megtanulhassa az alapvető fejlesztési folyamatokat egészen a tervezéstől a kész szoftverig. Jelen alkalmazás célja, hogy a felhasználók számára lehetővé tegye saját bevásárló listák létrehozását, megosztását másokkal és a környezetükben megtalálható vásárlási lehetőségek megtekintését.

Az alkalmazás funkciói között szerepel többek között a regisztráció, a profil kezelés, saját listák létrehozása, kedvencek funkció és szűrési lehetőségek. Az alkalmazás célja, hogy megkönnyítse az emberek számára a legolcsóbb termékek megtalálását.

A dolgozat megírása során az alábbi munkafolyamatok elsajátítására van lehetőség: UX/UI tervezés a letisztult és felhasználóbarát végeredményhez, Trello használata a feladatok szervezéséhez, Flutter-Firebase adatbázis kezelése, Flutter alapok elsajátítása, MVVM fejlesztési minta megértése, Git használata.

Tartalmi összefoglaló

Szakdolgozatomként létrehoztam a CleanIT applikációt, mely a felhasználók számára megkönnyíti a mindennapi tisztítószerek és egyéb eszközök bevásárlását és az áraik átláthatóságát. Jelen dolgozatom témáját az utóbbi időben megnövekedett infláció adta, amelynek folyamán egy olyan alkalmazást készítettem el, ahol a különböző termékek árait a felhasználók fel tudják tölteni és nyomon követni. Jelenleg az emberek nap-mint-nap megtapasztalják az üzletekben kapható termékek árának folyamatos változását, emellett akár két üzlet között is hatalmas árkülönbség figyelhető meg. Ebből kifolyólag jutottam arra a döntésre, hogy praktikus lenne egy olyan applikációt megtervezni és létrehozni, ahol a különböző termékek árait a felhasználók fel tudják tölteni és nyomon követni. Én leginkább a közösségi árkövetésre koncentráltam, ahol a felhasználók új termékeket, áruházakat tudnak hozzáadni, majd az adott termékekhez a legfrissebb árakat feltölteni. Ezt a logikát szinte mindenféle terméken lehetne alkalmazni, én most viszont csak a takarítószerekre szűkítettem le az applikációmat. Ugyanakkor szerintem nagyon hasznos lehet élelmiszerek, vagy akár szórakoztatási termékek árának követésére is.

A választásom a Flutterre, mint egy nyílt forráskódú szoftverfejlesztő készletre és a Dart programozási nyelvre esett, mert ennek a két eszköznek a segítségével multiplatformos mobilalkalmazásokat lehet készíteni. Ez azért hasznos számomra, mert csak egy kódbázist kell fenntartani, és nem szükséges külön natív applikációkat fejleszteni. Ez egyben idő és költséghatékony is. Mindemellett gyors és esztétikus applikációkat tudunk vele fejleszteni. Azért választottam a mobilalkalmazást szakdolgozatom témájának, mert a mai világban a legtöbb ember rendelkezik okostelefonnal, ezért a mobilapplikációk használata nagy népszerűségnek örvend. A Flutter szoftverfejlesztő keretrendszert kimondottan jó választásnak tartom, mert segítségével gyorsan lehet alkalmazásokat fejleszteni. Egyetemi tanulmányaim során mindig is érdekelt, hogy hogyan lehet egy applikációt felépíteni teljesen a kezdetektől. A témavezetőm segítségével nagyon sok új eszközt és hasznos dolgokat tanultam meg, amelyeket az applikáció és a dolgozatom elkészítése folyamán kamatoztatni tudtam.

Kulcsszavak: flutter, dart, adatbázis, árkövetés, mobilalkalmazás, multiplatform

Tartalomjegyzék

1. Alkalmazás célja, motivációja

Az elkészített alkalmazásom célja az, hogy megkönnyítse a tisztítószerek vásárlását és ennek az eszköznek a segítségével a felhasználók jelentős mennyiségű pénzt tudnának megtakarítani. Ehhez természetesen a felhasználói interakciók is szükségesek, hogy az adatbázisban minél nagyobb mennyiségű adat álljon rendelkezésükre. Minél több termék és a termékekhez tartozó ár kerül be, annál hatékonyabb tud lenni az applikáció, mert a bevásárlólista funkció segítségével több üzletben is láthatják a végösszegeket.

Személy szerint nekem mindig is fontos volt, hogy a vásárlásaimat a legköltséghatékonyabb módon tudjam teljesíteni. Viszont mindig nagyon sok időbe telt megtalálni a legjobb árakat, vagyis azokat az üzleteket, ahol a legjobb áron tudok hozzájutni különböző termékekhez. Sajnos ez nem mindig sikerül, mert lehet, hogy az egyik áruházban jelentősen tudunk spórolni egy terméken, de lehet, hogy egy másik viszont lényegesen többe kerül. Az pedig, hogy körbejárjuk az összes környező üzletet, nagyon időigényes tud lenni. A mai rohanó világunkban sajnos nehezen tehetjük meg azt, hogy akár 8-10 üzletet is meglátogassunk a legjobb árak reményében. Emellett az online szórólapok átnézése is időigényes folyamatnak minősül. Emiatt hoztam létre egy olyan applikációt, ami kiszámolja és megmutatja nekünk, hogy melyik üzletben járhatunk a legjobban a végösszeget tekintve.

A megalkotott alkalmazásom az MVVM architektúra alapján a különböző programozási paradigmákat követi. Számomra kezdetben a Flutter és a Dart teljesen ismeretlen volt. Az ismereteimet először egy Udemy-n elérhető kurzussal bővítettem. Ez azért is volt nagyon hasznos, mert megtanultam az alapokat, hogy hogyan épül fel a Flutter, hogyan kell widgeteket és új képernyőket létrehozni. Egészen az alapoktól a kurzus vége felé eljutottam haladó technikákig is. Mivel ez volt az első komolyabb alkalmazás, amit egyedül készítettem el, az elején sok akadályba ütköztem, de ilyenkor mindig a legjobb tudásom szerint próbáltam tovább haladni. Ugyanakkor nem csak a programozás részével kellett foglalkoznom, hanem egyéb technológiákat is kellett alkalmaznom. A tervezési fázisban először is meg kellett terveznem az alkalmazásom prototípusát. Ez is egy új terület volt nekem, mert ezelőtt még nem foglalkoztam UI tervezéssel. Nagy segítségemre volt a proto.io, ahol szinte egy kész alkalmazást tudtam megtervezni, úgy, hogy egy sor kódot sem kellett írnom. Itt utánanéztem a különböző tervezési mintáknak és azokat követve próbáltam megvalósítani az alkalmazásom prototípusát. Ez a későbbiekben nagy segítségemre volt, mert így könnyebben tudtam haladni a fejlesztési fázisban. Az elkészített prototípust könnyen ki tudtam exportálni, így ezzel elkészültem a

képernyőképekkel. A képernyőtervek után jöhettek a Moodboardok elkészítése. Ami azért is volt nagyon hasznos, mert ebben a fázisban meg tudtam határozni az applikációm stílusát és a használt színeket, stílusokat. A fejlesztési folyamatot a Trello és a Github segítésével vittem végig. A Trello-ban az elején elkészítettem a teljes alkalmazásfejlesztési ütemtervet. Itt először is listákat kellett létrehoznom, majd csak ezután tudtam jegyeket létrehozni. Mindegyik jegy egy funkció megvalósításához tartozott az applikációban. A Github verziókövető rendszer segítségével tudtam rendszerezni az alkalmazásom kódbázisát. Az elején voltak vele nehézségeim, de a végére nagyon sok új és hasznos Git parancsot tanultam meg, amelyeket a későbbiek folyamán tudok majd hasznosítani a munka világában egyaránt.

2. Mobilalkalmazás fejlesztése

2.1 Natív és multiplatform fejlesztői eszközök története

Mivel a fejlesztők a natív alkalmazásfejlesztés során egy adott platformra kódolnak, teljes mértékben hozzáférnek a natív eszközfunkciókhoz. Ez általában jobb teljesítményt és sebességet eredményez a több platformos alkalmazásfejlesztéshez képest [1].

Másrészt, ha egy alkalmazást több platformon szeretne elindítani, a natív alkalmazásfejlesztés több kódot és több fejlesztőt igényel. E kiadások mellett a natív alkalmazásfejlesztés megnehezítheti a különböző platformokon egyidejűleg, konzisztens felhasználói élményt nyújtó indítást. Itt lehetnek hasznosak az olyan több platformos alkalmazásfejlesztési keretrendszerek, mint a Flutter. Natív alkalmazások is nagy mennyiségben fordulnak elő. Segítségükkel teljesen kihasználhatjuk az adott operációsrendszer adta lehetőségeket. Androidon az Android NDK segítségével fejleszthetünk natív alkalmazásokat. iOS-en pedig a SwiftUI és Objective-C áll rendelkezésünkre. Android alapú alkalmazásokat az összes operációsrendszeren tudunk fejleszteni. Natív iOS alapú alkalmazás fejlesztéséhez szükségünk van egy MacOS operációsrendszerű eszközre. Ugyanakkor egy startup számára kitűnő választás lehet egy multiplatformos keretrendszer választása, mert ezzel időt és pénzt spórólhatnak. Nem beszélve arról, hogy ezzel a fenntartási költségeiket is csökkenthetik, mert csak egy kódbázist kell fenntartaniuk. Sokszor azért fejlesztenek inkább natív applikációkat, mert gyorsabbnak tartják, ugyanakkor napjainkban egyre nagyobb térhez jutnak a multiplatform alapú alkalmazások. Utóbbi időben több keresztplatformos keretrendszer is jelentős népszerűségnek örvend [1].

A szakdolgozatom tervezési fázisában én is több ilyen keretrendszer közül választhattam. A három legelterjedtebb közül, melyek a Flutter, React Native és a Xamarin, végül a Flutter-re esett a választásom, mert ebben a szoftverfejlesztői csomagban nem csak mobilapplikációkat tudok készíteni, hanem ugyanazt a kódbázist fel tudom használni más platformok alkalmazásaihoz is. A Flutter keretrendszer a Dart programozási nyelvet használja [2].

2.2 Flutter és Dart

A Flutter egy nyílt forráskódú keretrendszer, amelyet a Google fejlesztett ki és támogat. A frontend- és full-stack fejlesztők a Fluttert arra használják, hogy egyetlen kódbázisból több platformra is elkészítsék egy alkalmazás felhasználói felületét (UI).

Amikor a Flutter 2018-ban elindult, elsősorban a mobilalkalmazások fejlesztését támogatta. A Flutter ma már hat platformon támogatja az alkalmazásfejlesztést: iOS, Android, web, Windows, MacOS és Linux. A szakdolgozatomként egy iOS, Android multiplatformos mobilalkalmazást készítettem el. Az alkalmazásomat ezeken az operációsrendszereken teszteltem. A Flutter leegyszerűsíti a konzisztens, vonzó felhasználói felületek létrehozásának folyamatát egy alkalmazás számára az általa támogatott hat platformon. A Flutter egy platformokon átívelő fejlesztési keretrendszer. A Flutter a Dart programozási nyelvet használja, amely először natív platformfüggő kódra fordul, majd ezt követően lesz belőle gépi kód. A fogadóeszközök megértik ezt a kódot, ami gyors és hatékony teljesítményt biztosít. A Google a Flutter-t úgy fejlesztette, hogy a hangsúlyt a könnyű használatra helyezte. Az olyan eszközökkel, mint a hot reload, amely lehetővé teszi, hogy a fejlesztők által írt widget kód eredményét a teljes kódbázis újrafordítása nélkül, futási időben is ellenőrizni tudják. Ez a funkció nagyban megkönnyítette a fejlesztési fázist, mivel egy-egy változtatás után azonnal láttam az eredményt a futtatott emulátoron. Így sokkal egyszerűbb dolgom volt a UI és a funkciók kialakítása közben. Más eszközök, például a widget inspector megkönnyítik a felhasználói felület elrendezésével kapcsolatos problémák vizualizálását és megoldását.

A Flutter és Dart számomra teljesen új volt. Ezért az elején kisebb nehézségekkel kellett megküzdenem. Meg kellett értenem, hogy hogyan is épül fel egy Flutter alkalmazás. A widgeteket jobban megismerve lett könnyebb az alkalmazás fejlesztése. A Flutter-ben a fejlesztők a felhasználói felület elrendezéseit widgetek segítségével építik. Ez azt jelenti, hogy minden, amit a felhasználó a képernyőn lát, az ablakoktól és a panelektől kezdve a gombokig és a szövegig, widgetekből áll. A Flutter widgeteket úgy tervezték, hogy a fejlesztők könnyen testre szabhassák őket. A Flutter ezt egy kompozíciós megközelítéssel éri el. Ez azt jelenti, hogy a legtöbb widget kisebb widgetekből áll, és a legalapvetőbb widgeteknek speciális céljaik vannak. Ez lehetővé teszi a fejlesztők számára, hogy új widgeteket kombináljanak vagy szerkesszenek. A Flutter a közösség által fejlesztett widgetek használatát is megkönnyíti. A Flutter csomagkezelője támogatja, hogy több widgetkönyvtárral rendelkezzen, és a Flutter ösztönzi a fejlesztői közösséget, hogy újakat készítsen és tartson fenn. A Flutter a nyílt forráskódú Dart programozási nyelvet használja, amelyet szintén a Google fejlesztett ki. A Dartot UI-k építésére optimalizálták, és a Dart számos erősségét a Flutter is felhasználja. A Dart egy fejlesztőbarát nyelv gyors alkalmazások fejlesztésére bármilyen platformon. Ez a programozási nyelv képezi a Flutter alapját is, amely számos alapvető fejlesztői feladatot is támogat, például a kód formázását, elemzését és tesztelését [1].

3. UX és UI design

3.1 Történeti áttekintő

A felhasználói élménytervezés egy konceptuális tervezési tudományág, amelynek gyökerei az emberi tényezőkben és az ergonómiában gyökereznek, egy olyan területen, amely az 1940-es évek vége óta az emberi felhasználók, a gépek és a kontextuális környezet közötti kölcsönhatásra összpontosít, hogy a felhasználói élményt szolgáló rendszereket tervezzen. Donald Norman professzor, a formatervezés, a használhatóság és a kognitív tudományok kutatója alkotta meg a "felhasználói élmény" kifejezést, és tette szélesebb körben ismerté azt [3].

A UI azokat a képernyőket, gombokat, kapcsolókat, ikonokat és egyéb vizuális elemeket jelenti, amelyekkel egy weboldal, alkalmazás vagy más elektronikus eszköz használata során interakcióba léphet a felhasználó [4].

A UI feladata a gépek és szoftverek, például számítógépek, háztartási gépek, mobileszközök és egyéb elektronikus eszközök felhasználói felületeinek tervezése, amelynek középpontjában a használhatóság és a felhasználói élmény maximalizálása áll [3].

A felhasználói élménytervezés (UX) az a folyamat, amelyet a tervezőcsapatok használnak olyan termékek létrehozására, amelyek értelmes és releváns élményt nyújtanak a felhasználóknak. Az UX-tervezés magában foglalja a termék megszerzésének és integrálásának teljes folyamatát, beleértve a márkaépítés, a design, a használhatóság és a funkció szempontjait. Bár a UI minden bizonnyal hatással lehet az UX-re, a kettő mégis különbözik egymástól [3].

3.2 Főbb, alkalmazott tervezési szabályok

A UX (felhasználói élmény) tervezési folyamata öt szakaszra osztható: EMPÁTIA, MEGHATÁROZÁS, ÖTLETALKOTÁS, TESZTELÉS és PROTOTÍPUS KÉSZÍTÉS. Ezek a szakaszok gyakran ebben a sorrendben követik egymást, és fontos tudni, hogy a UX egy iteratív folyamat. Ezt a folyamatot szem előtt tartva jobb applikációt tudunk építeni a felhasználó számára [3].

Empátia: Ez a UX tervezési folyamat első szakasza, ahol meg kell találnod, milyen kihívásokkal szembesülnek az applikáció felhasználói, különböző feladatok elvégzése közben. Itt először is ki kellett találnom, hogy hogyan is szeretném megvalósítani az alkalmazást és miért lehet hasznos más felhasználó számára. Felhasználói szemszögből kellett átlátnom az

applikációm részeit. Át kellett gondolnom azt is, hogy a felhasználók milyen funkciókat tudnak beleképzelni az alkalmazásba és milyen logika alapján lehetne hasznos számukra.

Meghatározás: Itt megpróbáltam különböző részekre szétbontani az alkalmazásomat. Először is a legfontosabb részekre koncentráltam, ennek a logikáját próbáltam kitalálni. A nehézségek akkor adódtak, amikor több részt kellett összekapcsolnom. Ezekhez az összetett részekhez több idő kellett mire rájöttem, hogy hogyan is lehetne megfelelően alkalmazni.

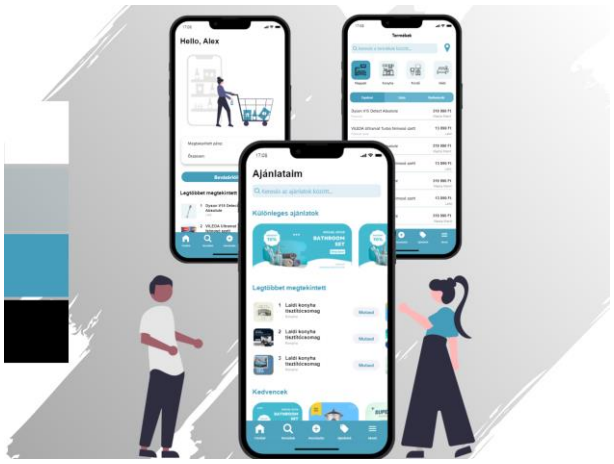
Ötletalkotás: Ebben a szakaszban próbáltam minél több ötletet összegyűjteni. Tudtam, hogy nem tudom megvalósítani mindegyiket, de így legalább volt miből válogatnom a későbbiekben. Ugyanakkor, ahogy egyre több és több megvalósítási példa jutott eszembe, úgy néha a korábbiakon is változtatásokat kellett végeznem, hogy összhangban legyenek.

Prototípus: A prototípus készítési folyamat nagyon lényeges volt a tervezési fázisban. Itt az ötleteim alapján próbáltam megvalósítani az alkalmazásom kinézetét. Több olyan funkció is volt, ami ötletként jól hangzott, de a prototípusban már nem mutatott megfelelően. Ezért volt jó, hogy az előző szakaszban több ötletem is volt és ezekből könnyedén tudtam válogatni.

Tesztelés: A tesztelés rész azért fontos, mert ezáltal észrevehetjük a hibákat a tervezési fázisban, és láthatjuk, hogy az eddig létrehozott tervek mennyire felelnek meg az elvárásainknak. A prototípusomban teszteltem a navigációt a képernyők között, így láthattam, hogy hogyan függenek össze az alkalmazásom részei.

4. Moodboard és logó

4.1 Moodboard



1. ábra: Moodboard

A moodboard a koncepciótervezés vizuális megjelenítése inspirációk gyűjteménye formájában. Az elnevezés tulajdonképpen elég pontos, mivel lényegében a tervező által összeállított hangulattábláról van szó. A moodboardok lényege, hogy a koncepció megalkotásának szakaszában rendszerezze az inspirációkat, legyen szó akár egy mobilalkalmazásról vagy bármilyen más kreatív munkáról - belsőépítészet, divat stb.

A moodboard lehet digitális és/vagy fizikai formában is. Az első esetben kiválasztott anyagokból készítünk kollázst szoftver segítségével, az "analóg" tábla esetében használhatunk papírt, parafatáblát vagy falat, ahová magazinkivágásokat, anyagokat, fotókat stb. ragasztunk.

A moodboard nagyon hasznos lehet az ötletek rendszerezésében. Azzal, hogy az összes koncepciót és inspirációt egy helyre helyezzük, gyorsan és hatékonyan ellenőrizhetjük a kompozíciót és annak harmóniáját. Kiderülhet, hogy a javasolt színek, minták vagy motívumok nem feltétlenül harmonizálnak egymással - egy hangulattábla segít abban, hogy már a kezdetektől fogva meghatározzuk és megtartsuk az irányt, és időt takaríthatunk meg a további szakaszokban [5].

2. ábra: Moodboard

A tervezéskor különböző moodboard képeket valósítottam meg. Ezek a képek megmutatják az alkalmazásban használt színeket, dizájnokat. Ugyanakkor ezekkel a képekkel megmutathatjuk az alkalmazásunk témáját és hangulatát. Könnyen átláthatjuk vele az applikációnk terveit, stílusait. Különféle dizájn elemeket is megjeleníthetünk ezeken a képeken. Ezeket a képeket a Canva képszerkesztő segítségével valósítottam meg. A kiexportált képernyőképek nagy segítségemre voltak, mert ezeket könnyen be tudtam illeszteni egy-egy moodboard-os képhez. Igyekeztem a jelentősebb funkciókat megjeleníteni, amelyek fontos részét képezik az alkalmazásomnak. Olyan képeket is készítettem, amelyek az alkalmazás témájára voltak kihegyezve, vagyis a takarítószeres motívumok is megjelentek rajta, ezzel is mutatva azt, hogy milyen témában fog majd elkészülni az alkalmazásom.

4.2 Logó

A logó elkészítését nagyban megkönnyítette a tervezési fázisban létrehozott kezdőképernyő kép, mert ebből fel tudtam használni az egyik képet a logóhoz. A tervezés kezdetén próbáltam minél összetettebb képet létrehozni, szöveggel és több kép megjelenítésével, de gyorsan rájöttem, hogy logónak egy egyszerű, de annál feltűnőbb motívumot kell választani.

3. ábra: Applikáció logója

Ezért esett a választásom a kezdőképernyőn szereplő kesztyűkre, mert ezzel a rózsaszín színnel egyedi és feltűnő megjelenést ad az alkalmazásnak. A logó tervezését Canvában hajtottam végre, ez egy képszerkesztő eszköz, ami nagyon megkönnyíti a képekkel való munkát. Ügyeltem arra, hogy a terveket 1024x1024 felbontásban készítsem el. Ezután az appicon.co weboldalt használtam, ahol le tudtam generálni az alkalmazás ikonjait. Ezen az oldalon több platform közül is választhattam. A megfelelő felületek kiválasztása után egy zip fájlt tudunk letölteni. Ezt a zip fájlt kicsomagolva és az Android mappát megnyitva tekinthetjük meg az elkészült ikonokat többféle felbontásban, ami ahhoz kell, hogy a logó megfelelően jelenjen meg például az alkalmazás áruházban. Ezeket a mappákat a Flutter projekten belül az android > app > src > main > res mappába kellett helyeznem. Figyelnem kellett arra, hogy a mappákban PNG kiterjesztésű fájloknak a neve megegyezzen, mert ez az applikáció buildelésénél gondot okozhat. Ha mégis megváltoztattuk ezeknek a fájloknak a nevét, vagy már úgy generáltuk le, akkor az android > app > src > main > AndroidManifest.xml fájlban az android:icon=”@mipmap/ic_launcher” nevét meg kell változtatnunk majd újrabuildelés után megoldódik a probléma.

Az első ilyen változtatás után viszont nem az elvárt eredményt kaptam. Sokkal kisebb volt az applikáció ikonja, mint ahogy arra számítottam. Többszöri próbálkozásra sem sikerült megoldani, mikor találtam egy weboldalt, ahol le volt írva, hogyha a generált kép négyzet alakban, vagyis nem lekerekítve lett létrehozva, akkor az Android operációs rendszeren kisebb méretűként jelenik meg. Ezt úgy lehet orvosolni, hogy az ikont már Canvából úgy exportálom ki, hogy a sarkai le legyenek kerekítve, de ez még nem elég ahhoz, hogy tökéletes megjelenést biztosítsunk az applikációnknak. Ezt a képet átlátszó háttérrel kellett kiexportálni, hogy fehértől eltérő háttér esetén is megfelelően jelenjen meg. Ha még professzionálisabb logókat szeretnénk tervezni, akkor a segítségünkre lehet az appicons.ai weboldal, ahol mesterséges intelligencia segítségével tudunk szebbnél szebb ikonokat generálni.

5. Képernyőtervek

Az alkalmazásom tervezését először a képernyőtervekkel kezdtem. Ehhez egy kiváló eszközt, a proto.io-t használtam. A proto.io egy iparágvezető webes prototípuskészítő platform, amely alkalmas UX-tervezők, vállalkozók, termékmenedzserek, marketingesek és bárki számára, akinek van egy nagyszerű ötlete. A proto.io segítségével mindenki pillanatok alatt életre keltheti ötletét. Nincs szükség különleges tervezési ismeretekre: a proto.io komponenskönyvtárak és sablonok segítségével a tervezés olyan egyszerű, mint a drag-and-drop. Könnyen hozzá lehet adni bármit, az egyszerű kattintási pontoktól kezdve a hatékony interakciókig és gyönyörű animációkig, ezzel egy olyan prototípust létrehozva, ami teljesen valós applikációnak tűnik.

Ennek az eszköznek a segítségével interaktív prototípusokat lehet készíteni, megosztani, bemutatni és tesztelni különféle képernyőméretekre: mobil, táblagép, web, TV, okosóra. Könnyen és gyorsan használatba lehet venni ugyanis nem kell telepíteni, és nem kell fájlokat mozgatni: egy teljes körű, webalapú prototípuskészítő platform. Ehhez kapcsolódva szinte teljesen fel tudtam építeni az alkalmazásomat, anélkül, hogy egy sor kódot le kellett volna írnom. Számomra ez nagy segítség volt az elején, mert így nem kellett azon izgulnom, hogy minél gyorsabban elsajátítsam az alkalmazásom fejlesztéséhez szükséges eszközöket [6].

A tervezést úgy kezdtem, hogy az interneten próbáltam minél több mobilapplikáció tervet megtekinteni és ezekből próbáltam ötleteket meríteni. Kezdetben nem ment olyan egyszerűen a képernyők létrehozása, mert egy elég összetett eszköztár fogadott. Első prototípus elkészítése után, ami megadta az applikációnak a vázát, jöhettek a javítások. Azért is jó volt használni ezt az eszközt, mert különböző funkciókat is tudtam tesztelni, mint például az oldalak közötti navigációt.

Miután elkészültem a tervekkel, könnyen ki lehetett exportálni egy HTML fájlba, ami megnyitva egyből megjelenített egy mobileszközt az applikációval. A képernyőterveket külön képként is ki lehet exportálni, ami a Moodboard létrehozásában segített.

6. Fejlesztési folyamat

6.1 Trello

A Trello rugalmas eszköz a munkák menedzseléséhez, amelyben a csapatok vizuális, produktív és eredményes módon készíthetnek terveket, dolgozhatnak együtt a projekteken, szervezhetik a munkafolyamatokat és nyomon követhetik az előrehaladást. A Trello segít kézben tartani a közös munka és az eredmények elérésének jelentős mérföldköveit és napi feladatait is az ötleteléstől a tervezésen át a kivitelezésig [7].

A Trello nagyon sokat segített a munkafolyamat megfelelő követésében. Segítségével rendezetten tudtam tartani a projekttel kapcsolatos feladatokat és a témavezetőmnek is könnyebb volt ellenőrizni az elvégzett és a hátralévő feladataimat. Az alkalmazásom készítésének az elején voltak nehézségeim a használatával, ugyanis ezelőtt még nem használtam ilyen alkalmazást. Elsőre szokatlan volt a felhasználói felülete, mivel elég sok funkciót tartalmazott. Ezeket a funkciókat jobban megismerve lett egyre gördülékenyebb a használata. Egy idő után már teljesen automatikus volt a használata számomra.

A szakdolgozatomhoz a táblát Trelloban még a tervezési folyamatok során készítettem el. Ezen a táblán nyomon követhetem az információkat és a változásokat. Ehhez a táblához listákat hoztam létre. Ezekben a listákban tudtam létrehozni a kártyákat. A listákhoz a munkafolyamat különböző részeit adtam hozzá. Ezeket a következőként neveztem el: Bug, Backlog, Work in Progress Code Review, Done. A tervezési fázisban a Backlog részbe kellett először felvennem a kártyákat, a megvalósítandó funkciók alapján. Ez az elején nagy tervezést igényelt, mert szét kellett bontanom az alkalmazásomat külön álló részekre. Azt is meg kellett néznem, hogy vannak-e olyan képernyők, ahol hasonló widgetek vannak. Ezzel kiszűrtem azokat a részeket, ahol azonosság van, így nem kell őket többször megvalósítani. Minden képernyőhöz két jegyet vettem fel. Az egyik az üzleti logikát jelentette, a másik pedig a UI-t. Ezekhez a kártyákhoz létrehoztam különböző színű címkéket, ezzel jobban átláthatóvá téve az összeillő részeket. Fontossági sorrend szerint is létrehoztam címkéket. Ezekből háromféle volt, alacsony, normál, sürgős. A UI címkével ellátott kártyákat sürgősnek, az üzleti logikával ellátott kártyákat pedig normálnak állítottam be. Amikor elkezdtem dolgozni egy funkción, akkor a hozzátartozó kártyát a Backlogból a Work in Progress listába tettem. Voltak olyan kártyák, ahol egy tennivalók listát is létrehoztam, így könnyebben tudtam figyelni az elvégzendő feladatokat, sikeresen teljesítés után pedig elvégzettként tudtam jelölni. A Trelloban egy olyan funkció is elérhető, amit a fejlesztés során sokat használtam. Minden pull

requestet hozzá tudtam kapcsolni egy kártyához, ezáltal könnyen elérhetővé vált az adott pull request. Miután elkészültem egy funkcióval. a kártyáját áttettem a Code Review részbe. Ez azt jelenti, hogy a témavezetőmnek át kell néznie, majd sikeres ellenőrzés és a pull request elfogadása után át lehetett helyezni a Done listába.

Bug jegyeket akkor vettem fel, amikor egy nem várt működést tapasztaltam egy olyan részen, ami már be lett olvasztva a master ágra. A bug jegynek az alábbi információkat kellett tartalmaznia; hol van a hiba és kép az adott hibáról, hova kell kattintani, vagy mit kell csinálni, hogy előjöjjön, mi lenne az elvárt működés vagy kinézet. Így ezeket a hibákat könnyedén tudtam javítani, mert tudtam, hogy hol jön létre a hiba. Ezekhez a jegyekhez is mindig egy új ágat hoztam létre.

6.2 GitHub

A fejlesztéshez a GitHub verziókövető rendszert használtam. A GitHub egy webes felület, amely valós idejű együttműködést tesz lehetővé. A GitHub segítségével könnyedén nyomon követheti a változásokat és navigálhat a verziók között. A Git a GitHub által használt verziókezelő rendszer. A Git nyílt forráskódú és ingyenesen használható kis és nagy projektek számára. Ez a rendszer követi nyomon a GitHubban végrehajtott minden változtatást [8].

Ez nagyban megkönnyítette a fejlesztési fázist, mert így tisztán és átláthatóan volt tárolva a kódom. Korábban már használtam GitHubot, viszont ennek az alkalmazásnak a fejlesztése során tanultam meg igazán a használatának a fontosságát. Fejlesztés során úgy éreztem magam, mintha egy valós projekten dolgoznék. Nagyon sok olyan álláshirdetést lehet találni mostanában, ahol a Git használatát szinte alapnak veszik. Számomra ezért is volt fontos, hogy mélyebben megismerkedjek ennek az eszköznek a helyes használatával. A fejlesztés git feature-branch munkafolyamat alapján zajlott, ahol minden új funkcióhoz egy új branchet kellett létrehoznom. Ez azért volt fontos, mert így csak a megfelelően megvalósított és működő funkciókat tudtam hozzáadni a master branchez. Így, ha egy új funkció létrehozásánál valami nem működött megfelelően, vagy bugokat észleltünk, akkor ezeket még azelőtt tudtam javítani, hogy egyesítettem volna a master branchez. Egy új branch létrehozása és megvalósítása úgy zajlott, hogy először is a masterből létrehoztam egy új branchet a git checkout -b new-feature paranccsal. Fontos, hogy mindig a master legfrissebb verziójával dolgozzak. Abban az esetben, ha az új branchet nem a masterből hozom létre, akkor az egyesítés előtt egy rebase-t is el kell végezni. Sikeres létrehozás után jöhetett a funkció megvalósítása. A módosított vagy újonnan hozzáadott fájlokat a git status parancs segítségével tudtam megtekinteni. Az elején ennek a

parancsnak a használata sokszor hasznos volt, mert így ki tudtam szűrni azokat a generált fájlokat, amelyekre nem volt szükségem és így könnyen tudtam frissíteni a .gitignore fájlt. Ennek a fájlnak a megléte fontos volt, mert így csak a nemgenerált fájlokat töltöttem fel GitHubra, ezzel jelentősen csökkenthetjük a fentlévő könyvtárnak a méretét. A Flutter a projekt létrehozásakor automatikusan létrehoz egy ilyen fájlt, de ez a fájl nem tartalmazza az összes olyan feltételt, ami segít kiszűrni a generált fájlokat. Ezért 2 további sort is bele kellett írnom az alapértelmezetten generált fájlhoz. **/flutter/generated_plugin_registrant. ***/flutter/generated_plugins.cmake Így már csak tényleg azok a fájlok kerülnek fel, amelyek nem generálva vannak. A Flutter egyszerre több .gitignore fájlt generál le az operációs rendszer specifikus mappákban. Ezeket töröltem, mert arra törekedtem, hogy csak egy ilyen fájlom legyen a projekt gyökérkönyvtárában. Ha további fájlokat szeretnénk figyelmen kívül hagyni, akkor csak egy új sort kell hozzáadnunk, ami tartalmazza a feltételt.

A fejlesztési folyamat úgy zajlott, hogy egy új funkció létrehozásakor létrehoztam egy új ágat. Ezt általában a főágról, a masterről származtattam le. Ez akkor volt hasznos, amikor egy teljesen különálló funkciót akartam hozzáadni az alkalmazásomhoz. Azokban az esetekben, amikor viszont szükségem volt egy-egy branch előző állapotára, akkor az adott branchből hoztam létre az új ágat. Ez sokat segített, mert így könnyebben tudtam tesztelni az új funkciót, a már meglévők használatával. Funkciók megvalósítása közben próbáltam minél többet commitolni, hogy könnyebben visszakövethetőek legyenek a változtatások. Miután elkészültem, kellett tartanom egy önellenőrzést a feltöltés előtt. Ehhez egy lista lett létrehozva GitHub Wiki-ként, ahova azokat a pontokat írtam, amelyeket ellenőriznem kellett a feltöltés előtt. Néhány pont, amelyet figyelembe kellett vennem az önellenőrzés folyamán: Padding használata SizedBox helyett, felesleges megjegyzések, vagy extra kódmagyarázó sorok, MediaQuery helyett Expanded vagy FractionallySizedBox használata, a szövegek kiszervezése egy külön fájlba. Ez az ellenőrzés nagyban megkönnyítette a témavezetőm dolgát, amikor átnézte a változtatásaimat. Ezután az következett, hogy a változtatásokat feltöltöttem a GitHub távoli tárolójába. Mindegyik branchez létrehoztam egy új pull requestet. Ez azért kellett, mert nem egyből a masterre töltöttem fel az új funkciókat. A master branchre beállítottam egy olyan szabályt, hogy legalább egy jóváhagyás szükséges ahhoz, hogy pull request-ben össze tudjam olvasztani az új ágat a masterrel. Így erre a főágra már csak azok a funkciók kerültek be, melyek átnézve és ellenőrizve voltak.

7. Alkalmazás szerkezeti felépítése

7.1 MVVM

Alkalmazásomat a Model-View-ViewModel (MVVM) architektúra alapján készítettem el. Ennek az architektúrának a segítségével külön tudjuk szedni az alkalmazás UI részét az üzleti logikától. Az MVVM segítségével a View-ből a ViewModel-be tudjuk kiszervezni az üzleti logikát. Így a View csak a megjelenítéssel tud foglalkozni. Több előnye is van az MVVM alkalmazásának:

Karbantarthatóság: Mivel az alkalmazás View és üzleti logika részét külön tudjuk választani, így a kódunk könnyen karbantarthatóvá és újrafelhasználhatóvá válik.

Tesztelhetőség: A View és a ViewModel elválasztása miatt könnyebben tudjuk tesztelni az üzleti logikát. Ezáltal alkalmazásunk tesztelhetősége jelentős mértékben javul.

Bővíthetőség: Ennek az architektúrának az alkalmazásával az applikációnk könnyen bővíthetővé válik. Ami a későbbi változtatásokat nagyban megkönnyíti majd.

A Model segítségével kapcsoljuk össze a View-t és a ViewModel-t. Segítségével valós időben tudjuk tárolni a lekérdezési adatokat, vagy az adatbázissal kapcsolatos lekérdezéseket.

A ViewModel a Model és a View között van. Segítségével tudjuk kezelni a felhasználói eseményeket és az üzleti logikát. A ViewModel segítségével adatokat tudunk lekérni a Model-től, amit a View-ben tudunk megjeleníteni. Itt tároljuk az adatbázisból lekérdezett adatokat, amelyeket a View-be közvetít tovább.

A View csak a megjelenítéssel foglalkozik. Itt jelennek meg a felhasználó számára látható widgetek. Felhasználói interakció hatására a View jelzi a ViewModel számára a létrejövő eseményt, amit majd a ViewModel fog elvégezni. A ViewModel-ből kapja meg az adatokat és azokat jeleníti meg. Az MVVM-et manapság széles körben alkalmazzák, mivel támogatja az eseményvezérelt megközelítést, ami Flutter alkalmazásoknál különösen hasznos lehet, mert a komponensek nagy része események alapján történik [9].

7.2 Adatbázis

Az alkalmazásom egyik fontos része az adatbázis, amelynek a terveit a prototípus elkészítése után valósítottam meg. Fontos volt, hogy meg legyenek tervezve az adatbázis táblák, mert addig nem tudtam foglalkozni az alkalmazásom adatbázis részével. Az applikációmat a Firebase segítségével hostoltam. Ennek a platformnak a segítségével könnyedén

megvalósítottam a felhasználók regisztrációját és bejelentkeztetését. Mindemellett a Cloud Firestore adatbázist is tudtam használni az adatok tárolására. A Cloud Firestore egy NoSQL adatbázisszolgáltatás, amelynek segítségével könnyedén tárolhatunk, szinkronizálhatunk és lekérdezhetünk adatokat mobil- és webes alkalmazásaink számára - globális szinten.

Ennek a szolgáltatásnak a tulajdonságait figyelembevéve terveztem meg az adatbázis tábláit. Mint ahogy az 1.számú mellékletben lévő diagramon is látható, az adatbázis 10 táblából áll. Ezek a táblák között többféle kapcsolat is megtalálható. Mindegyik tábla tartalmazott egy ID-t, ezzel megkönnyítve az adatok tárolását. A Firestore minden új dokumentumhoz egy egyedi ID-t hozott létre, így a táblák könnyen összekapcsolhatóvá váltak.

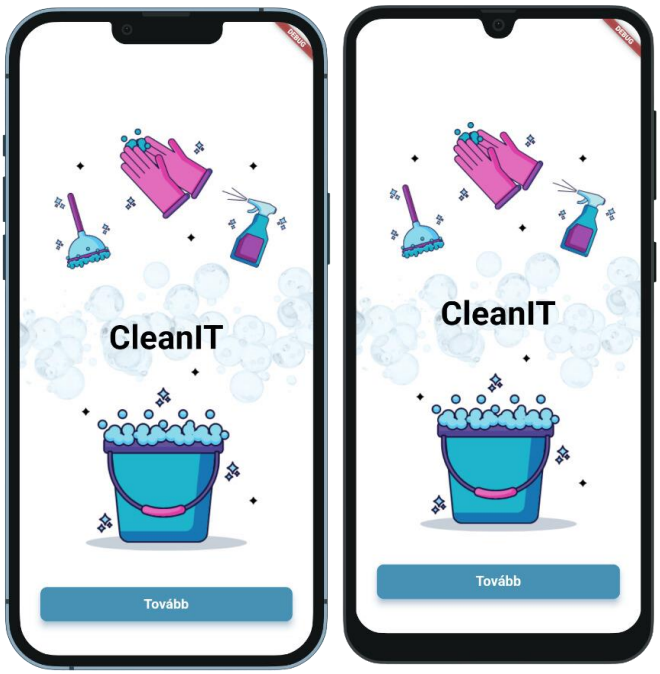
Az MVVM architektúra alapján ezért létre kell hozni egy Service osztályt is, ami az adatbázis műveletekkel foglalkozik. A Service osztályban szereplő metódusokat a ViewModel hívja meg és kezeli a lekérdezések eredményeit. A ViewModel és a Service között egy új réteget kell ezért bevezetnünk, ez lesz a DTO (Data Transfer Object). Ez az alkalmazás legalsó rétege. Ennek az a célja, hogy az adatbázisból jövő dokumentumot a kódunk számára értelmezhető objektummá konvertálja. Nagyon hasznos ennek a rétegnek használata mivel egy esetleges adatbázis szolgáltatás lecserélése után nem sérül majd a ViewModel és a View közötti kommunikációs objektum. Ez annak köszönhető, hogy a két felső réteg független a legalsó rétegtől. Azért használunk különböző rétegeket, mert így bármikor lecserélhető és egy esetleges cserélnél csak a lecserélt réteg kommunikációs objektumait kell módosítani.

7.3 Üzleti logika

Az üzleti logikát a ViewModel osztályokban valósítottam meg. Ez az a része az alkalmazásomnak, amely összekapcsolja a View és a Model osztályokat. Minden View-hez létrehoztam egy ViewModel-t. Ez gondoskodott arról, hogy az adatbázis műveletek megfelelően legyenek meghívva, miközben a hibakezelés is ezen a részen történt. A Model objektum változásait és kezelését a ViewModel végzi. Fontos, hogy az üzleti logika részét a ViewModel-ben kezeljük, vagyis a View már csak egy Model objektumot kapjon, amiből kiolvasva tud majd adatokat megjeleníteni. A ViewModel osztályok a ChangeNotifier-t használják, ami az adatok változását figyeli és értesíti a View osztályt a Provider csomag segítségével.

7.4 Kész alkalmazás

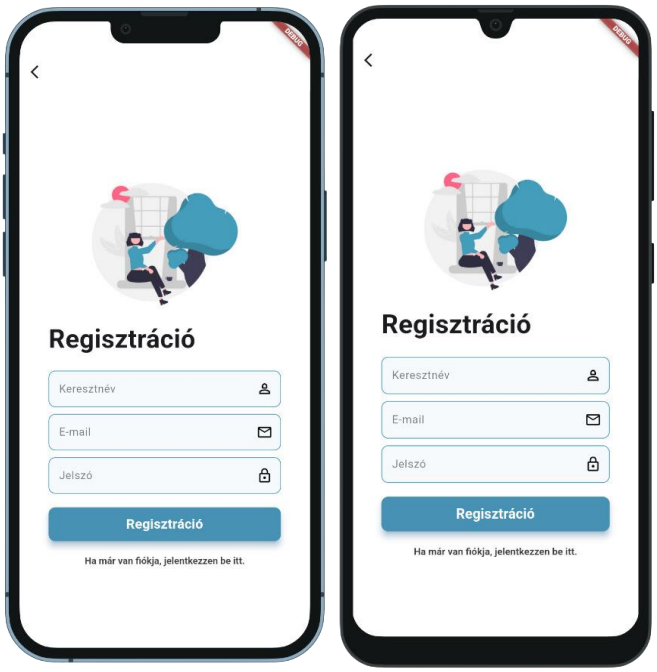
7.4.1 Kezdőképernyő



4. ábra: Kezdőképernyő
iPhone 13 Pro Max és Samsung Galaxy A50

Ez a képernyő jelenik meg, amikor először nyitjuk meg az alkalmazást. Innen tudunk átnavigálni a regisztrációs oldalra. Figyelnem kellett a kép reszponzivitására, ezért a FractionallySizedBox widgetet kellett használnom a helyes megjelenítésért. Különösen azért fontos ennek a használata, mert kisebb képernyőkön nem jelent meg megfelelően a kép. A megjelenítéshez még a Positioned.fill widgetet használtam, a fit attribútumát BoxFit.cover-re állítottam, hogy beterítse a rendelkezésre álló teret. A képnek az elérési útvonalát egy külön fájlba mentettem el, hogy a későbbiekben könnyen újrafelhasználható és könnyen módosítható legyen. Az oldalon látható még egy ElevatedButton, ami a navigációért felelős. Erre a gombra kattintva meghívásra kerül a ViewModel goToNextScreen metódusa, ami a Kezdőképernyő és a Regisztrációs oldal közötti navigációt kezeli. Ezt a gombot kiszerveztem egy külön fájlba, hogy a későbbiekben több helyen is tudjam majd használni, ezzel könnyen újrafelhasználható és egységes lesz az applikáció kinézete. A navigációhoz a Navigator.of(context).pushNamedAndRemoveUntil metódust használtam. Ennek segítségével képernyő váltáskor mindig törlődik a képernyők stack tartalma, így az oda-vissza navigációkor nem tud megtelni a képernyő stack tartalma, ami az alkalmazásunk lassulásához és esetleges hibájához vezethet.

7.4.2 Regisztrációs képernyő



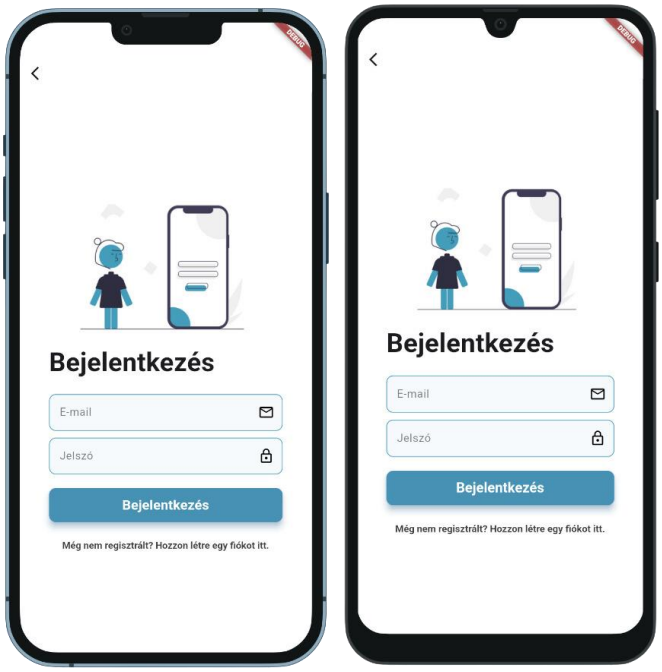
5. ábra: Regisztrációs képernyő
iPhone 13 Pro Max és Samsung Galaxy A50

A Kezdőképernyőről átnavigálva jutunk el erre az oldalra, ahol a felhasználónk regisztrációját tudjuk elvégezni. Itt háromféle felhasználói adatot kell megadnunk. Először is a keresztnevünket, amelyet a sikeres regisztráció után majd a Főoldalon láthatunk viszont. Ezután az e-mail és a jelszó megadása következik. Ez a két adat kell ahhoz, hogy sikeresen tudjunk regisztrálni a Firebase rendszerébe. A regisztrációt az előbb felsorolt mezők alatt található gomb megnyomásával tudjuk végrehajtani. Sikeres regisztráció után átnavigálunk az applikáció Főoldalára. Sikertelen regisztráció után különböző tájékoztató üzeneteket láthatunk a képernyőn. Mindegyik mezőhöz megtalálható egy validációs metódus a ViewModel-ben, amelyek azért felelnek, hogy a felhasználók által megadott adatokat ellenőrizzék és egy esetlegesen rosszul kitöltött mező után megfelelően tájékoztatva legyenek. Ezek a metódusok még a submitSignUp metódus előtt kerülnek meghívásra, így nem futunk bele abba a hibába, hogy a felhasználónk rosszul megadott adatokkal legyen létrehozva. Ha mindent helyesen adtunk meg, akkor a gomb megnyomására meghívásra kerül a ViewModel-ben található submitSignUp metódus. Ebben a függvényben először létrehozunk a UserModel-be mentett adatok alapján egy UserDTO-t, ami azért felel, hogy a UserModel adattagjainak az értékét az adatbázis számára értelmezhető dokumentumként tárolja. Ennek a konverziónak az elvégzése után kerül meghívásra a Service osztály, createUser metódusa, aminek egy argumentuma lesz,

az előbb létrehozott UserDTO. A Service osztálynak nagyon fontos szerepe van, mivel ez a réteg kommunikál az adatbázissal, itt végzi el a tényleges felhasználó létrehozást a .createUserWithEmailAndPassword függvény meghívásával. Ez a metódus a FirebaseAuth csomaghoz tartozik. Több hiba is lehetséges ennek a függvénynek a meghívása után, mint például; egy már használatban lévő e-mail, vagy túl gyenge jelszó megadása miatt. Mivel a FirebaseAuth-nak csak e-mailre és jelszóra van szüksége, hogy sikeres legyen a regisztráció, ezért FireStore-ban a users nevű táblában létrehozunk egy új dokumentumot, ami a felhasználó keresztnevét és e-mail-címét tárolja el. Ezt a két metódust egy try-catch blokkba tettem, hogy az esetlegesen felmerülő hibák megfelelően legyenek kezelve. Hiba esetén hamissal, sikeres regisztráció és dokumentum hozzáadása után igazzal tér vissza a függvény. A hibakezelés a ViewModel-ben található, ahol egy SnackBar üzenet segítségével lesz tájékoztatva a felhasználó a sikertelen regisztrációról. A regisztráció gombra kattintva a képernyőn megjelenik egy CircularProgressIndicator, ami egy töltést szimuláló widget Flutter-ben. Ez a töltés ikon még a FirebaseAuth metódus lefutása előtt indul el és akkor ér véget, amikor a ViewModel submitSignUp metódusában hamisra állítjuk az isLoading adattagot. Sikeres bejelentkezés után a töltés ikon helyett a Főoldalt fogjuk látni a képernyőn.

Ha már sikeresen regisztráltunk korábban, akkor erről a képernyőről átnavigálhatunk a bejelentkezés képernyőre. Ezt úgy tehetjük meg, hogy a Regisztráció gomb alatt található szövegre kattintunk, ami elvégzi nekünk a képernyőváltás.

7.4.3 Bejelentkezés képernyő



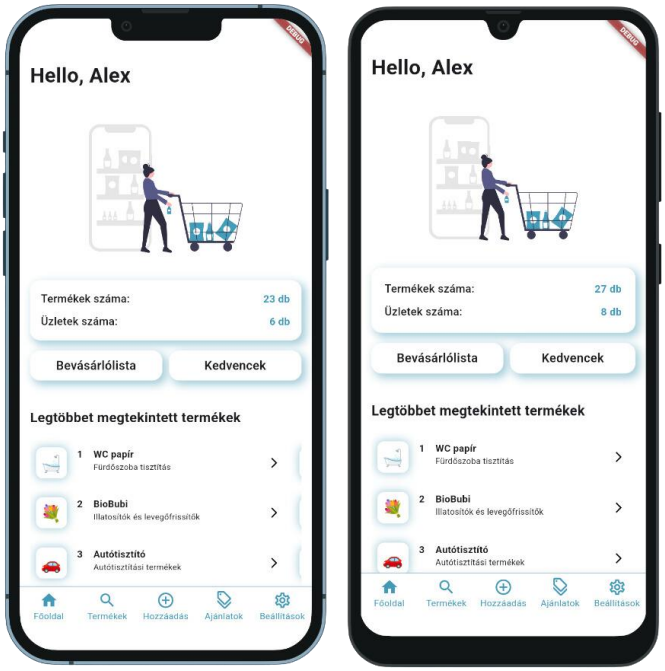
6. ábra: Bejelentkezés képernyő
iPhone 13 Pro Max és Samsung Galaxy A50

Korábban elvégzett regisztráció esetén ezen az oldalon tudunk bejelentkezni. Itt két adatot kell megadnunk ahhoz, hogy sikeres legyen a bejelentkezés. Először a regisztráció során megadott e-mail-címünket, majd a hozzátartozó jelszót kell megadnunk. Miután sikeresen megadtuk a bejelentkezéshez szükséges adatainkat, akkor a TextFormFieldek alatt található gomb megnyomásával teljesíthetjük a bejelentkezési folyamatot. Sikertelen bejelentkezés miatt többféle üzenetet is láthatunk a képernyőn. Minden mezőhöz tartozik egy validációs metódus, ami abban segít, hogy a felhasználó helyes adatokat tudjon megadni. Ezek a metódusok a gomb megnyomása után, de még a submitLogin metódus végrehajtása előtt lesznek meghívva. A ViewModel-ben létrehozunk egy UserModel-t a felhasználó adatainak tárolására. Ez azért hasznos, mert így ezeket az adatokat egy egységként tudjuk kezelni. A View-ben nincs tárolva semmilyen felhasználóhoz köthető adat, itt csak a megjelenítés van megvalósítva. Ennek a logikája a TextFormField onSaved részében van megvalósítva. Sikeres validáció után kerülnek mentésre a UserModel-be a helyesen beírt adatok. A ViewModel submitLogin metódusában van megvalósítva a bejelentkezési logika. Először itt is átkonvertáljuk a UserModellünket egy UserDTO objektummá. A Service osztály, loginUser metódusának ezt a UserDTO-t fogjuk átadni argumentumként. Ebben az esetben is a Service osztály kommunikál az adatbázissal. A bejelentkeztetést a FirebaseAuth csomaghoz tartozó signInWithEmailAndPassword metódus

végzi el. Ennek a metódusnak át kell adnunk a paraméterben kapott e-mail és jelszót. Ezt a metódust is egy try-catch blokkba helyeztem, hogy az alkalmazás és a Firebase közötti hibák megfelelően legyenek kezelve. Hiba esetén hamissal, sikeres bejelentkezés esetén igazzal tér vissza a loginUser függvény. A hibakezelés ebben az esetben is a ViewModel feladata, ahol egy SnackBar üzenet segítségével lesz tájékoztatva a felhasználó a bejelentkezés sikertelenségéről. A bejelentkezés gombra kattintva a képernyőn megjelenik egy CircularProgressIndicator, ami egy töltést szimuláló widget Flutter-ben. Ez a töltés ikon még a FirebaseAuth metódus lefutása előtt indul el és akkor ér véget, amikor a ViewModel submitLogin metódusában hamisra állítjuk az isLoading adattagot. Sikeres bejelentkezés után a töltés ikon helyett a Főoldalt fogjuk látni a képernyőn.

Ha még nincs felhasználónk, vagy szeretnénk egy újat létrehozni, akkor erről a képernyőről átnavigálhatunk a regisztrációs oldalra, ahol elvégezhetjük a felhasználónk létrehozását. Ezt a navigációt úgy végezhetjük el, hogy a Bejelentkezés gomb alatt található szövegre kattintunk, amivel átválthatunk a regisztrációs képernyőre.

7.4.4 Főoldal képernyő



7. ábra: Főoldal képernyő
iPhone 13 Pro Max és Samsung Galaxy A50

Sikeres bejelentkezés, vagy regisztráció után ez a képernyő fogadja a felhasználót. Itt egyszerre több minden látható, ami hasznos lehet az alkalmazást használó számára. A képernyő tetején egy köszöntő szöveg látható, ami mindig az adott felhasználó keresztnevét jeleníti meg a „Hello, ” szöveg után.

Ezalatt egy kép található, amiben az alkalmazás színei találhatóak meg. Ez a kép kellemes megjelenést ad ennek az oldalnak és témájában is passzol az alkalmazáshoz. A képet egy FractionallySizedBox widgetbe helyeztem, hogy állandó méret helyett az adott képernyő méretéhez igazodjon.

A kép alatt két információt is láthatunk az alkalmazásban szereplő adatokról. Először az alkalmazásban szereplő termékeknek darabszámát láthatjuk. Ez az összes, az adatbázishoz hozzáadott terméknek a számát jelenti. Alatta pedig az adatbázishoz hozzáadott üzleteknek az összegét láthatjuk. Ezeket az információkat Firebase lekérdezésekkel szerezhetjük meg az adatbázisból. A ViewModel-ben szerepel két metódus is ezeknek az adatoknak a kezeléséhez. A termékek darabszámának az összegét a getProductCount metódusban, a Service osztály getDocumentCount metódusának hívásával tudjuk megszerezni. Ez a függvény egy paramétert vár, ami a termékeket tartalmazó tábla neve FireStore-ban. Ebben a függvényben van végrehajtva a get metódus, amivel a termékek táblából le tudjuk kérdezni az összes terméket. Ezeket FireStore dokumentumok formájában adja vissza. Ennek a lekérdezésnek az eredményét egy querySnapshot változóba menti el, aminek a dokumentumainak számát a length tulajdonságával érhetjük el. Ez az érték lesz a getDocumentCount metódusnak a visszatérési értéke. Miután sikeresen megszerezte ezt az értéket a getProductsCount, azután megjeleníti majd a View-ben. Az üzletek darabszámának az összeget hasonlóan a getStoresCount metódusban a Service osztály getDocumentCount metódusának hívásával tudjuk megszerezni. Ez a függvény egy paramétert vár, ami az üzleteket tartalmazó tábla neve FireStore-ban. Ebben a függvényben van végrehajtva a get metódus, amivel az üzletek táblából le tudjuk kérdezni az összes terméket. Ezeket FireStore dokumentumok formájában adja vissza. Ennek a lekérdezésnek az eredményét egy querySnapshot változóba menti el, aminek a dokumentumainak számát a length tulajdonságával érhetjük el. Ez az érték lesz a getDocumentCount metódusnak a visszatérési értéke. Miután sikeresen megszerezte ezt az értéke a getStoresCount, azután megjeleníti majd a View-ben.

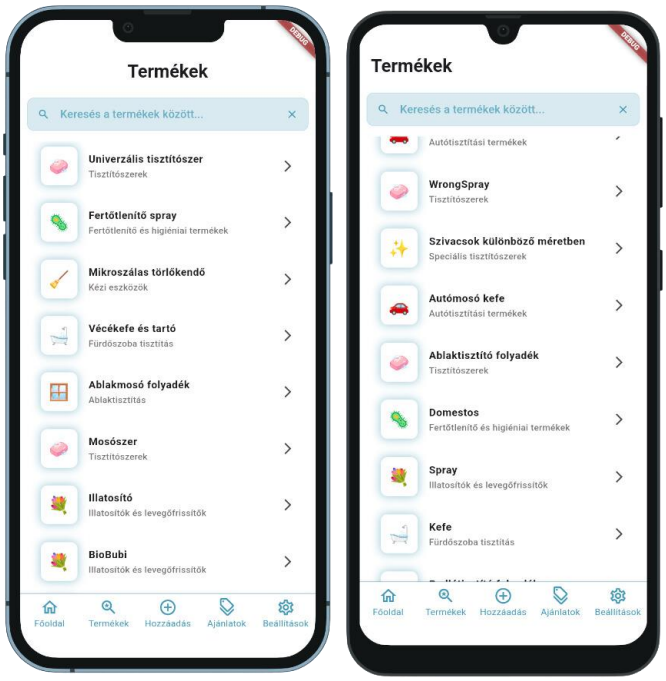
Ez a Card alatt láthatunk két gombot, amelyek megnyomásával a Bevásárlólista és a Kedvencek képernyőre navigálhatunk. Ezeket a gombokat is egy Cardban valósítottam meg,

amihez az alkalmazás színével megegyező árnyékot adtam hozzá. Ezzel is növelve az applikáció esztétikus megjelenését.

A gombok alatt egy listát láthatunk, ahol a 9 legtöbbet megtekintett termék jelenik meg. A felhasználó ebből a listából könnyedén navigálhat az adott termék oldalára. Minden termék látogatása során a termékhez tartozó ViewCount mező lesz megnövelve. Először 3 darab terméket láthatunk, majd horizontálisan görgetve tekinthetjük meg a továbbiakat.

Ezen az oldalon láthatjuk először az alsó navigációs menüt, aminek a segítségével elérhetjük az alkalmazás további képernyőit.

7.4.5 Termékek képernyő



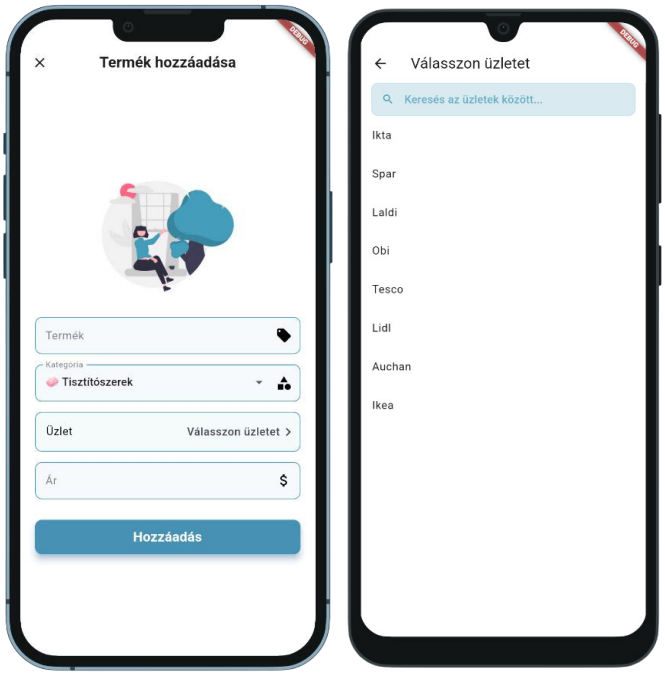
8. ábra: Termékek képernyő
iPhone 13 Pro Max és Samsung Galaxy A50

A fenti képernyőképeken a Termékek oldal látható. Itt találhatják meg az összes adatbázishoz hozzáadott terméket. A keresőmező segítségével külön termékekre szűrhetünk rá, ami által könnyebben megtaláljuk az általunk keresett termékeket. Alapértelmezetten az összes elérhető termék betöltődik a listába, keresésnél viszont csak a keresett termékek jelennek meg. Nem számít a kis- és nagybetű különbség mivel az adatbázisban a termékek neve is eltárolódik kisbetűs változatban is. Ennek a segítségével tudtam megvalósítani azt, hogy ne jelentsen gondot a kis- és nagybetűk közötti különbség.

Ezen a listán a termékek nevét, képét és kategóriáját láthatjuk. A kép, ami ilyenkor megjelenik a termék hozzáadásánál jön létre, amikor kiválasztjuk a kategóriáját. Emellett látható egy apróbb nyíl is, ami azt a célt szolgálja, hogy felhívja a felhasználó figyelmét arra, hogy erről az oldalról a felhasználó által választott termék oldalára lehet navigálni.

Azért tartottam fontosnak ennek az oldalnak a megvalósítását, mert így könnyebben elérhetővé válnak a termékek és a keresés segítségével nem kell végiggörgetni az egész lista tartalmán. Ezért ez egy hasznos funkció, amikor a felhasználó gyorsan szeretne tudomást szerezni a termékek áráról, vagy éppen a bevásárlólistájához szeretné bővíteni.

7.4.6 Termék hozzáadása képernyő



9. ábra: Termék hozzáadása képernyő
iPhone 13 Pro Max és Samsung Galaxy A50

Az alsó navigációs menüben a Hozzáadásra kattintva megnyílik egy Termék hozzáadása oldal. Itt tudunk új termékeket hozzáadni az adatbázishoz. A képernyő felső részén egy színben hozzáillő kép látható. Alatta 4 mezőt található, amelyek közül mindegyiket ki kell tölteni ahhoz, hogy sikeres legyen a hozzáadás.

Először a termék nevét kell megadni, aminek az értéke nem lehet null. Utána a termék kategóriáját választhatjuk ki egy legördülő listából. Itt egy kategória nevét és a hozzátartozó emojit láthatjuk. Később ez az emoji fog majd megjelenni a termék képeként. Azért találtam ki ezt a módszert, mert így nem kell a felhasználónak a képfeltöltéssel bajlódnia, viszont egy emoji megjelenítésével mégis színesebb és esztétikusabb lesz az alkalmazás. Ráadásul ezzel a megoldással jelentősen spórolhatunk az adatbázisunk tárhelyével is.

Ezután az üzletet kell kiválasztanunk. Itt először is rá kell kattintanunk a „Válasszon üzletet” mezőre. Itt átnavigál minket egy másik oldalra, ahol először az összes üzlet megjelenik a listában. Innen azt az üzletet válasszuk ki amelyikhez hozzá szeretnénk adni a termékünket. Itt is megtalálható egy kereső mező, aminek a segítségével különböző üzletek nevére szűrhetünk rá, ezzel is megkönnyítve a keresést. Ha egy olyan áruházat szeretnénk kiválasztani, ami még nem létezik az adatbázisban, akkor csak be kell gépelnünk a teljes nevét majd

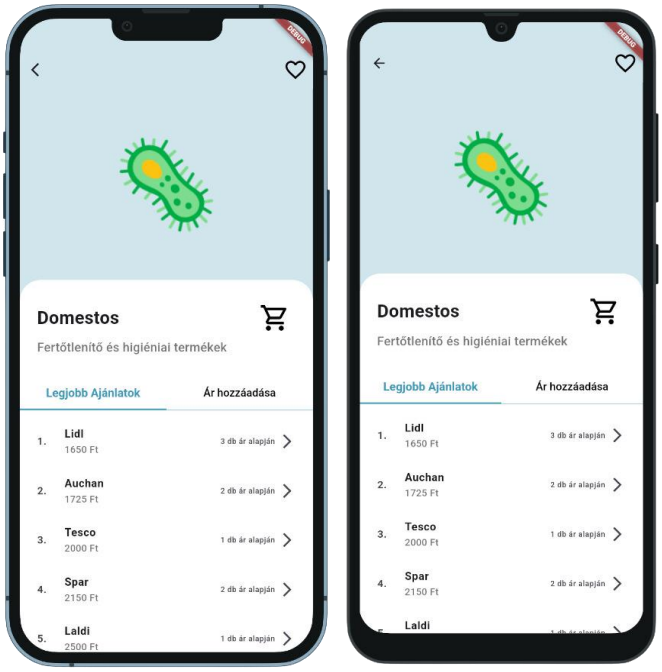
kiválasztva a listából ugyanúgy hozzá tudjuk adni. A kis- és nagybetűkre nem kell figyelnünk, mert automatikusan nagy kezdőbetűvel kerül feltöltésre az adatbázisba. Ezután a kiválasztás után még nem lesz egyből elérhető az új üzlet. Sikeres termék hozzáadásakor lesz majd látható és a következő hozzáadásnál már kiválasztható lesz a listából. Miután kiválasztottuk az üzletet, akkor a „Válasszon üzletet” szöveg helyett az általunk választott üzlet lesz látható.

Az üzlet kiválasztása után jöhet az utolsó hiányzó adat megadása. Ebben a mezőben a terméknek az árát kell megadni. A felhasználó csak pozitív, egész számokat tud megadni. Ha ezektől eltérő értéket ad meg, akkor a ViewModelben megvalósított validációs függvény jelez a hozzáadás előtt. A mező megnyitásakor alapértelmezetten egy ilyen billentyűzet nyílik meg, ahol csak a számokat láthatjuk. A számok mellett még megadhatunk pontot vagy vesszőt is, éppen ezért fontos a validációs metódusnak a megléte.

Ha mindent helyesen töltöttünk ki, akkor a Hozzáadás gombra kattintva hozzáadhatjuk a terméket az adatbázishoz. Hiba esetén a felhasználó tájékoztatást kap a hozzáadás sikertelenségéről. Abban az esetben, ha már egy adatbázisban létező terméket szeretnénk hozzáadni, akkor nem adódik hozzá még egyszer, hanem csak a termékhez lesz egy új ár feltöltve. Ez pedig az alapján van ellenőrizve, hogy a termék neve létezik-e már az adatbázisban.

Az üzleteknek csak a nevük van eltárolva az adatbázisban, kis- és nagybetűs formában, hogy könnyebben kereshetőek legyenek. A továbbiakban az üzleteket hozzáadhatjuk a kedvenceink közé és a bevásárlólistában is elő kerül majd még a nevük.

7.4.7 Termék képernyő



10. ábra: Termék képernyő
iPhone 13 Pro Max és Samsung Galaxy A50

A Termék képernyőn több információt is láthatunk a kiválasztott termékről. Ezt az oldalt több helyről is el tudjuk érni. Többek között a Termékek, Ajánlat, Főoldal és a Kedvencek képernyőről is. Ahányszor rámegyünk egy terméknek az oldalára, akkor FireStoreban az adott termék dokumentumában lévő ViewCount mező megnövekszik eggyel. Ennek a segítségével lehet kilistázni a 9 legtöbbet látogatott terméket a Főoldalon. Hasonló logikát alkalmaztam a legtöbbet megtekintett ajánlatoknál is. Ott is minden esetben, amikor egy felhasználó meglátogatja egy ajánlat oldalát, akkor ez a számláló eggyel növekszik.

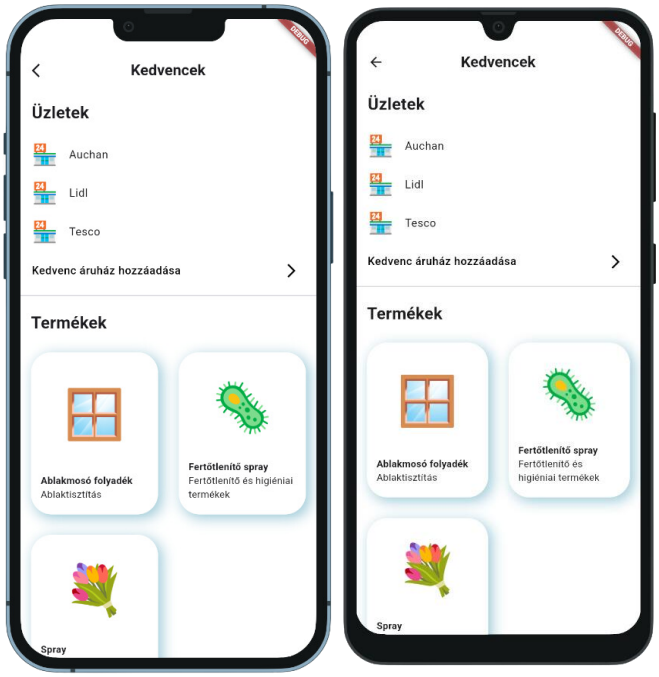
A Termék oldalon láthatjuk még a szív alakú ikont, ami segítségével a kedvencek közé tudjuk tenni a terméket. Első megnyomásra a kedvencek közé tudjuk tenni. Sikeres műveletet az ikon változásán vehetjük észre. Következő megnyomásra viszont eltávolítjuk a kedvencek közül.

Erről az oldalról tehetjük be a terméket a Bevásárlólista termékei közé. Az applikációnak a Bevásárlólista az egyik fő funkciója, ezért nagyon fontos, hogy a termékeket hozzáadhassuk ehhez a listához. A Termék oldalon csak hozzáadni tudjuk termékünket, az eltávolítást a Bevásárlólista oldalon végezhetjük el.

Egy másik lényeges funkció ezen az oldalon látható, mégpedig a Legjobb Ajánlatok rész. Ezen a felületen láthatjuk a termékhez hozzáadott árakat és a hozzátartozó üzleteket. Itt az árakat növekvő sorrendben láthatjuk, vagyis a legjobb ajánlat található meg legfelül. Az itt látható ár az áruházhoz tartozó összes hozzáadott árnak a mediánja. A listaelem jobb oldalán pedig azt láthatjuk, hogy hány darab ár tartozik az üzlethez. A mellette lévő kis nyílra kattintva tekinthetjük meg az összes elérhető árat, ami hozzá lett adva az adott üzlet, adott termékéhez. Ezen az oldalon az összeg mellett láthatjuk a hozzáadás dátumát is. Ez egy hasznos funkció, mert a felhasználók így láthatják, hogy mennyire friss az áruházhoz tartozó ár.

Az Ár hozzáadása fülön egy új árat adhatunk hozzá egy általunk választott üzlethez. Az üzlet választás ugyanúgy működik, mint a Termék hozzáadásánál. Először át kell navigálnunk az Üzlet választása oldalra, majd az ottani listából választhatunk a már hozzáadott üzletek közül, vagy hozzáadhatunk egy újat is. Az összeg megadásánál az előző oldalhoz hasonlóan figyelnünk kell, hogy pozitív, egész számokat adjunk meg. Hibás érték megadása esetén a validációs függvény jelzi majd a hibát.

7.4.8 Kedvencek képernyő



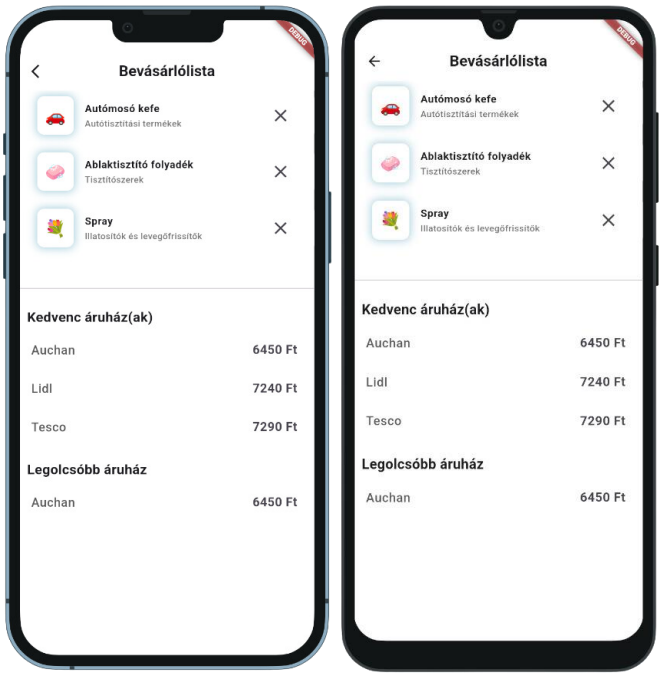
11. ábra: Kedvencek képernyő
iPhone 13 Pro Max és Samsung Galaxy A50

A Kedvencek oldalon a felhasználó által kedvelt üzleteket és termékeket lehet megtekinteni. Azért hasznos ez az oldal, mert így nem kell mindig megkeresni minden egyes terméket külön-külön, hanem a Főoldalról egyből el lehet navigálni a kedvencek oldalra, ahol láthatjuk egy kedvencek közé tett termékeket.

Egy terméket úgy tehetünk a kedvencek közé, hogy elnavigálunk az oldalára, ahol a szív ikonra kattintva tudjuk elmenteni. Sikeres hozzáadás után a szív piros színű lesz. Ha ki akarjuk venni a kedvencekből, akkor a csak annyi a dolgunk, hogy még egyszer rá kell kattintanunk a szív ikonra. Mivel a kedvencekből könnyedén elérjük a termék oldalát, így a bevásárlólistához is gyorsabban hozzá tudjuk adni.

Nem csak termékeket, hanem üzleteket is tehetünk a kedvenceink közé. Ez a funkció azért hasznos, mert a Bevásárlólista képernyőn majd a kedvenc üzleteinkben mutatja majd a végösszeget. Egy üzletet úgy tehetünk a kedvencek közé, hogy rányomunk a „Kedvenc áruház hozzáadása” részre, ami elnavigál minket egy áruház kereső oldalra. Ez az oldal hasonló ahhoz, amit a termékek hozzáadásánál láthattunk, de nem itt nem tudjuk kiválasztani őket. A Kedvenc áruház hozzáadása oldalra lépve először az összes áruház megjelenik a listában. A lista elején azokat az áruházakat láthatjuk, amelyek már a kedvenceink között szerepelnek. A kilistázott üzleteket neve mellett egy szív alakú ikon látható, amire egyszer rányomva a kedvencekbe tudjuk helyezni az áruházat. Sikeresen hozzáadás után az üres szív ikon átvált teli pirosra. Egyből a kedvencek között találhatjuk az áruházat miután visszanavigáltunk az előző oldalra.

7.4.9 Bevásárlólista képernyő



12. ábra: Bevásárlólista képernyő
iPhone 13 Pro Max és Samsung Galaxy A50

Az elkészített alkalmazásom egyik leglényegesebb funkciója a Bevásárlólista képernyőn található. Az előzőekben ismertettem, hogy hogyan lehet hozzáadni termékeket a Bevásárlólistához. Ezt a képernyőt a Főoldalról tudjuk elérni a Bevásárlólista gombra kattintva. A képernyő felső részén egy listát láthatunk az eddig hozzáadott termékekről. Itt tudjuk eltávolítani a Bevásárlólistáról azokat a termékeket, amelyeket már nem szeretnénk, hogy rajta legyenek a listán. A bevásárlólistánk tartalma alatt különböző üzleteket láthatunk. Először a Kedvenc áruházainkat, majd a Legolcsóbb áruházat. A Kedvenc áruházak részen azokat az üzleteket láthatjuk, amelyeket korábban hozzáadtunk a kedvenceink közé. Ebben a listában csak azok az üzletek és a termékek végösszege jelenik meg, amelyekben megtalálható az összes Bevásárlólistához hozzáadott termék. Ha van olyan üzlet, ami a kedvencek között van, de ebben az üzletben az egyik termék nem elérhető, akkor nem kerül bele a listába. Azért tartom fontosnak ennek a funkciónak a meglétét, mert mindenkinek vannak kedvenc üzletei, ahova szeretnek járni. Éppen ezért lehet hasznos, ha láthatjuk, hogy a kedvenc üzleteink közül melyikkel járunk a legjobban. A Legolcsóbb áruház pedig, azaz áruház, ahol az összes elérhető üzlet közül a legjobb áron tudjuk megvásárolni a bevásárlólistánk tartalmát. Itt is megjelenik az üzlet neve és a végösszeg.

7.4.10 Ajánlatok és ajánlat képernyők

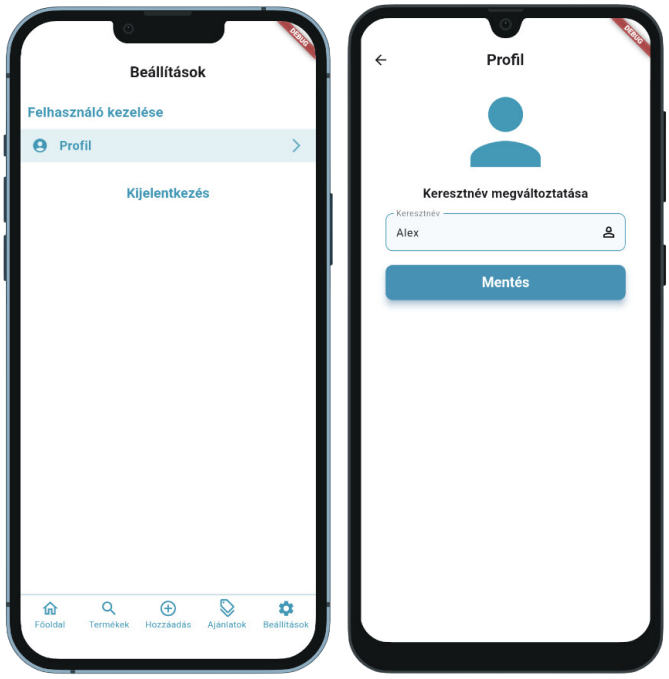
13. ábra: Ajánlatok és ajánlat képernyők
iPhone 13 Pro Max és Samsung Galaxy A50

Ez a két képernyő az ajánlatokkal kapcsolatos. Az Ajánlatok képernyőn láthatjuk a felhasználók számára elérhető ajánlatokat. Az első sávban véletlenszerűen láthatjuk az ajánlatokat. A második sávban a legtöbbet megtekintett ajánlatokat láthatjuk, hasonlóan megvalósítva, mint a Főoldalon. Minden ajánlat megtekintés után eggyel növekszik a ViewCount mezőjének értéke. Ennek a horizontális listának az elején találhatóak a legtöbbet megtekintett, majd csökkenő sorrendben következnek a további ajánlatok.

Ezekből a listákból kiválasztva egy ajánlatot jutunk el az ajánlat képernyőjéhez. Ezen az oldalon láthatjuk az ajánlathoz tartozó emojikat, a nevét és hogy melyik áruházban érhető el. Az előbbi információk alatt láthatunk egy navigációs menüt. Az első oldalán egy rövid leírást láthatunk az ajánlatról. A második, Termékek oldalon pedig az ajánlathoz hozzárendelt termékeket érhetjük el. Ezen az oldalon a termékre kattintva a termék oldalára jutunk el, ahol megtekinthetjük az árait más üzletekben is.

Ajánlatokat a felhasználó nem tud hozzáadni, azt csak a Firebase felületén lehetséges és onnan kerülnek kilistázásra az elérhető ajánlatok. Későbbiekben lehetne készíteni egy külön felületet, hogy az ajánlatok feltöltése könnyebb legyen az üzletek számára.

7.4.11 Beállítások és Profil képernyők



14. ábra: Beállítások és profil képernyők
iPhone 13 Pro Max és Samsung Galaxy A50

A Beállítások képernyőről érhetjük el a Profil oldalt, ahol a felhasználónknak a keresztnevét tudjuk megváltoztatni. Amikor erre az oldalra lépünk, akkor a Keresztnév mezőbe betöltésre kerül a jelenlegi keresztnevünk. Ezt megváltoztatva egyből láthatjuk a változást a mezőben. A Főoldalra navigálva már az új keresztnevünket láthatjuk. A Beállítások képernyőn még a felhasználónkból tudunk kijelentkezni. Sikeres kijelentkezés után az alkalmazás Kezdőképernyőjére leszünk navigálva.

Irodalomjegyzék

1. Amazon Web Service. (2023). https://aws.amazon.com/what-is/flutter/ [Utoljára megtekintve: 2023.12.09.]

2. Kotlin (2023). The Six Most Popular Cross-Platform App Development Frameworks. https://kotlinlang.org/docs/cross-platform-frameworks.html#8-educational-materials [Utoljára megtekintve: 2023.12.09.]

3. Akintunde, C. (2020). Basic Introduction to User Experience and User Interface Design. UX Design Bootcamp. https://bootcamp.uxdesign.cc/basic-introduction-to-user-experience-and-user-interface-design-f0aae08a2b44 [Utoljára megtekintve: 2023.12.10.]

4. Coursera Inc. (2023). https://www.coursera.org/articles/ui-vs-ux-design [Utoljára megtekintve: 2023.12.11.]

5. Chmielewska, P. (2020). https://itcraftapps.com/blog/moodboard-mobile-app-design-inspiration/#whatisamoodboard [Utoljára megtekintve: 2023.12.12.]

6. proto.io. (2023). https://proto.io/ [Utoljára megtekintve: 2023.12.12.]

7. Trello, Atlassian. (2023). https://trello.com/home [Utoljára megtekintve: 2023.12.09.]

8. Coursera Inc. (2023). https://www.coursera.org/articles/what-is-git [Utoljára megtekintve: 2023.12.10.]

9. Mohite, J. (2020). Flutter: MVVM Architecture. https://medium.com/flutterworld/flutter-mvvm-architecture-f8bed2521958 [Utoljára megtekintve: 2023.12.12.]

10. Canva. (2023). https://www.canva.com/ [Utoljára megtekintve: 2023.12.11.]

11. Dart. (2023). https://dart.dev/overview [Utoljára megtekintve: 2023.12.12.]

12. Firebase. (2023). https://firebase.google.com/ [Utoljára megtekintve: 2023.12.11.]

13. Flutter. (2023). https://flutter.dev/ [Utoljára megtekintve: 2023.12.13.]

14. GitHub, Inc. (2023). https://github.com/ [Utoljára megtekintve: 2023.12.10.]

15. Limpitsouni, K. (2023). https://undraw.co/ [Utoljára megtekintve: 2023.12.09.]

16. Rajas (2020). Flutter styling tricks for beginners. https://medium.com/@srajas02/flutter-styling-tricks-3192907f56e3 [Utoljára megtekintve: 2023.12.10.]

17. Udemy, Inc. (2023). https://www.udemy.com/ [Utoljára megtekintve: 2023.12.12.]

18. Vijayan, V. (2022). Flutter Best Practices – Part 1. https://itnext.io/flutter-best-practices-part-1-e89467ea4823 [Utoljára megtekintve: 2023.12.13.]

Ábrajegyzék

Köszönetnyilvánítás

Tisztelettel és hálával szeretném kifejezni köszönetemet Kiss-Vetráb Mercedesnek, aki, mint témavezetőm, rendkívüli szakértelemmel és elkötelezettséggel segített a szakdolgozatom elkészítésében. Szakmai útmutatása és értékes tanácsai nélkülözhetetlenek voltak az alkalmazás fejlesztése során, és mélyen hálás vagyok a támogatásáért és bátorításáért.

Köszönetemet fejezem ki családomnak is, akik mindig mellettem álltak. Támogatásuk, ösztönzésük és végtelen szeretetük elengedhetetlen volt a nehéz időkben és hozzájárultak sikeremhez.

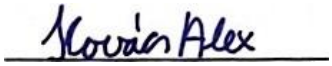
Nyilatkozat

Alulírott Kovács Alex 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.

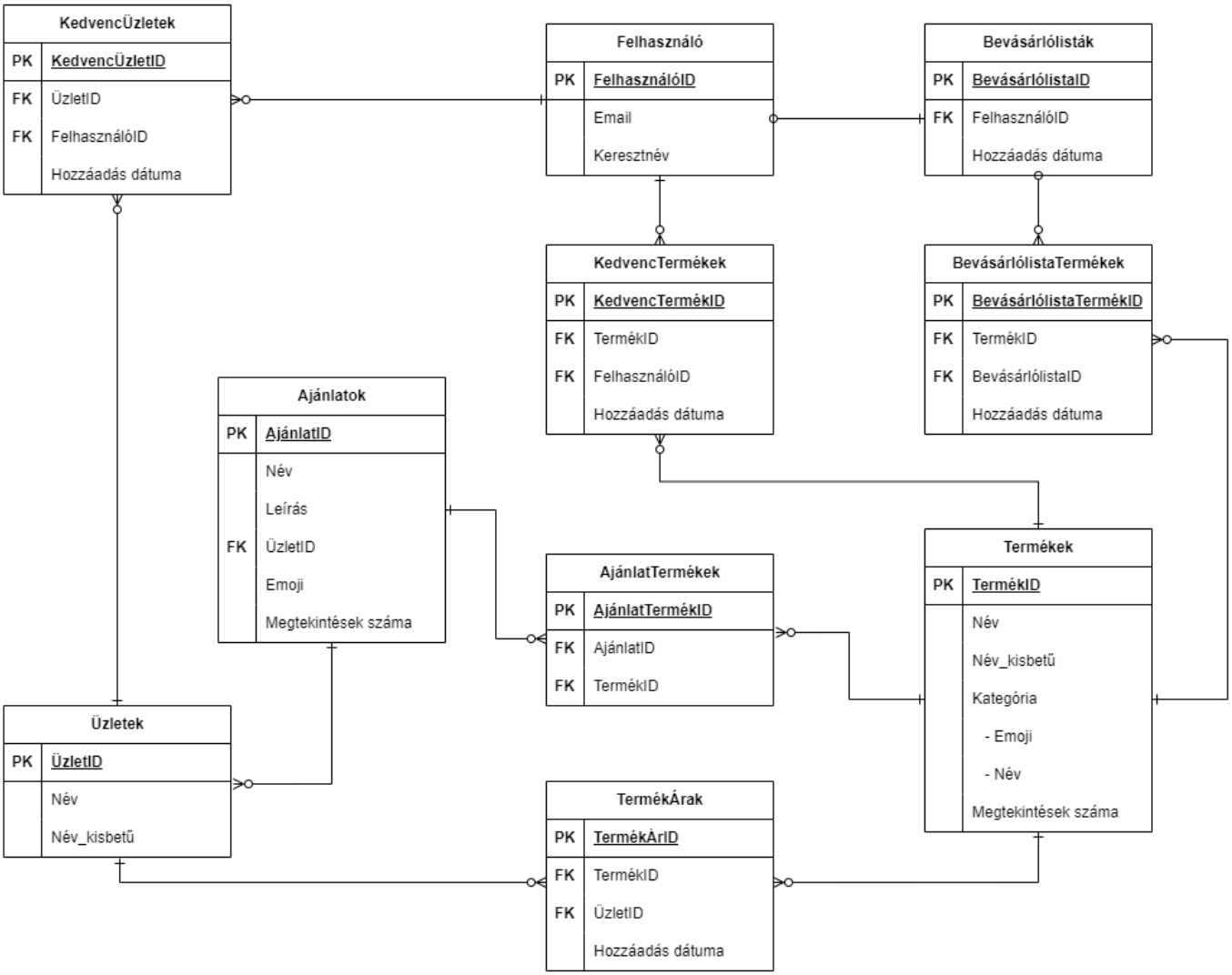
Tudomásul veszem, hogy szakdolgozatomat a Szegedi Tudományegyetem Diplomamunka Repozitóriumában tárolja.

Dátum (Szeged, 2023.12.15.)



aláírás

Mellékletek



1. melléklet: EK-diagram