ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK	Hallgató neve:	**Szalai András**
2023	Hallgató törzskönyvi száma:	**T/005821/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Szalai András**
Törzskönyvi száma: T/005821/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Android alkalmazás diabéteszeseknek
Android application for diabetics

Intézményi konzulens: Simon-Nagy Gabriella
Külső konzulens:

Beadási határidő: 2022. május 15.

A záróvizsga tárgyai: Számítógép architektúrák
Szoftvertervezés és –fejlesztés specializáció

A feladat

Tervezzen és valósítson meg egy vércukorszint menedzselő mobilalkalmazást diabétesszel élő felhasználók számára. Kutassa fel és vizsgálja meg a feladat megoldásához használható technológiákat és adatforrásokat. Ismertesse a hasonló célú rendszerek működését és képességeit. Valósítson meg egy olyan tesztalkalmazást, amely támogatja a felhasználót a vércukorszint menedzselésében, és képes a felhasználó szokásai alapján személyre szabott tanácsokat megjeleníteni.

A dolgozatnak tartalmaznia kell:

- a feladat pontos ismertetését,
- a hasonló rendszerek működését és képességeit,
- a feladat megoldásához alkalmazható technológiák tömör bemutatását,
- a választott megoldások indoklását,
- a tervezés és megvalósítás részletes leírását,
- a tesztelés tervét és eredményeit,
- az elkészített rendszer eredményeinek részletes bemutatását és értékelését,
- a továbbfejlesztési lehetőségek számba vételét.

ÓBUDAI EGYETEM * Neumann János Informatikai Kar * 1. Budapest

Ph.

.....
Dr. Vámossy Zoltán
intézetigazgató

A szakdolgozat elévülésének határideje: **2024. május 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat / diplomamunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban / diplomamunkámban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2022.12.12.

.....
hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Szalai András

Neptun Kód: [redacted]

Tagozat: Nappali

Telefon: [redacted]

Levelezési cím (pl.: lakcím): [redacted]

Szakdolgozat címe magyarul:
Android alkalmazás diabéteszeseknek

Szakdolgozat címe angolul:
Android application for diabetics

Intézményi konzulens:
SIMON-NAGY GABRIELLA

Külső konzulens:

Alk.	Dátum	Tartalom	Aláírás
1.	2021. 09. 21.	A téma meghatározása	
2.	2021. 10. 19.	A kutatási területek áttekintése	
3.	2021. 11. 22.	Az irodalomkutatás összegzése	
4.	2021. 12. 10.	A rendszerspecifikáció véglegesítése	

A hallgató a Szakdolgozat I. tantárgy követelményét teljesítette, beszámolóra bocsátható.

.....

Intézményi konzulens

Budapest, 2021. december 10.

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Szalai András

Neptun Kód: [redacted]

Tagozat: Nappali

Telefon: [redacted]

Levelezési cím (pl.: lakcím): [redacted]

Szakdolgozat címe magyarul:

Android alkalmazás diabéteszeseknek

Szakdolgozat címe angolul:

Android application for diabetics

Intézményi konzulens:

DR. SIMON-NAGY GABRIELLA

Külső konzulens:

Alk.	Dátum	Tartalom	Aláírás
1.	2022. 09. 20.	Megvalósítási lehetőségek áttekintése	
2.	2022. 10. 19.	Fejlesztési fázis ellenőrzése	
3.	2022. 11. 24.	Tesztelés ellenőrzése	
4.	2022. 12. 09.	Eredmények, összegzés	

A hallgató a Szakdolgozat II. tantárgy követelményét teljesítette, védésre bocsátható.

.....

Intézményi konzulens

Budapest, 2022. december 9.

Tartalom

Ábrajegyzék

1. BEVEZETÉS

Az utóbbi időkben jelentősen fejlődött a mobilalkalmazások gyártása. Új szereplők léptek be a piacra, aminek hatására olyan alkalmazások jelentek meg, melyek a népszerűség mellett a minőséget is egyre inkább szem előtt tartják. Elengedhetetlenné vált olyan termékek előállítása, melyek szorosan egymás között versengenek a felhasználók kegyeiért. Legyen szó a szórakoztatóipari-, tudományos célt szolgáló szoftverekről, vagy akár ezek ötvözetéről. Ezen dolgozat célja az, hogy a tudományt felhasználva miképpen lehet olyan alkalmazásrendszert készíteni, mely a lehető legprecízebben képes a felhasználó pártfogója lenni egy bizonyos témában, mely a Diabétesz nevet viseli. Ez egy bonyolult, végeláthatatlan terület, aminek kialakulását máig kérdések és homály övezi az orvostudományban. A cél e tudomány összehangolása az informatikával, mely utóbbi révén nagy fejlődésnek örvendhet már a mai orvoslástan is. A szakdolgozat szándékosan nem tér ki a betegség egyes aspektusainak bizonyítására. Azonban részét képezi vizsgálódás, válaszok keresése a betegséggel járó nehézségek miért-és mikéntjére, de legfőképpen azok megoldása. A kutatás eredménye remélhetőleg választ adhat egészségügyi és számítástechnikai kérdésekre – és problémákra –, amelyek e területet határolják.

2. MOTIVÁCIÓ

Ezt a tanulmányt több dolog ihlette. Az informatika egy ideje már olyan szintre fejlődött, hogy a háztartásainkat olyan gépek veszik körül, amik nélkül nehezen tudnánk elképzelni a mindennapjainkat. Mivel már szinte minden tevékenységre találunk telefonos alkalmazásokat, olyan kényelmi szintet teremtettünk magunknak, hogy arról is megfeledkezünk, milyen napi teendőink vannak. Miért is jutna eszünkbe, ha a naptárba bepötyögtük? Emlékeztetőt is állítottunk be… A karóráink ma már pulzust, vérnyomást és alvásciklust mérnek. Ebben a felgyorsult világban egyre inkább elavulttá válnak azok a dolgok, amelyek egy hátrányos dologgal járnak: idő. Miért ne lehetne a zsebünkben egy pöccintésre a kedvenc diabétesz személyi asszisztensünk? Személyes tapasztalatom szerint a vércukorszint nyomon követése – ha az adatok rögzítését és elemzését vesszük figyelembe – elég időigényes feladat. És akkor még nem ugrottunk be a diabetológiára kontrollra vagy beszéltünk a dietetológusunkkal, hogy helyrepofozza az étrendünket.

A 2021-es évben a 20-79 év közötti cukorbetegek száma csaknem fél milliárd – a Nemzetközi Diabétesz Szövetség közlése szerint. A felmérés szerint ez a szám csak emelkedni fog, illetve a betegek aránya is nő a nem diabéteszesekhez képest. Ezen kívül minden kilencedik ember cukorbetegségben hal meg [1]. A technológiai- illetve orvostudományi háttért tekintve, valamint a tény, hogy mennyire szüksége van egy olyan eszközre, amely megkönnyíti a betegségben élők hétköznapjait, vált számomra törekvéssé egy egyszerű, kényelmes és hasznos rendszer megalkotása.

1. ábra: Diabéteszesek a világon

3. IRODALOMKUTATÁS

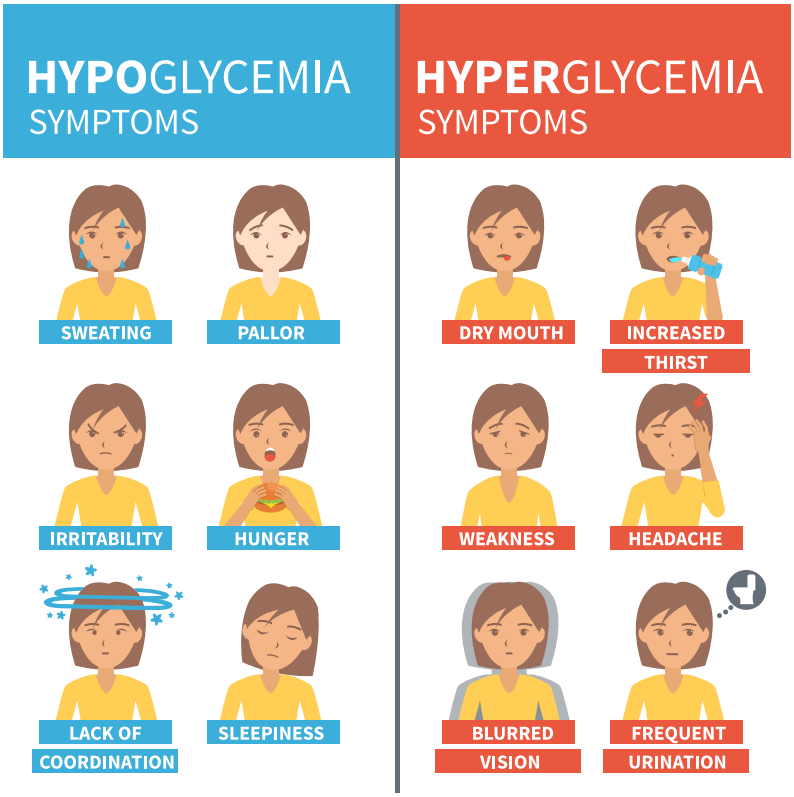
3.1. Megvalósítandó rendszer

Témaválasztásom célja egy olyan alkalmazás megtervezése és elkészítése, mely segíti a cukorbeteg emberek mindennapi életét. Egy olyan applikáció fejlesztése, ami nem csak egyszerűbbé, de tudatosabbá is teszi a diabéteszesek életvitelét, legyen szó sportolásról, diétáról, vagy a gyógyszerek megfelelő adagolásáról. A fejlesztendő szoftver esetében – a könnyed kezelésen túl – fontos szerepet kap a statisztikák felállítása és értékelése, amely segítségével a program a rendelkezésre álló adatokból képes a felhasználót tanácsokkal ellátni, hogy elkerülje a jövőben kialakuló szövődményeket. Ahhoz, hogy megértsük, miben lehet hasznunkra egy ilyen „vércukor monitorozó app”, előbb érdemes tisztában lenni magával a betegséggel.

3.2. Diabétesz

A diabétesz [2,3] egy olyan autoimmun betegség, melyet az inzulintermelés hiányán kívül főként a hiperglikémia, azaz a magas vércukorszint jellemez. Amennyiben ez az állapot tartós, abban az esetben hosszútávú károsodást okozhat a szemben, vesékben, szívben és az érrendszerben. Két típusa létezik: 1-es és 2-es típus. Míg az előbbi esetében az inzulin kiválasztás teljes hiánya lép fel - általában már fiatalabb korban -, addig az utóbbi egy klinikai tünetektől mentes jelenség, többnyire idős embereknél jelentkezik. Noha hajlamosabbak rá a túlsúlyos, illetve a mozgásszegény életmódot folytató emberek, oka máig ismeretlen, ezért megelőzni nem lehet, csak a kezelésére van lehetőség. A vércukorszint karbantartására pedig több eszköz is rendelkezésre áll: megfelelő étrend, mozgás, illetve gyógyszeres kezelés, amennyiben szükséges.

Szerencsére a vércukorszint emelkedésének illetve csökkenésének jelei fizikailag viszonylag hamar kimutathatók, ezért még időben képesek vagyunk a kezelésére. Hipo- és hiperglikémia (alacsony- és magas vércukorszint) esetén egyszerre több tünet jelentkezik a páciensnél. Léteznek olyan „anomáliák”, melyek kevésbé látványosan hívják fel magukra a figyelmet. Ilyen a „Somogyi-effektus” és a „hajnali jelenség”. Hajnali jelenségről akkor beszélünk, ha az éhomi vércukorszint reggel emelkedik, méghozzá megelőző éjszakai hipoglikémia nélkül. A diagnózis felállításához éjszakai vércukormérés szükséges. Erre megoldást az inzulin adagok növelése jelenti. Szemben a Somogyi-effektussal, melyről akkor beszélünk ha az alacsony vércukrot 12-36 órán belül magas követi. [4]



2. ábra: A hipoglikémia és hiperglikémia leggyakoribb jelei

Ezen ismeretek alapján beláthatjuk, hogy egy olyan betegségről beszélünk, amely kiszámíthatatlan és a nem megfelelő kezelés esetén komoly szövődményekkel járhat. A probléma kézbentartására nagy segítségünkre lehet egy alkalmazás, mely használatával leredukálhatjuk a szövődmények kialakulását okozó tényezők számát.

3.3. Mobilalkalmazások

Mobiltelefonos alkalmazások már bő 15 éve könnyítik meg az emberek életét. A mobilapplikációk esetében kiemelten fontos szempont a gyorsaság, az egyszerű használat és a felhasználói igények minél több szempontból való kielégítése. Ezutóbbi alatt azt értem, hogy manapság már elterjedtek az olyan alkalmazások, amelyek szinte a tudtunk

nélkül elemeznek minket, és ezáltal kínálnak személyre szabott felhasználói élményt. Viselkedésünket elemezve egyes appok – mint például Instagram, Spotify vagy a Facebook – képesek azelőtt releváns információkkal ellátni minket, mielőtt mi arra gondoltunk volna.

Ezt felhasználva olyan mobilalkalmazást fejleszthetünk, mely képes elemezni, hogy az adott cukorbeteg felhasználó milyen tevékenységeket, meddig és mikor csinál annak érdekében, hogy karbantartsa a vércukrát. A szoftver tippekkel, tanácsokkal láthatja el a felhasználót, melyeknek teljesítése, illetve nem teljesítése esetén az eredményt megkapva újabb feladatokkal és célkitűzésekkel támogatja a felhasználót. A program a használóját annak környezetének megvizsgálásával is tud hasznos tanácsokat adni sportolási tevékenységekhez.

Több ilyen, kimondottan a cukorbeteg állapotot elemző alkalmazás is létezik már. Funkcióik többnyire megegyeznek: lehetőség alapadatok felvitelére, mérések és gyógyszerek rögzítése, sportolási tevékenységek és szénhidráttáblázat.

3.4. Hasonló rendszerek

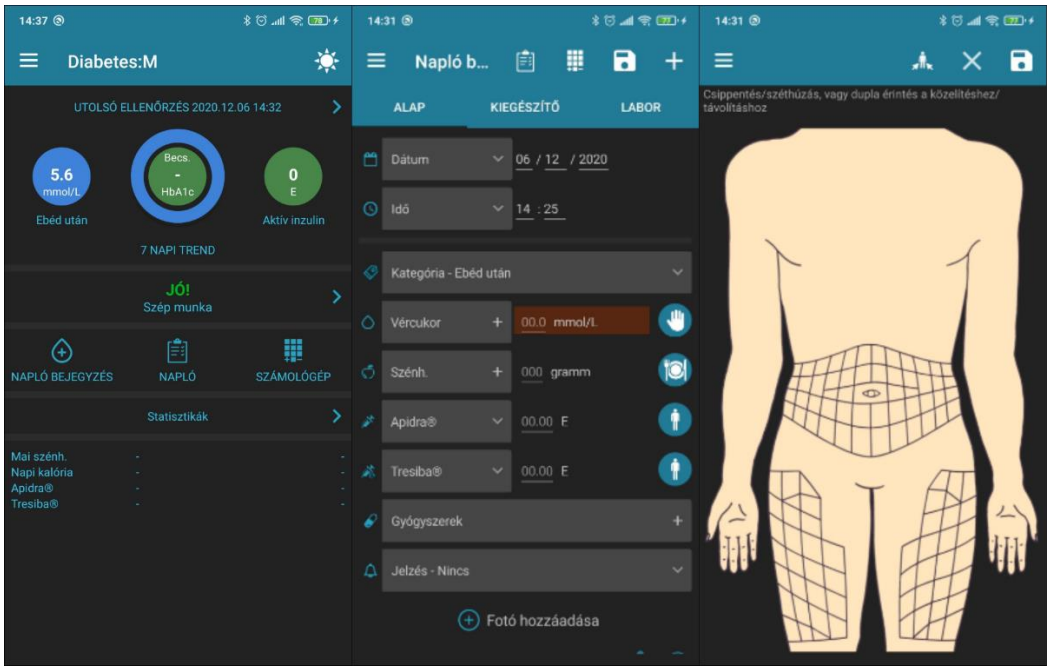
Természetesen létezik pár alkalmazás, amit referenciaként fel tudok használni az általam tervezett alkalmazáshoz. Olyan szoftvereket szeretnék bemutatni, melyeket magam is ki tudtam próbálni a dolgozat írása alatt. Mivel operációs rendszert tekintve Androidos készülékem van, illetve a megoldás is ezen a platformon készül, többnyire a Google platformján elérhető rendszereket vizsgáltam meg. [5]

3.4.1. forDiabetes

Elsőként a forDiabetes [6] alkalmazással kezdeném. Első pillantásra, ami szembetűnik az a komplexitás. Az alapadatok regisztrálásával kezd az app. Pár koppintással új mérési eredményeket vihetünk fel manuálisan, vagy akár bluetooth-on keresztül. Ezt az opciót sajnos nem tudtam kipróbálni, de mindenképpen hasznos lenne az általam tervezett alkalmazásban, ha nem kellene még csak hozzáérni sem a telefonhoz adatfelvitel közben. Lehetőséget ad arra, hogy sportolást is rögzítsünk, melyről több információval is szolgál. Képes HbA1c megsaccolására is. Tartalmaz kalória-táblázatot, diagramokat, illetve emlékeztetőket is beállíthatunk vele.

3.4.2. Diabetes:M

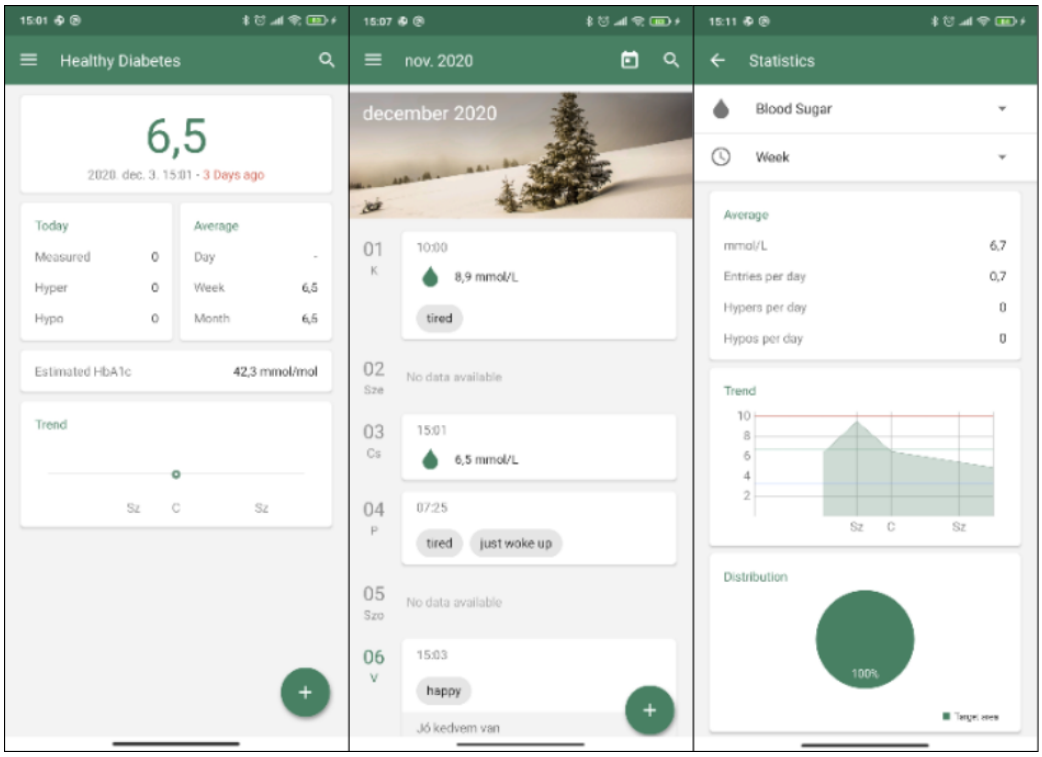
Létezik a Diabetes:M [7] alkalmazás, melyben hasonló lehetőségek vannak azzal a különbséggel, hogy itt nincs lehetőség aktív tevékenység felvitelére. Azonban itt lehetőségünk van követni azt, hogy mely testrészünkön hol adtuk be magunknak az inzulint, illetve hol mértük meg a vércukrunkat. Ez azért hasznos, mert idővel a szervezetben kialakulhatnak inzulin csomók, amelyek lassítják az inzulin felszívódását és nem lesz elég hatásos. Továbbá, az app több diagrammot is biztosít, hogy jobban tájékozódjunk az eredményeinkről.



3. ábra: Diabetes:M alkalmazás képernyőképe

3.4.3. Healthy Diabetes

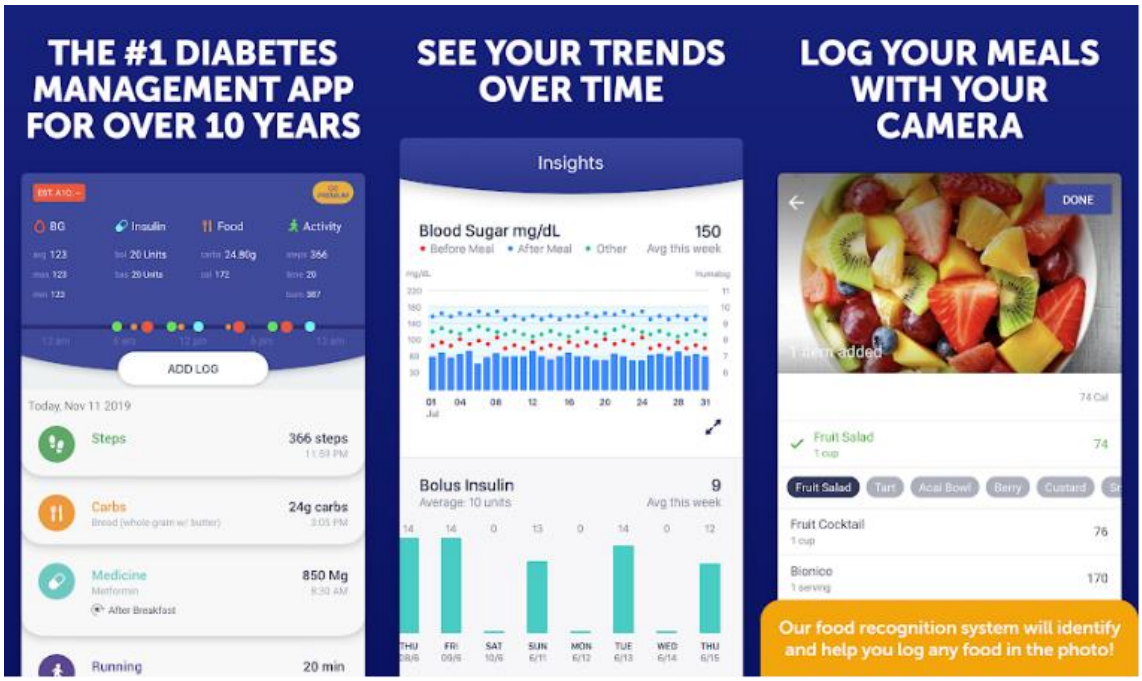
Ez az alkalmazás számomra főként az egyszerűsége miatt emelkedik ki. Megnyitás után a kezdőképernyőn egy grafikon jelenik meg, amely az elmúlt napok alapján rajzolt görbével szemlélteti a vércukor képünket. Fontosabb statisztikák, mint például a hipo-és hiperglikémia és az átlag mérések nap, hét, hónapra lebontva is itt jelennek meg. A többi alkalmazással ellentétben itt nincs lehetőségünk a személyes adatok felvitelére. Azonban mérések után egyszerűen ki tudjuk választani, hogy milyen közérzetünk van, mit étkeztünk, illetve automatikusan határozza meg, hogy étkezés előtt vagyunk, vagy utána. Természetesen lehetőség van ezen adatok felülírására. [8]



4. ábra: Healthy Diabetes alkalmazás képernyőképe

3.4.4. Glucose Buddy

Ezt az alkalmazást értékeli legtöbb pontra healthline.com és az everydayhealt.com kutatása. Mivel kizárólag iPhone-ra érhető el, ezért sajnos nem tudom tesztelni, hogy megfelel-e az igényeimnek, viszont egy kis olvasás után a főbb szempontok, amikről írnak az alapfunkciót mellett az a széleskörű élelmiszer-adatbázis, amely lehetővé teszi, hogy vonalkód alapján olvassunk be élelmiszereket, így eltárolva annak tápanyag tartalmát. Szinkronizálni tudjuk az Apple Health alkalmazással is, hogy könnyen nyomon követhessük fizikai aktivitásunkat [9].



5. ábra: Glucose Buddy alkalmazás funkciók

3.4.5. Összegzés

Előfordul még az Inzulin napló, illetve a Beat Diabetes alkalmazás, amik már inkább csak szimplán adatok rögzítésére alkalmas és előre leírt, nem személyre szabott instrukciókkal lát el minket. Az imént felsorolt szoftverek olyan funkciókat tartalmaznak, melyek nagyon hasznosnak bizonyulnak. Funkciók, amiket felhasználnék a fejlesztésben az a könnyed kezelhetőség, kalóriatáblázat, becslések a jövőbeli eredményről, emlékeztetők, illetve a sporttevékenységek tárolása és elemzése. Ezekhez pedig pluszban egy olyan komponens beépítését tervezem, amely a mért adatok alapján képes a felhasználónak egyénre szabott tanácsokat megjeleníteni. Ettől lesz ez a szoftver más, mint a többség.

3.5. Használt módszerek

3.5.1. Architektúra

Első sorban egy alapvető szempont, amit figyelembe kell venni a tervezés során az, hogy milyen architektúrán valósuljon meg az applikáció. Két lehetőséget érdemes megvizsgálni: kliens, vagy szerver-kliens struktúra.

Kliens architektúrát általában olyan alkalmazások tervezésénél használnak, amikor relatív kis mennyiségű adattal dolgozunk. Az ilyen rendszerek általában nem igényelnek hálózati kapcsolatot, illetve kevés erőforrással dolgoznak, hogy ne terheljék le a rendszert. Így működnek a különböző képszerkesztő, illetve multimédiás alkalmazások. Azonban akadnak olyan többjátékos szoftverek is, melyek esetében mi magunk lehetünk a kiszolgáló – ahol lényegében csak hálózati elérést biztosítunk, nem

pedig műveletigényes számítást egy kérelemre –, így játszva barátainkkal. Abban az esetben, ha a kommunikáció jelentős időkiesést (overhead) okozna, szintén érdemes csak a kliensre bízni a számításifolyamatokat, amennyiben nem korlátoz minket a hardver. Fontos tényező lehet a helytakarékosság. Ebben az esetben cél, hogy a kliens tárhelye minél kevésbé legyen leterhelve.

Szerver-kliens esetén a nagy, számítás- és erőforrásigényes folyamatokat a kiszolgáló, azaz szerver végzi nekünk. Ehhez alapvető, hogy legyen kapcsolat a két gép között és viszonylag elég gyors legyen a kommunikáció. A kiszolgálónál felmerülhet adatbázis használata is, mely megoldást adhat a tárhelyproblémára. Karbantarthatóságot szem előtt tartva hasznosabb, mint az előző megoldás, hiszen a szerveren végzett módosítások – javítás, frissítés – nincsenek hatással a kliensgépekre. Erre a rendszerre jó példa a webalkalmazások, illetve felhő alapú szolgáltatások is.

3.5.2. Adatbázis

Az alkalmazáshoz szükségem lesz egy adatbázisra, ahol biztonságosan eltárolhatók a felhasználók adatai. Erre főként abban az esetben van szükség, ha a felhasználó új eszközt használ, így a továbbiakban elég csak bejelentkeznie és az eltárolt adatok a szervertől továbbítódnak a telefon felé. Továbbá, ez lehetőséget ad egy webalkalmazás elkészítésére is, hogy a telefon mellett lehetőség legyen a számítógépen is elérni az appot. Továbbá, mivel újabb és újabb gyógyszerek jelennek meg a piacon, elég az adatbázisban frissíteni a táblákat ahhoz, hogy a készülékeken az új adatok is megjelenjenek.

Az adatbázisok az adatok tárolását tekintve két csoportra oszthatók: relációs (SQL) és reláció mentes (NoSQL) adatbázisok. Mivel az SQL az ACID (atomicitás, következetesség, elszigeteltség és tartósság) modellen alapul, ezért összetett lekérdezéseket is képesek vagyunk végrehajtani. A modellből következik még, hogy ha tartósan szeretnénk információt eltárolni praktikusabb választás lehet. A NoSQL ezzel szemben akkor használatos, ha ideiglenesen tárolunk adatokat – például online vásárláskor a kosár használata –, és ha nem tartalmaz az adatbázis tábla megszorításokat észszerűbb ezt választani [10,11]. Az IBM többek között a következő képpen értékelte a két típust:

	SQL	NoSQL
Előny	• Rugalmas lekérdezés • ACID	• Skálázható • Rugalmas adatmodellek • Nagy teljesítmény
Hátrány	• Merev adatmodellek • Vízszintes skálázás nem támogatott	• ACID hiánya • Osztott rendszerek problémája

6. ábra: SQL vs NoSQL

Amennyiben az adatok gyors írás/olvasására van szükség, akkor NoSQL-t, viszont, ha szeretnénk megőrizni az adatok integritását, akkor a relációs adatbázisok bizonyulnak hasznavehetőbbnek [12].

3.5.3. Tanácsadás

Napjainkban tisztában vagyunk azzal, hogy egyes alkalmazások – mint a Facebook, Instagram, YouTube, Netflix – mikor személyre szabott tartalmakat jelenítenek meg, a saját adatainkat használják fel, hogy virtuális élményeinket minél inkább az igényeinkhez igazítsák. Ez egyes vállalatoknak lehetőséget ad, hogy könnyebben elérjék ügyfeleiket, és akár visszatérő vásárlók legyenek. Egyes „on-demand" szolgáltatások ezt tökélyre fejlesztették, véleményem szerint a legjobb ilyen alkalmazás a Spotify. Az applikáció több olyan lejátszási listát is kínál, mely eddig meg nem hallgatott zenéket tartalmaz, amik nagy valószínűséggel elnyerik a felhasználó tetszését. Itt a lényeg: nagy valószínűséggel. Ez az app nem „erőszakolja rá" a felhasználóra a „megszerezni kívánt" élményeit. Lassan adagolva, nem erőltetve kínálja az új stílusú zenéket. Míg az előbb említett alkalmazásoknál – személyes tapasztalat alapján – gyakran előfordul, hogy egy bizonyos tartalom egyetlen megtekintése után – mely lehetett félrekattintás is – egyre többször dobja fel hasonló tartalmat, addig az utóbbi ezt okosabban csinálja. Az alkalmazásomban szeretném én is ezt a hasonlóan kimért „tanácsadást" implementálni. A felhasználó az én esetemben segítségre, útmutatásra vágyik, ezért figyelni kell arra, hogy minél sokatmondóbb, de semmi képpen se túl sokat mondó javaslatot kapjunk. Ez alatt azt értem, hogy ahány ember, annyi variációja létezik ennek a betegségnek, és az alkalmazás ne akarjon okosabb lenni, mint egy jó személyi asszisztens, jelen esetben egy diabetológus.

A kezelni kívánt betegségben léteznek minták, melyek felismerésével megelőzhetjük a betegség okozta szövődményeket. Erre a leghatásosabb és legegyszerűbb módszer a rendszeres vércukormérés. A Journal of Diabetes Science and Technology nevű folyóiratban a szerzők több vizsgálat eredményeit is összeszedték, melyben a páciensek vércukorszint menedzselését, szokásait vizsgálták [13]. Továbbá, személyes tapasztalatom, hogy a rendszeres sportolás és a diéta valóban nagyban befolyásolják a vércukorszintet. Nem beszélve azokról a dolgokról, amelyeket mi magunk nem tudunk befolyásolni és stresszel járnak [14].

4. SPECIFIKÁCIÓ

4.1. Célkitűzés

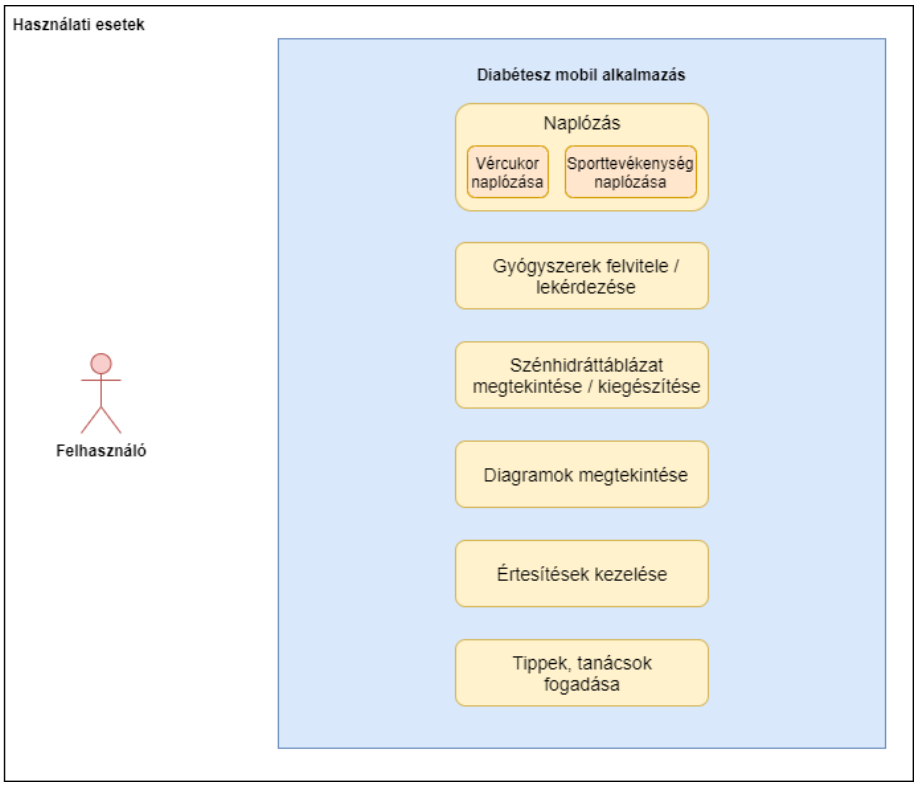
A cél egy olyan alkalmazás elkészítése, amely a felhasználókat személyre szabott tippekkel, segítséggel látja el. A rendszer a diabétesz típusától függően képes az adatok elemzésére, illetve azok – a körülményeket tekintve – helyes felhasználására. Ez alatt azt értem, hogy továbbá a külső tényezőket – melyek a felhasználótól függetlenek, ilyen az időjárás, stressz, a napszak, melyek a felhasználó életvitelétől önállóan működnek –

figyelembevéve képes releváns tanácsokkal ellátni a felhasználót. Egy alapfunkció a vércukorértékek naplózása lenne. Az egyszerűség végett a felhasználónak nem kell törődnie azzal, hogy rögzítse az időpontot, mert ez a rendszer feladata lesz. Neki csupán az értéket kell felvinni és hogy étkezés előtt vagy után történt a mérés – ez utóbbi szintén automatizálható lépés. További cél az inzulin- vagy gyógyszeradagok felvitele és figyelése. Ezt elég a legelső használatkor felvinni, különben már csak a korrekció mennyiség felvitelére van szükség. Funkció lesz a sportolási tevékenységek dokumentálására is, ahol kiválaszthatjuk magát a sportot, illetve az azzal eltöltött időt is. A rendszer ezek, valamint a testtömeg alapján kalkulálja ki az elégetett kalóriát, de lehetne itt használni akár más mértéket is. Mivel kiemelt figyelmet kap a rendszer egyszerű használata, gesztusokkal (például az ujjal különböző irányokba való söprése) lehetne navigálni a menüben és az adatok rögzítésénél. A felhasználó számára rendelkezésre állnak majd grafikonok – mondjuk az elmúlt heti sporttevékenységének a függvényében kialakult vércukráról, vagy az alap adatok (testsúly, vérnyomás) alakulása a vércukor változásának függvényében –, amik szintén segítségül szolgálnak majd.

Fontos célom az imént említetteken felül a felhasználó számára szolgáltatott adatok egyénre szabottsága. Ez pedig a tanácsadó komponens feladata lesz. Ez fogja végezni a kapott adatokból a számunkra értékes információk kinyerését és feldolgozását. Mivel a cukorbetegség egy összetett állapot, amit nagyon sok befolyásoló tényező alakít, szükségét érzem egy sokrétű, összetett komponens felhasználására, mely képes ezeket a problémákat kezelni, ezáltal a beteg személyes asszisztensévé válni.

4.2. Alkalmazás feladatai

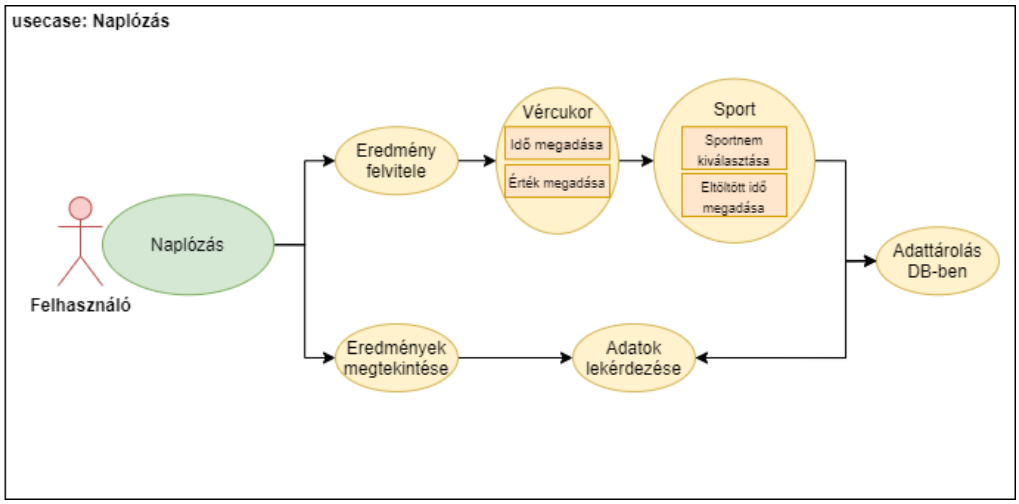
Ahhoz, hogy megkezdhessük a tervezést, szükséges ismernünk, hogy a rendszerünknek milyen feladatokat – használati eseteket – kell ellátnia. Ennek ábrázolására van a következő használati eset diagram:



7. ábra: Használati eset diagram

4.2.1. Naplózás

Az eredmények naplózását két részre: sportolás és vércukor eredményekre oszthatjuk. Ezt az alapfunkciót a felhasználó egy menüponton keresztül éri el, melyre rábökve a rendszer egy oldalon jeleníti meg a szükséges beviteli mezőket. A beteg alapbeállításait alapul véve – amit legelső belépésnél megad – jelenik meg a mértékegység (mmol/l vagy mg/dl). Az érték megadását követően kiválaszthatjuk a mérés idejét: étkezés előtt/után, vagy ha nem abban a pillanatban mértünk, megadhatjuk a napszakot is. Ezt követően lehetőség van kiválasztani, hogy a felhasználó alkalmaz-e valamilyen korrekciót a mért értékek mellett. Aktivitás naplózása esetén lehetőség van bevinni a sportnemet, illetve az azzal eltöltött idő mennyiségét. Ezek felvitele után megkapjuk, mennyi kalóriát égettünk, illetve a rendszer később ábrázolni fogja, mennyire hatásos volt az aktív időtöltés.

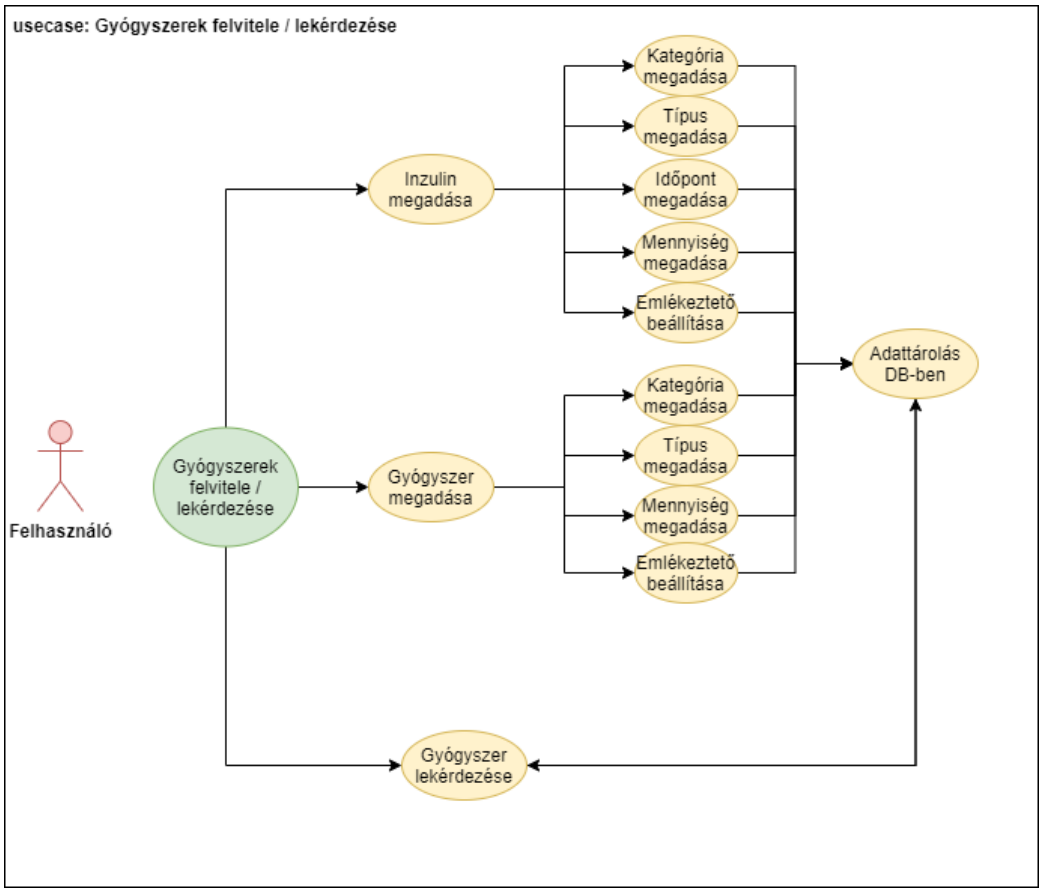


8. ábra: Naplózás használati eset diagram

4.2.2. Gyógyszerek felvitele / lekérdezése

A beteg inzulin adagja és gyógyszerei – révén, hogy 1-es, illetve 2-es típusú diabéteszes lehet – felvitelére is szükség van. Első belépésnél kiválaszthatja a gyógyszer fajtáját (inzulin esetében gyors és bázis inzulinét), a mennyiségét, és hogy rendszerit milyen időpontokban étkezik. Inzulin esetében létezik analóg, illetve humán inzulinok, melyek hatásukat tekintve eltérők lehetnek. A két csoporton belül megkülönböztetünk gyorshatású – étkezések előtt használt – és bázis – elnyújtott hatású – inzulint. Gyógyszeres kezelés esetén alapvetően szintén kétféle létezik: inzulinhatást javító és inzulinelválasztást serkentő szerek [15]. Ezért is fontos a gyógyszerek rögzítése. Ha esetlegesen egy új termék jelenik meg a piacon és a beteg azt használná, így tudja, hogy az előző gyógyszere mi volt, és mennyit szedett belőle. Persze lehetőség van az adatok későbbi módosítására, amennyiben változnak azok.

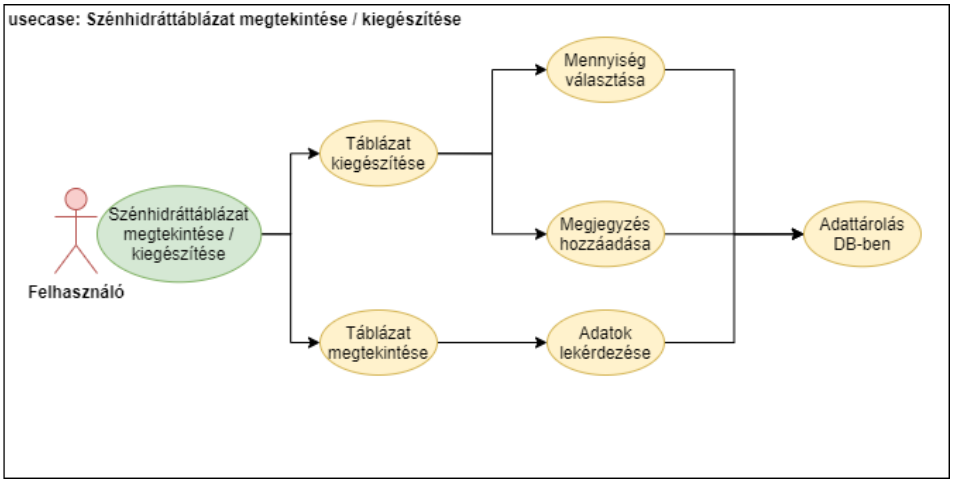
1-es típusú betegeknél a felhasznált inzulin a bevitt étellel arányosságban állnak. A másik esetben lehetőséget adhatunk arra, hogy a rendszer figyelmeztesse a felhasználót a gyógyszer beszedésének esedékességéről.



9. ábra: Gyógyszerek felvitele / lekérdezése használati eset diagramja

4.2.3. Szénhidráttáblázat megtekintése / kiegészítése

Mivel egy cukorbetegnek elengedhetetlen a megfelelő diéta, segítségére lehet egy szénhidrát-és kalória táblázat, melyből könnyen tájékozódhat. Segítségével könnyebben felmérheti a megfelelő mennyiségű ételadagokat. Ebbe a menüpontba navigálva lehetőség van az adatok közti böngészésre, illetve keresésre.



10. ábra: Szénhidráttáblázat megtekintése / kiegészítése használati eset diagram

4.2.4. Diagramok megtekintése

A felhasználó nagy mennyiségű adatot fog eltárolni a szerveren, így ezek könnyebb olvashatósága, és a konklúziók levonása végett diagramok segítik a felhasználót. Ez szintén segíti a gyógyszer megfelelő mennyiségének kialakítását, illetve a beteg a diabetológusával konzultálhat az eredményekről.

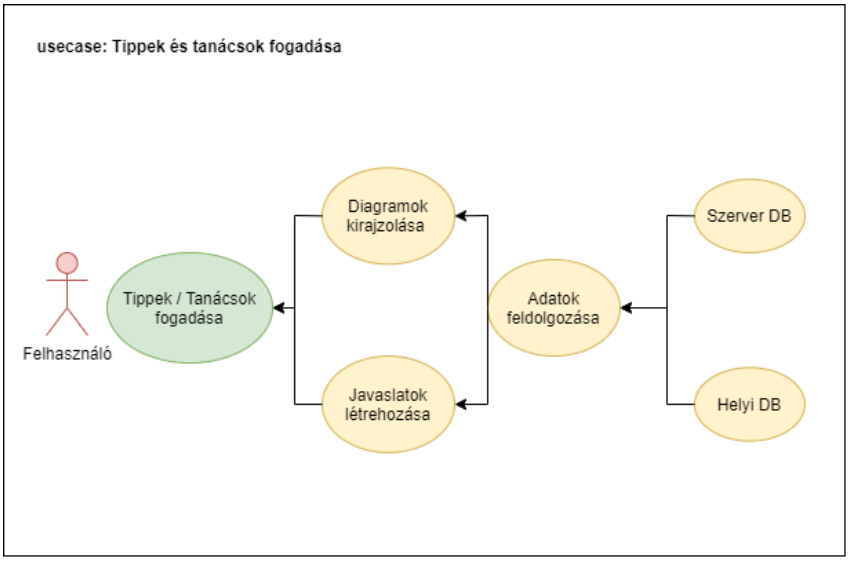
4.2.5. Értesítések kezelése

A páciens egyénileg beállíthatja, milyen értesítéseket szeretne kapni. Beállíthatja annak fontosságát és gyakoriságát is.

4.2.6. Tippek és tanácsok fogadása

Ez a funkció a felhasználótól függetlenül működik, viszont mivel ez a komponens lesz a rendszer lelke, a beteg főként erre az alkotóelemre fog leginkább támaszkodni. A tanácsadó modul a kapott adatok alapján egyénre szabott tanácsokkal látja el a felhasználót. Mivel sok befolyásoló körülmény alakíthatja a vércukrot, a betegnek az abban a helyzetben legrelevánsabb segítségre van szüksége.

A rendszer figyelmeztetéseken keresztül lép kapcsolatba a felhasználóval, aki az adatokat szolgáltatja számára. Ezek mellett nem csak a felhasználóval, hanem a webbel is szükséges kapcsolatot tartani, mivel a betegtől független tényezőket, mint például az időjárás, napszak, stressz. A mai világban egyre inkább meghódítja életünket az online tér, a social media, ezért a felhasználó képernyő idejének profilozására is szükség lehet a minél relevánsabb adatok szolgáltatásának érdekében.

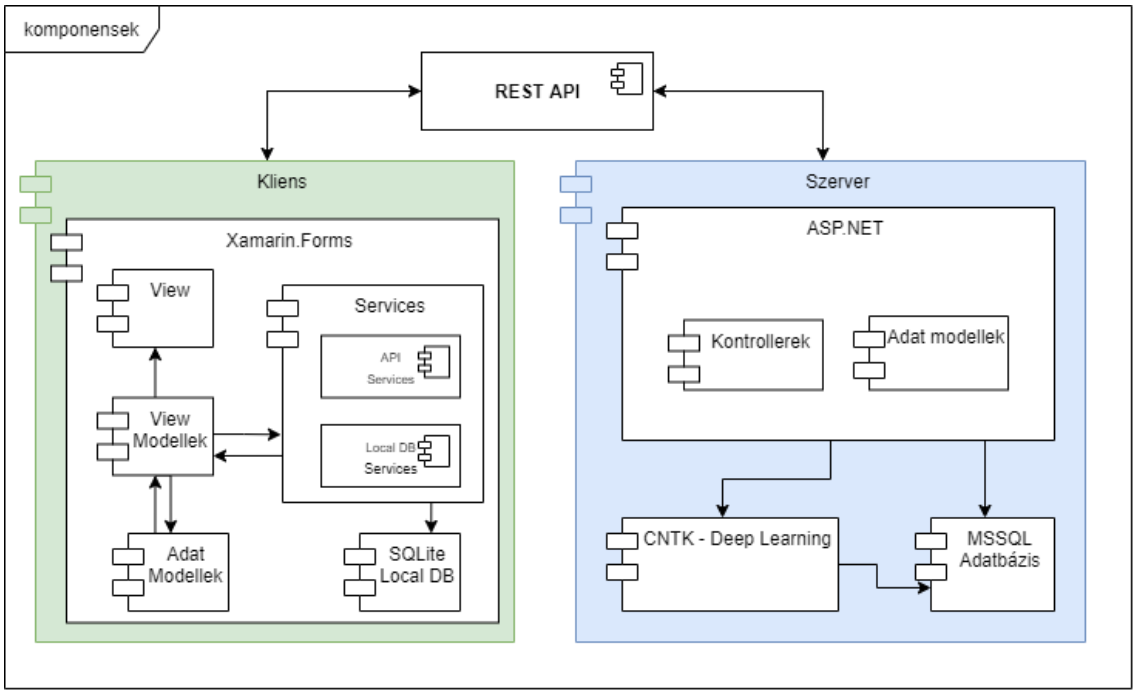


11. ábra: Tippek és tanácsok használati eset diagram

5. RENDSZERTERV

5.1. Komponensek

Az általam készített alkalmazás két fő komponensből áll: szükség van egy szerveroldali, illetve egy kliens alkalmazásra.



12. ábra: Komponens diagram

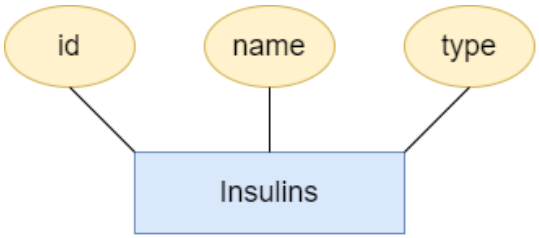
5.2. Szerver

A kiszolgáló modul több kisebb komponensből áll. Azon feladatok elvégzése miatt jött létre, amelyek sok számítási igénnyel bírnak, és ezek eredményeit küldi el a felhasználónak. Ezen kívül ide tartozik még az authentikáció (felhasználók hitelesítése), authorizáció (felhasználói jogok), amire a kliens önmagában nem képes.

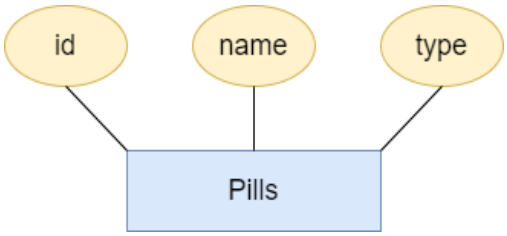
5.2.1. Adatbázis

Adattárolásra a szerveren egy MSSQL adatbázist használok. Ez fogja tartalmazni a felhasználókat a belépéshez szükséges adataikkal. Itt tárolódnak a felhasználói profil adatai és a mérések eredményei is.

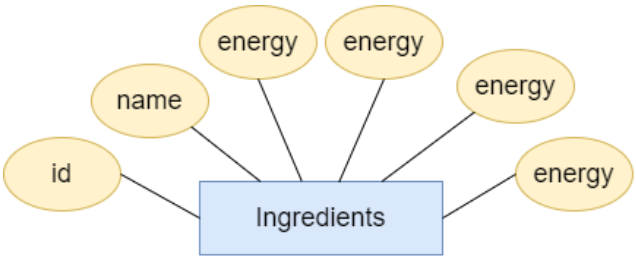
Mivel a felhasználónak lehetősége van offline használni az alkalmazást, ezért itt csak azok az adatok jelennek meg, amiket a kliens alkalmazás időnként feltölt ebbe a „felhőbe”. Majd, ezt követően a szerver képes ezekkel az adatokkal dolgozni, azokat kiértékelni és figyelmeztetéseket / tanácsokat küldeni a felhasználónak. Az authentikáció / authorizáció elvégzéséhez szükséges táblák mellet a következő táblákra lesz szükség: inzulinok, gyógyszerek, összetevők, mérési eredmények, sportnapló.



13. ábra: Inzulinok tábla

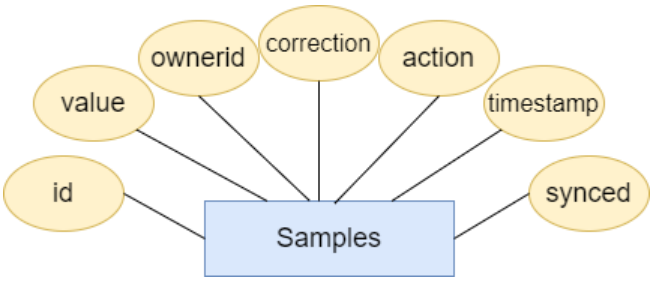


14. ábra: Gyógyszerek tábla



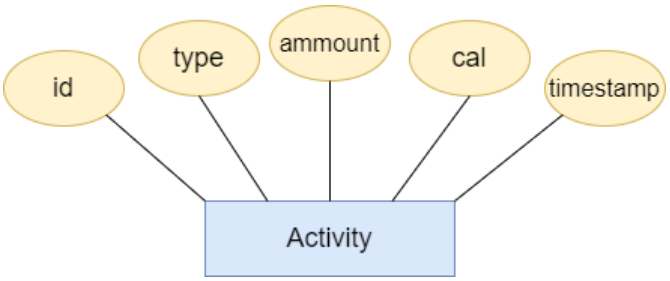
15. ábra: Összetevők tábla

Ezek azok a táblák, amelyekre a felhasználóknak csak olvasási jogosultsága van. A táblák adatai a kliensalkalmazás első indítása után letöltődnek a készülékre, és onnantól már nem lesz szükség a szerver eléréséhez, ha látni szeretnénk ezen táblák tartalmait.



16. ábra: Mérési eredmények (vércukornapló)

Ez a tábla tartalmazza a mérések eltárolásához szükséges további adatokat, mint a korrekció vagy hogy milyen aktív tevékenységet végzett a felhasználó a mérés után. Szükség van egy olyan mezőre is, amiből megállapíthatjuk, hogy szükséges-e az adatokat a felhasználó eszközével szinkronizálni, vagy sem.

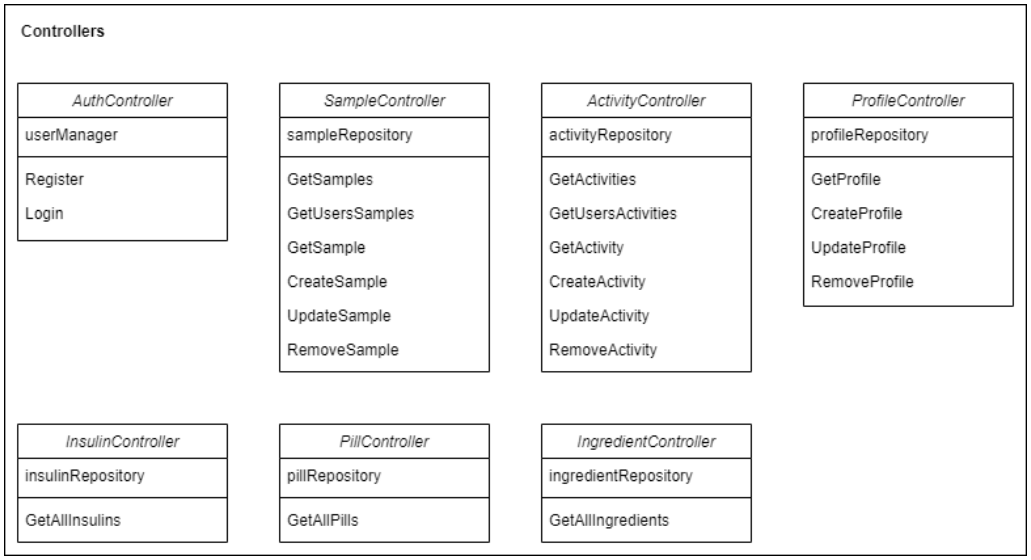


17. ábra: Aktivitás

Az aktivitás táblában rögzítésre kerülnek a felhasználók aktív tevékenységei. Legfontosabb mezői a sportnem, mennyiség (idő) és a sportolással elégetett kalória.

5.2.2. Kontrollerek

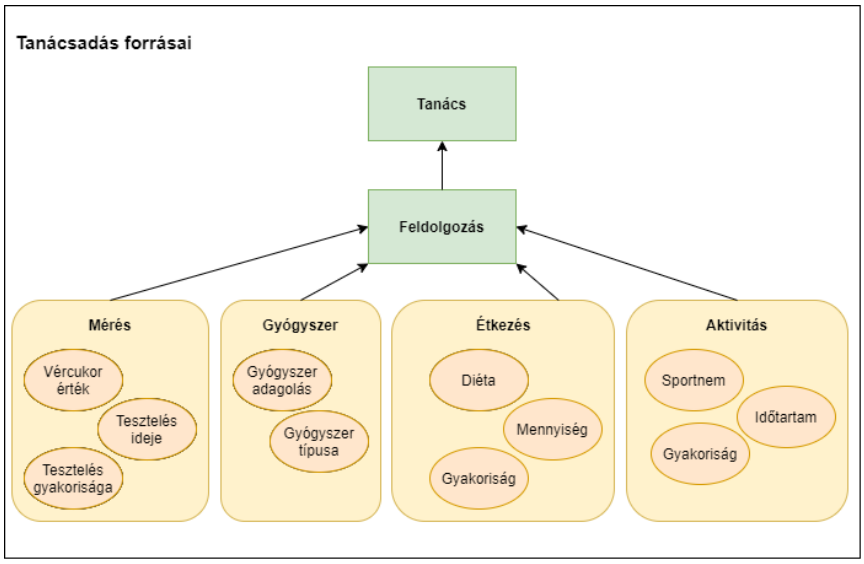
Az adatok továbbítása a kliens és a szerver közt REST API hívásokon keresztül történik. Az alkalmazásban minden adattáblához külön kontrollert hozok létre az egységbezárás miatt.



18. ábra: Kontrollerek

5.3. Kliens

A kliens oldal fő feladata, hogy API kéréseket intézzen a szerver felé – mely adatot szolgáltat –, így információkkal lássa el a felhasználót. Ide tartoznak a diagrammok is, amiket vizualizálva teljesebb képet kapunk adatainkról, amely segíti tervezéseinket a jövőben. A tanácsadó komponens előrejelzéseket és javaslatokat készít, amikkel figyelmeztetheti a felhasználót, tanácsokat ad számára a normális vércukorszint fenntartására. Ide tartoznak a különböző diagrammok, amiket a felhasználó a saját preferenciája alapján – egyszerre többet – tekinthet meg, így könnyebben vizualizálva az adatokat.



19. ábra: Tanácsadás forrásai

5.3.1. Helyi adattárolás

Offline mód esetén az adatok a telefonon tárolódnak. Ehhez egy könnyen, gyorsan végbemenő tárolási módra lesz szükség.

6. SAJÁT MEGVALÓSÍTÁS

6.1. Módszerek a megvalósításhoz

Az általam készítendő mobil applikációban fontos szempont a megbízhatóság és perzisztencia, ezért a rendszer kliens-szerver architektúrán alapul. Léteznek lekérdezések, melyeket önmagában a kliens nem tud kiszolgálni. Ekkor kénytelen a szerverhez fordulni, majd azokat visszaküldve a kliensnek történik a kommunikáció a két gép között. Erre egy példa az első indítás, mikor a gyógyszerek/inzulinok nem eleve az eszközön tárolódnak – hiszen szeretnénk, hogy up-to-date listával rendelkezzünk a már piacon levő orvosi szerekről. Itt említendő a biztonsági mentés, mellyel a felhasználó biztonságban tudhatja adatait egy esetleges telefoncsere után is. Szem előtt tartva a tárhellyel való takarékosságot is ez a megoldás bizonyul kézenfekvőnek [16].

A szerveren fut egy adatbázis kezelő alkalmazás, ahonnan a lekérdezett adatokat továbbítjuk a kliensgép felé. Felmerülhet a kérdés, hogy SQL vagy NoSQL adatbázisban tároljuk az adatokat? A 2013-ban zajlott IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM) konferencián összehasonlították a két típust. Az összehasonlítás során arra jutottak, hogy bizonyos műveletekben gyorsabb a NoSQL, viszont hozzáteszik, hogy a teszteket nem hajtották végre komplexebb adatbázis műveleteken [13]. Cameron Prudy, az Oracle volt alelnöke szerint: „A NoSQL azért hasznos, mert sok alkalmazást felépíthetünk anélkül, hogy ad hoc műveleteket (join, update) használnánk bennük, így kevés adatbázis funkció

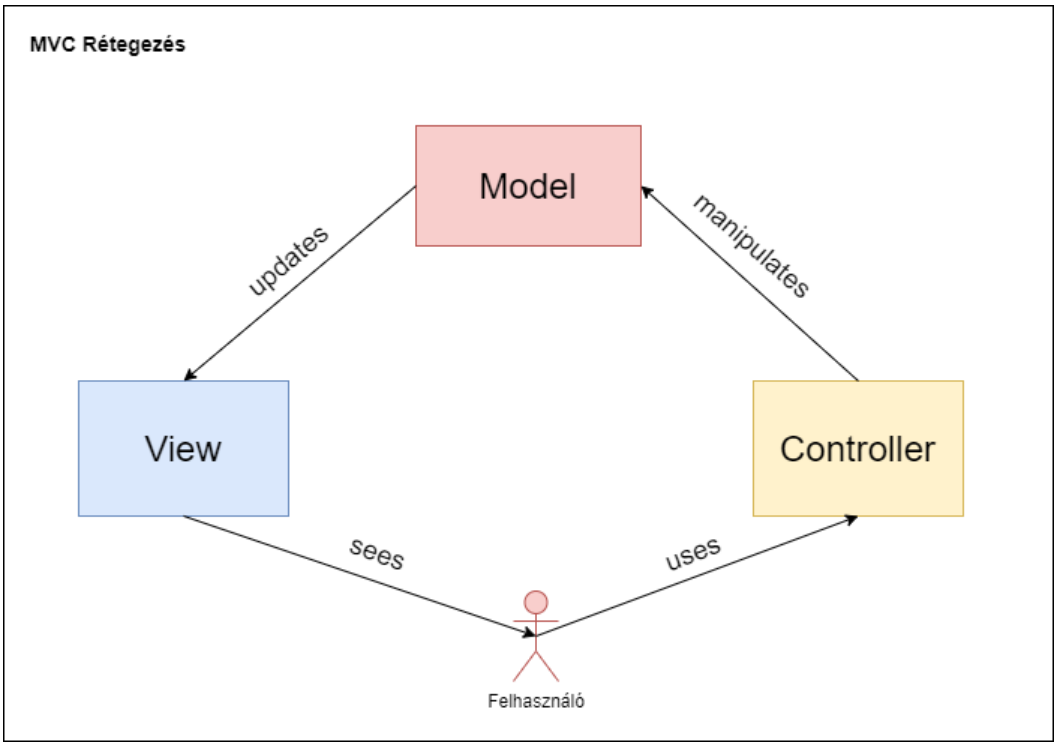
használatára van szükség.” Továbbá kiemeli, hogy vannak olyan műveletek, melyek relációs (SQL) adatbázisban könnyen végrehajthatók. Ezzel szemben NoSQL-ben lassan, vagy egyáltalán nem lehetségesek ezek a műveletek [14]. Ezek alapján arra a következtetésre jutottam, hogy SQL-ben, mivel nem egy BigData – többmillió rekordból álló adattárházat fenntartó – alkalmazást, hanem párezer rekordos táblákban alapműveleteket gyorsan elvégző alkalmazást szeretnénk. NoSQL adatbázisok esetén célszerű lehet CouchDB vagy Couchbase használata, viszont a CRUD – létrehozás, olvasás, írás, törlés – műveletekhez ekkora adathalmaz esetén még mindig célszerűbb az imént említett relációs adatbázis használata. Ezen felül a szerver HTTP és FTP kéréseket is teljesít, mivel célszerű lehet a későbbiekben lehetőséget adni az adatok exportálására az adatbázisból.

A kliensoldali alkalmazás fejlesztéséhez több fejlesztői környezet közül választhatunk. Ezek közül véleményem szerint a Facebook React Native-ját és a Microsoft által fejlesztett Xamarin-t érdemes kiemelni. A „crossplatform” fejlesztést tekintve mindkettő ugyanolyan hasznosnak tűnik. A Xamarin fordítási ideje és hatékonyságot nagyon meggyőző, illetve a Xamarin.Essentials könyvár hasznos API-kat tartalmaz a fejlesztéshez. A React mellett szól, hogy egy közösség vezérelt keretrendszer, így támaszkodhatunk a kollektív alkalmazásfejlesztés előnyeire.

6.2. Felhasznált technológia és funkciók

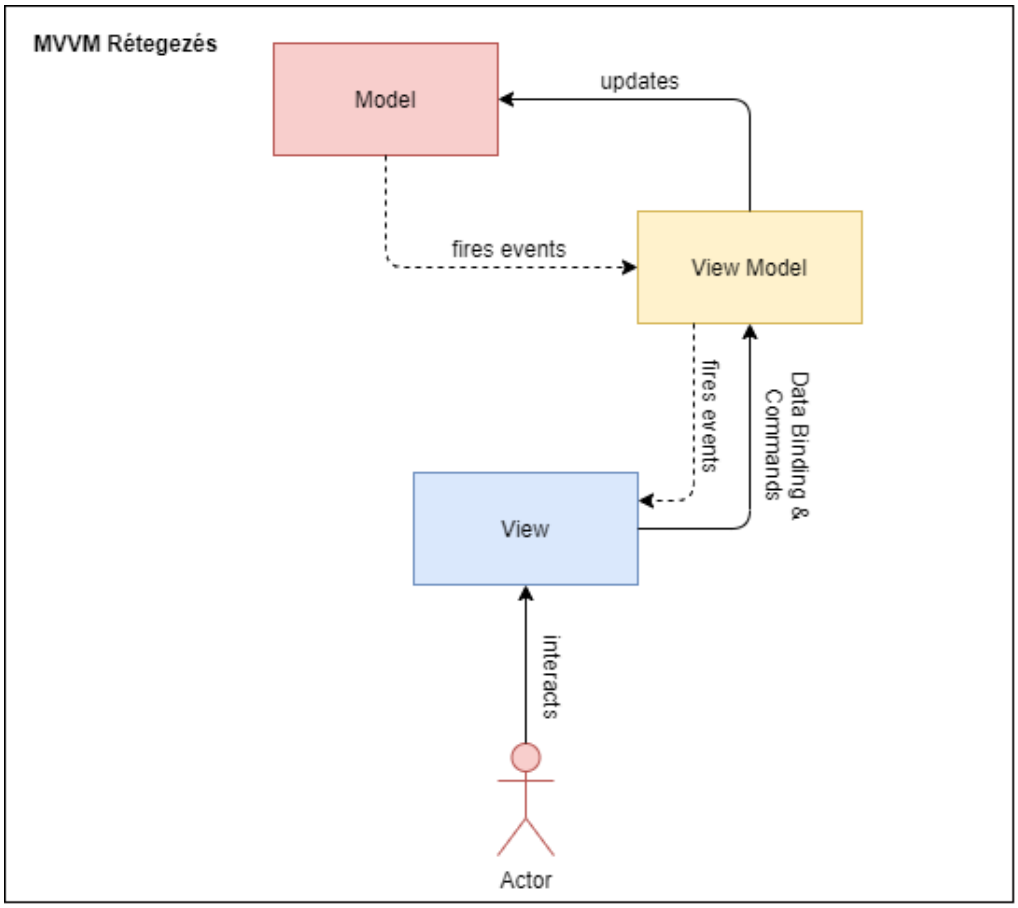
6.2.1. Tervezési minta

Az általam készített program szerveroldali komponense egy Web API, mely részben MVC tervezési mintán alapul. Ez azt jelenti, hogy logikai felépítést tekintve a backend alkalmazás 3 részre osztható: adatréteg (Model) – ez a réteg felelős az üzleti logika kezeléséért, illetve a hálózaton történő adatok tárolásáért –, nézet (View) – UI réteg, megjeleníti az adatréteg információit –, vezérlő (Controller) – ez a logikai réteg értesítést kap a felhasználó kéréseiről és továbbítja azokat a Model réteg felé, illetve vissza a kliensnek. Azonban, a View-t jelen esetben nem a .NET-es környezet, hanem egy Angular JS-ben írt TypeScript webalkalmazás szolgáltatja. Ez a réteg egy email service prezentálására szolgál. A tervezési mintát ASP.NET keretrendszerrel valósítom meg.



20. ábra: MVC tervezési minta

A kliensoldal MVVM mintára épül. A felhasználó a View-val kommunikál, az azon elhelyezett vezérlők, melyekkel interakcióba tud lépni a ViewModel réteggel vannak adatkötéssel összekötve. Ez a réteg írja le a tartalmat, amelyet a felhasználó a képernyőjén lát, és kapcsolatba tud vele lépni. A módosított adatok a Model rétegben eseményen keresztül értesítik a VM-et, az pedig a View-t, hogy frissítse a képernyőt.



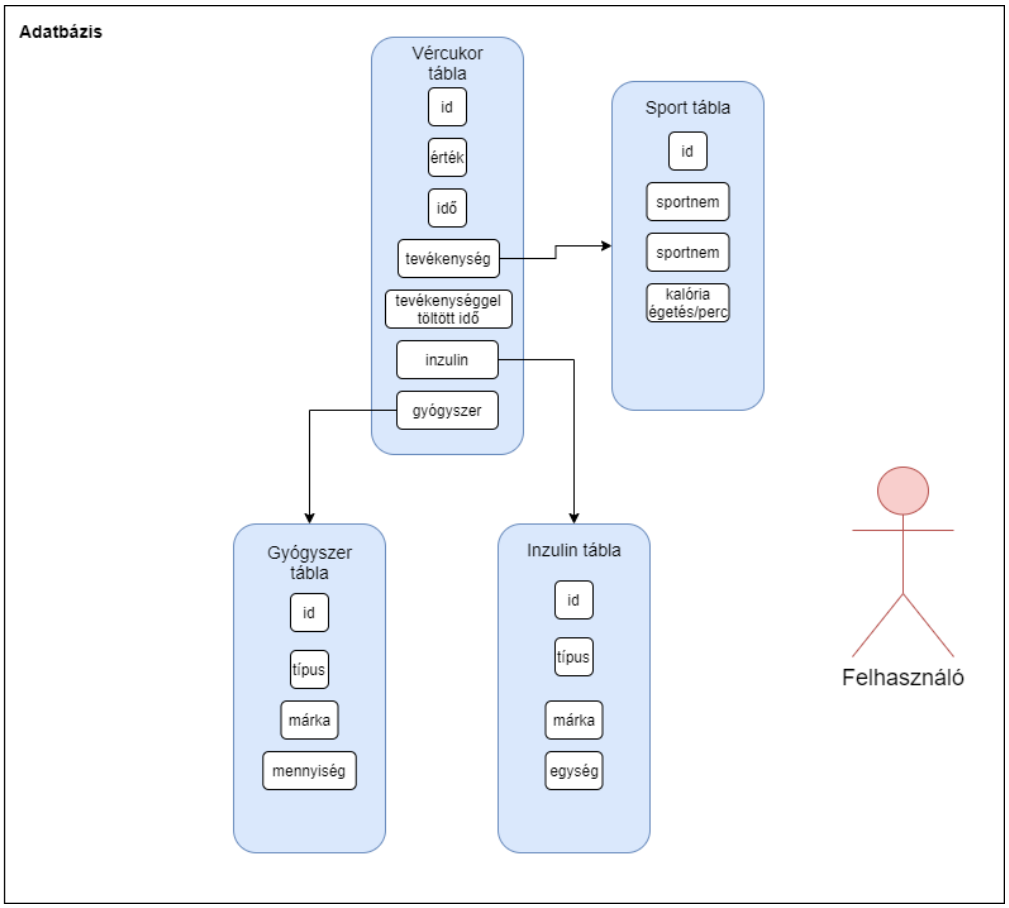
21. ábra: MVVM tervezési minta

6.2.2. ASP.NET

A keretrendszer kifejezetten webalkalmazások készítésére lett tervezve. Kiválóan alkalmazkodik az MVC tervezési mintához. Mivel az authentikáció is a program része, szükség van egy olyan alap keretre, amely külső hitelesítést is támogat a könnyebb felhasználás érdekében. Továbbá Windows mellett Linux-on, macOS-en és Docker rendszereken is működik. Ez lehetőséget ad később egy platformfüggetlen továbbfejlesztésre is [15].

6.2.3. MSSQL Database

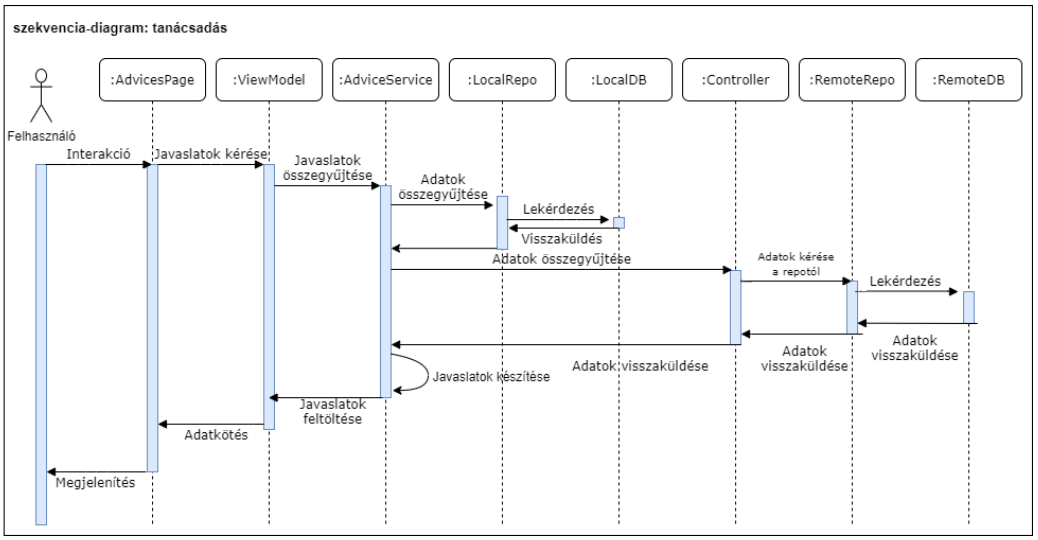
Az adatok tárolására egy MSSQL adatbázis lesz a Model réteg segítségére. Innen fogja a Controller visszaszolgáltatni az adatokat a felhasználó számára.



22. ábra: Adatbázis komponensei

6.2.4. Tanácsadás

A tanácsadó modul a Xamarin.Forms alkalmazásba lesz integrálva. Egy service fog gondoskodni a helyi és távoli adatok lekéréséről, feldolgozásáról és azok UI, pontosabban a View Model rétegbe történő szállításáról. A View-ban a diagramok és a tanácsok megjelenítésére szolgáló oldalak fogják ezeket megjeleníteni. Az egyes tanácsok szövegeit .xml fájlban tárolom el, így a service innen fogja kiolvasni a súgó szöveget, ha a felhasználó rányom egy listaelemre.



23. ábra: Tanácsadás szekvencia diagram

6.2.5. Fejlesztés

A fejlesztést Visual Studio 2022 fejlesztői környezetben végzem. Azért ezt a környezetet választottam, mert az Android alkalmazás implementálásához is egy .NET-es keretrendszert szeretnék használni, ez nem más, mint a Xamarin.Forms.

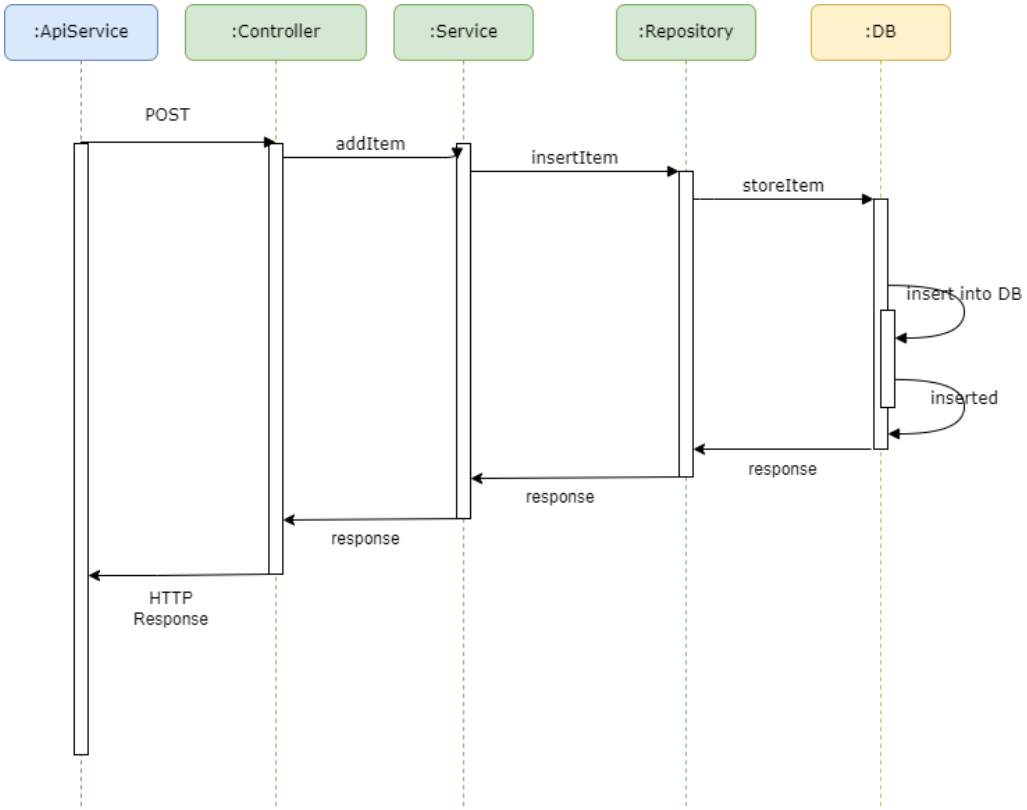
Véleményem szerint az utóbbi manapság kezd elavult lenni, helyét átvette a React Native, a Flutter és az Ionic. Azonban a redmondiak 2021 novemberben kiadták .NET 6 keretrendszert, mely jó lehetőséget ad a Xamarin utódjának, a .NET MAUI-nak. Mint ígérik, ez lehetőséget ad xamarinos alkalmazásaink továbbfejlesztésére anélkül, hogy bajlódnunk kellene a migrációval, amihez egy .NET Upgrade Assistant nevű segédeszközt is kapunk.

A szerveroldal tartalmazni fog egy Mail Service-t is, mely képes emailek kiküldésére a felhasználónak. Ez a szerveren tárolt adatok közlésére, vagy egészségügyi innovációk, új gyógyszerek bejelentésére is használható lesz. Mivel nem ez a szakdolgozat fő profilja, ezért ennek bemutatására kisebb hangsúlyt fektetek. Az ehhez tartozó View-t Angular használatával, TypeScript nyelven írom.

7. IMPLEMENTÁLÁS

7.1. Szerver

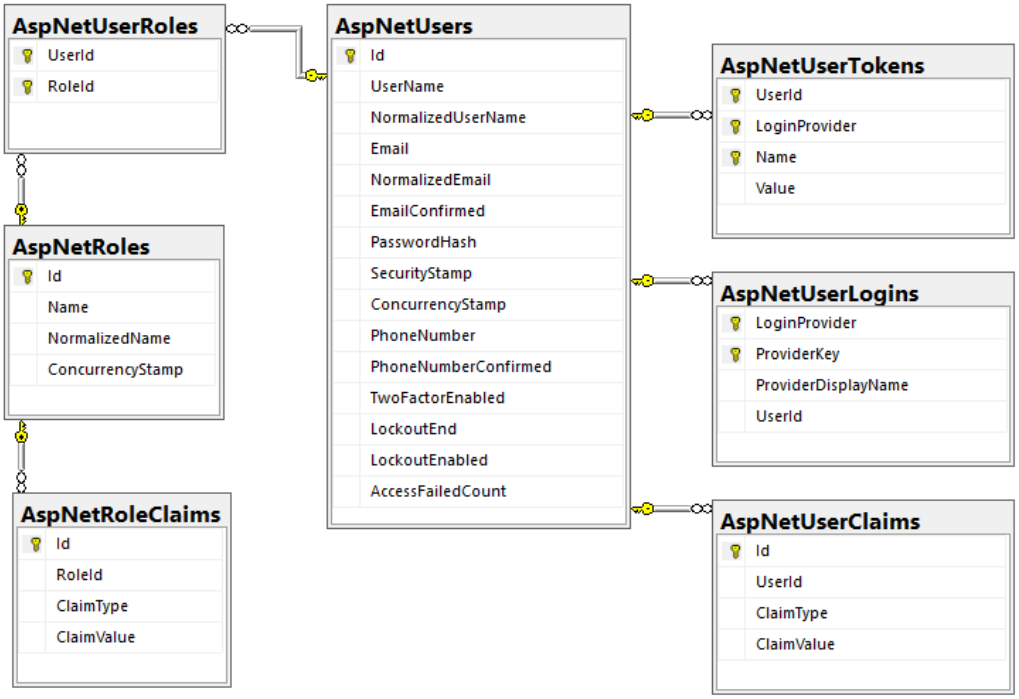
A szerver lényegében egy Web API, amihez kérést intézve a felhasználó adatokat kérhet le vagy vihet fel a telefonjáról. Ezen felül a szerver képes lesz emailek küldésére, melynek szimulálására a backend oldalon egy *Mail Service* lesz a segítségünkre, frontend oldalon pedig egy egyszerű, TypeScript-ben írt Angular webalkalmazás. A végrehajtandó kérést a szerver megkapva megpróbálja azt végrehajtani. Ennek sikerességéről értesíti a klienst.



24. ábra: API hívás szekvencia diagram

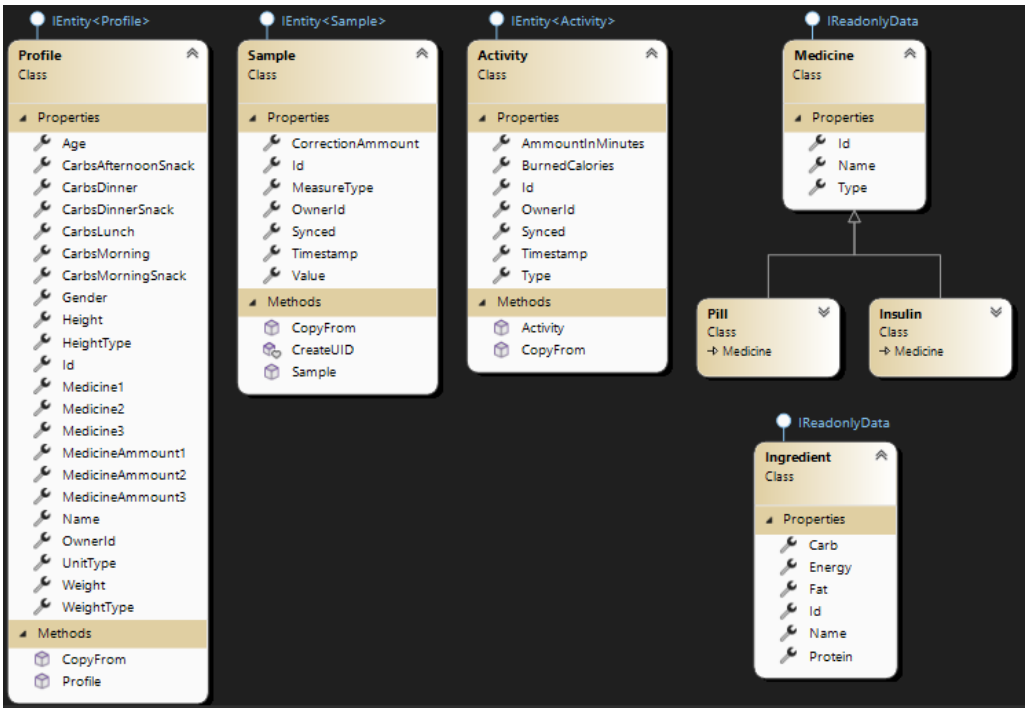
7.1.1. Adatok

A szerver adatmodelljeit főként a RDBMS rendszerben eltárolandó adatok alkotják. Az authentikációt és authorizációt az ASP.NET keretrendszer ASP.NET Core Identity API segítségével oldom meg. Ez a könyvtár segít a felhasználók, jelszavak, szerepkörök (jogosultságok), tokenek menedzselésében. Migrálást követően az adatbázisban a következő táblákat kapjuk:



25. ábra: ASP.NET Core Identity táblák az adatbázisban

Ezek mellett az általam létrehozott Entity-k tárolására is szükség van, hogy a felhasználó bármikor elérhesse azokat. Minden felhasználóhoz tartozik egy *Profil*, mely a felhasználó betegséggel kapcsolatos adatait tartalmazza. Ide tartozik például nem csak a súlya, kora, de az általa használt gyógyszerek adatai is. Kétféle gyógyszer tárolását – az *Insulin* és *Pill* – kell megoldani, melyek tulajdonságaikban megegyeznek, ezért egy közös ősből, a *Medicine* osztályból származnak le. A felhasználó tájékozódására segítségül lesz az *Ingredients* tábla, amely az egyes alapanyagok tápanyagtartalmát hivatott eltárolni. A legfontosabb adatmodell a mérés és az aktivitás – *Sample* és *Activity*, – ugyanis a kliens ezek alapján fogja tanácsokkal ellátni a felhasználót.



26. ábra: Backend osztály diagram

Az adatok továbbításáért a különböző *Service*-ek és *Repository*-k lesznek a felelősek. A korábban említett *MailService* egy egyszerű interface-t valósít meg, aminek egyetlen feladata az emailek kiküldése az adatbázisban tárolt emailcímekre.

```
7   namespace ApiBase.Data.Interfaces
8   {
9       public interface IProfileRepository
10      {
11          void Create(Profile profile);
12          Profile Get(string id);
13          IQueryable<Profile> GetAll();
14          IQueryable<Profile> GetByOwner(string uid);
15          void Remove(string UID);
16          void Update(Profile profile);
17      }
18  }
```

27. ábra: ProfileRepository interfész

7.1.2. Kontrollerek

Kontrollerek esetében fontos, hogy olyan végpontokat is definiáljanak, melyeket csak regisztrált felhasználók tudjanak elérni. Például, ne tudjon egy teljesen ismeretlen felhasználó olyan kérést intézni, mellyel az összes felhasználót megkapja. Sőt, egyesekre csak az adminisztrátor legyen képes. Az *ASP.NET* eléggé megegyszerűsíti nekünk a dolgunkat ebben az esetben, ugyanis a kontroller-t elég egy *„Authorize”* dekorátorral

ellátni, ami paraméterként megkapva a *„Role”* -t csak az adott szerepkörrel rendelkező felhasználók számára teszi elérhetővé a kontrollert vagy metódust.

```
13  namespace ApiBase.Controllers
14  {
15      [Authorize(Roles = "Admin")]
16      [ApiController]
17      [Route("[controller]")]
18      public class AdminController : ControllerBase
19      {
20          private readonly UserManager<IdentityUser> _userManager;
21          private readonly RoleManager<IdentityRole> _roleManager;
22          private readonly Microsoft.Extensions.Configuration.IConfiguration _configuration;
23          private readonly IHubContext<EventHub> _hub;
24
25          public AdminController(UserManager<IdentityUser> userManager, RoleManager<IdentityRole
26          {
```

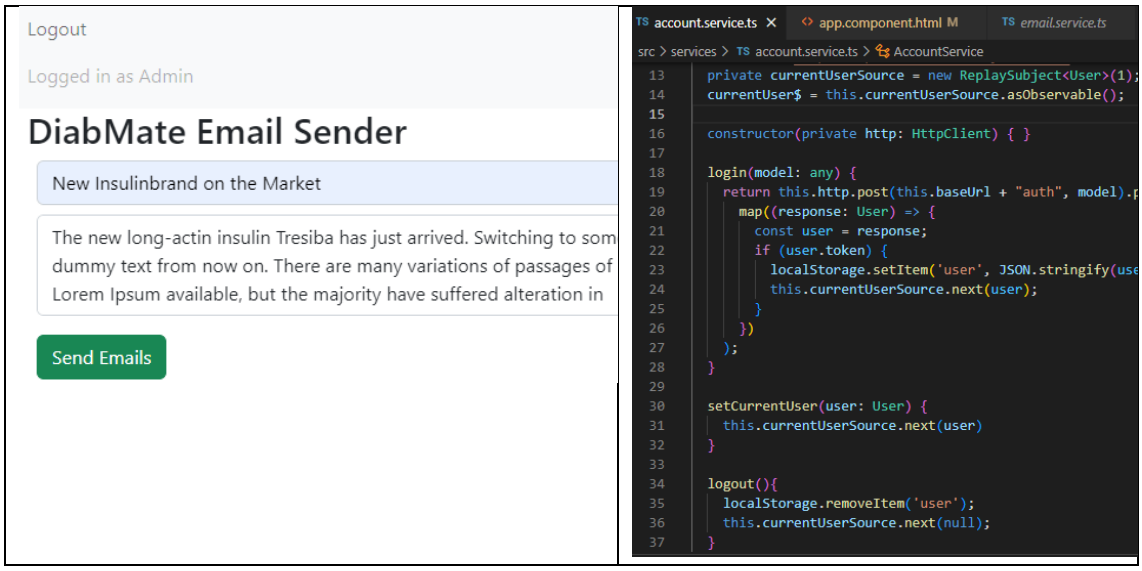
28. ábra: AdminController kontroller részlet

Swagger UI-t használva a böngészőben nem csak láthatjuk, de tesztelhetjük is az API végpontokat. Roppant hasznos, ha nem akarunk más „third-party” appot használni az endpoint-ok tesztelésére, mint a Postman vagy az Insomnia.

29. ábra: Swagger UI - Api végpontok

7.1.3. Email Service

A végpontok elérése mellett lehetőség van emailek küldésére is egy Angular-os webes felületről. Bejelentkezve lehetőség van emailt küldeni a rendszerben szereplő felhasználók által megadott email címekre. Egy egyszerű felületen tudjuk megadni a tárgyat és a levél tartalmát. Küldés után a felhasználók postaládájába érkezik a levél.



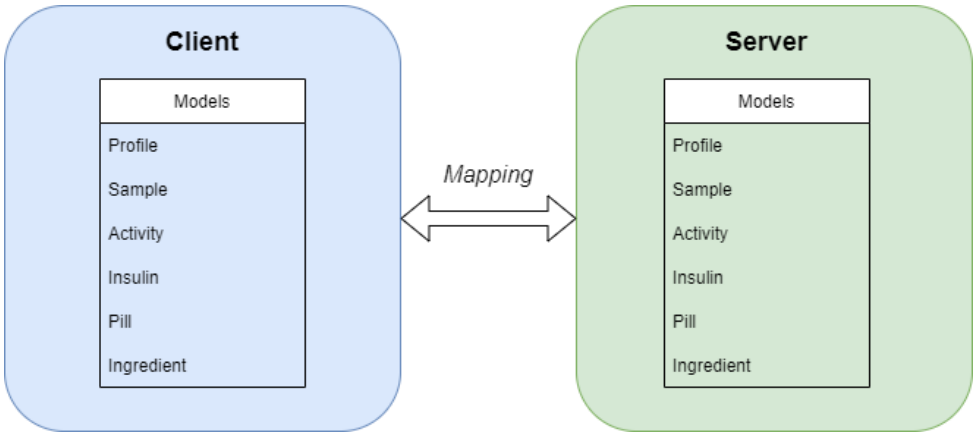
30. ábra: MailService Angular frontend UI és Service

7.2. Kliens

Az MVVM mintát használó kliensen keresztül a felhasználó interakcióba léphet a szerverrel, de használhatja az alkalmazást offline módban is. Fontos szempont ez az alkalmazás fejlesztésénél, hiszen nem minden esetben áll módunk csatlakozni az internethez. Mivel nyilvánvalóan a telefon is rendelkezik saját háttértárral, ezért az internet kapcsolat csak arra hivatott, hogy az adatokat szinkronizálni tudjuk az adatbázisban eltárolt adatokkal.

7.2.1. Model

Elsőként a szerveren eltárolt adatok, osztályok mappelését kell megoldani, mivel a frontenden is meg szeretnénk jeleníteni ezeket az adatokat. Ehhez persze egy osztálynak nem minden adatára van szükség, hiszen nem feltétlenül az összes tárolt adatot szeretnénk megjeleníteni a felhasználó számára.

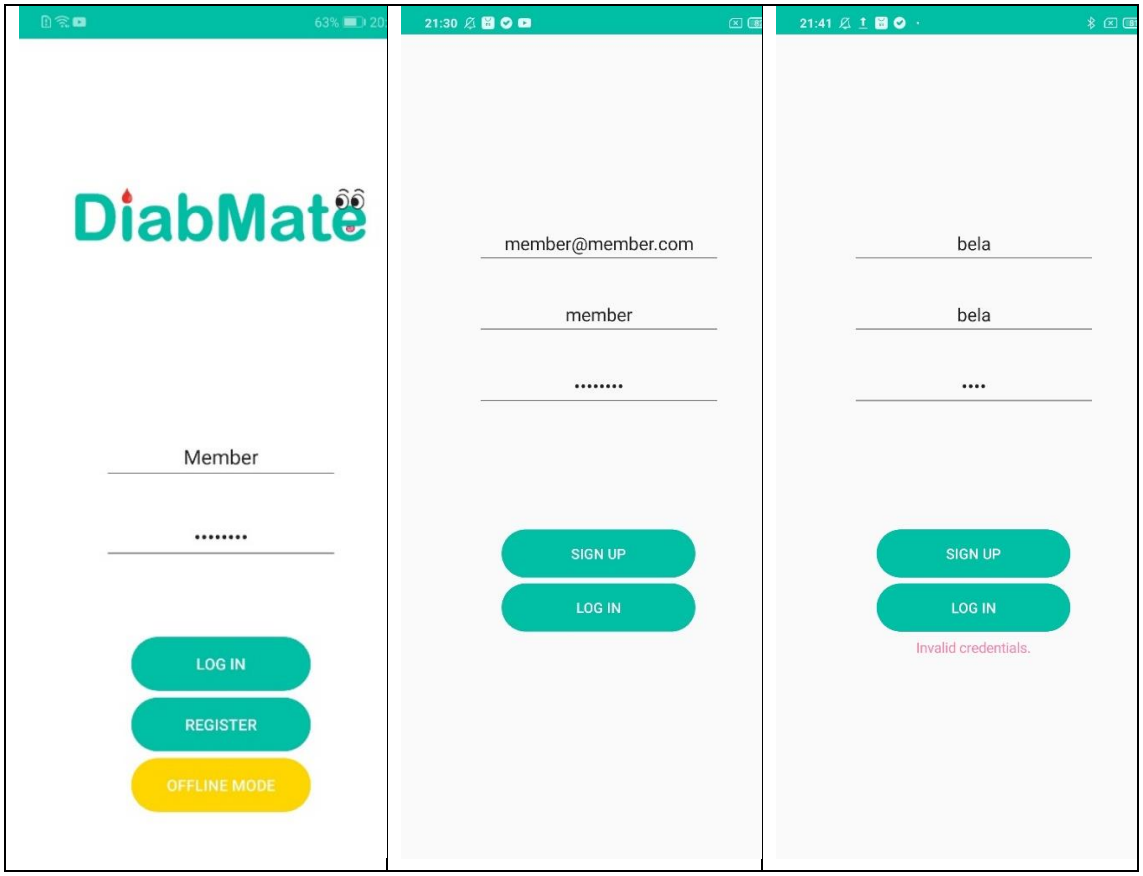


31. ábra: Model osztály mappelés

7.2.2. View

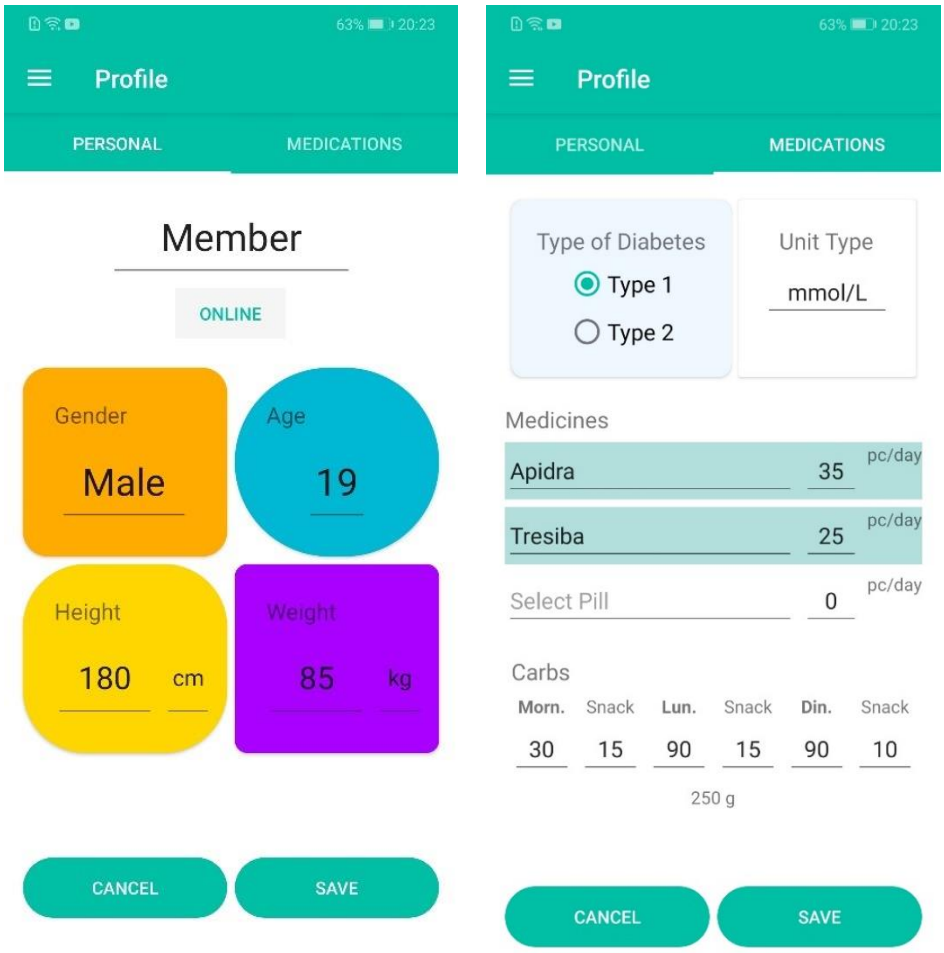
Xamarin.Forms-ban az oldalak megjelenítéséért .xaml és .xaml.cs fájlok felelnek. A VM-ben szereplő mezők két irányú adatkötéssel jelennek meg az egyes oldalakon, így a View képes frissíteni a VM-et és fordítva. A View-k egy Shell alá tartoznak, amely egyfajta keretként szolgál az alkalmazás megjelenítésére. Alapvető elemeket tartalmaz, mint például a URI alapú navigáció a View-k között és kereső az egyes oldalakon. Ide tartoznak még a „Flyout" és „Tab" UI komponensek, amelyek a tipikus mai mobilalkalmazások kinézetét hivatalosak implementálni.

Az alkalmazás indítását követően megjelenik a bejelentkező felület, háttérben az alkalmazás logójával. A „Login Page"-ről lehetőség van a regisztrációs oldalra navigálni, vagy offline módban bejelentkezni, hisz egyeltalán nem biztos, hogy internethozzáféréssel szeretnénk használni az alkalmazásunkat. Sikertelen regisztráció esetén hibaüzenetet kapunk egy „Invalid credentials" üzenettel, sikeres művelet esetén az alkalmazás visszaléptet minket a bejelentkező felületre.



32. ábra: Bejelentkezés és Regisztráció lapok

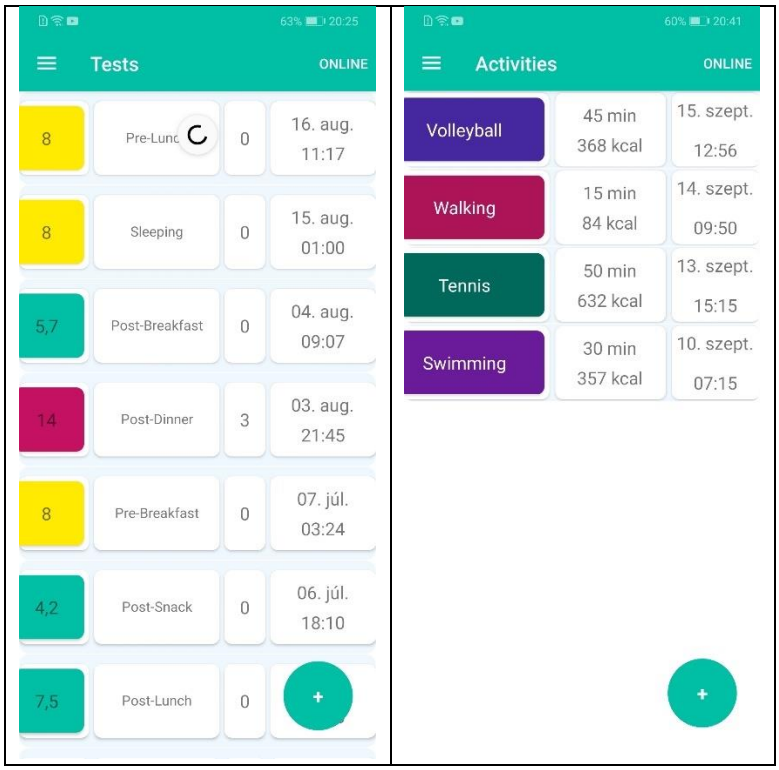
Az alkalmazást megnyitva a sikeres authentikációt követően a profil oldalra jutunk, ahol a fontosabb személyes adatainkat látjuk. Itt különböző színű formákat használtam az egyes beviteli mezők mögött, amely láttán nem tűnik unalmasnak a nyitó felület. Jobbra görgetve a „Medications” lapra érünk, ahol a betegséggel kapcsolatos adatainkat rögzíthetjük, köztük a napi szénhidrát adagjainkat is, amely alatt egy folyamatosan frissülő összeadó van. A gyógyszer listákból kedvünkre választhatunk az inzulinok / tabletták közül, megadva a napi adagot. Alul található a két legfontosabb gomb: „Mentés” és „Mégse” felirat jelzi a céljukat. Utóbbira nyomva a begépelt adatok törlődnek és a beviteli mezők visszaálnak a legutóbb mentett állapotukra.



33. ábra: Profil oldal Személyes és Gyógyszerek lap

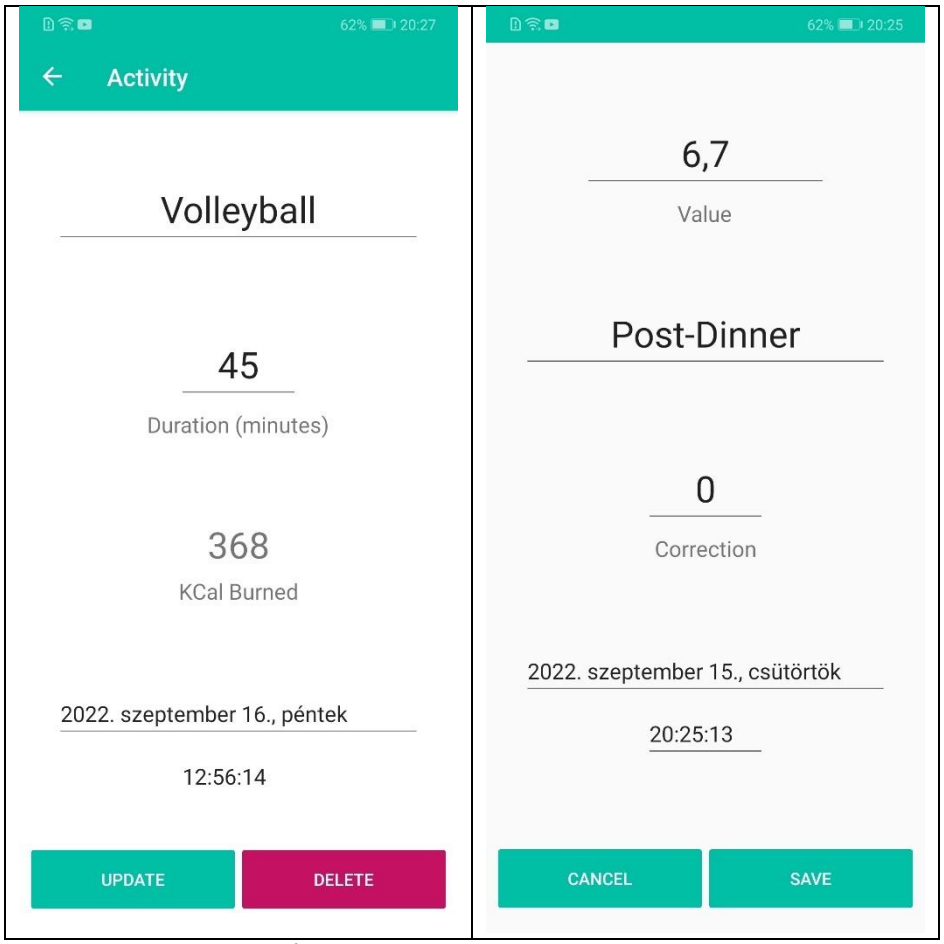
A naplózást két menüpont alatt tehetjük meg, mivel vércukrot és sporttevékenységet is tudunk naplózni. Az alapműködés a két esetben ugyanaz. Lehet listázni a már felvitt értékeket, illetve lehet hozzáadni, szerkeszteni és törölni már meglévő adatot. A lista felett lebegő „+” ikonra kattintva tudunk új mérést/aktivitást hozzáadni. A vércukorértékek függvényében különböző színek jelzik az értékek jóságát. Mérési eredmény felvitele esetén megadhatjuk a mérés idejét, értékét, az alkalmazott korrekció mennyiségét és hogy mely étkezés előtt vagy után mértünk. Aktivitás

szempontjából a következőkre van lehetőség: sportnem, időtartam, és időpont felvitele. Emellett megjelenik az elégetett kalória, mely kiszámításához a program a felhasználó súlyát veszi figyelembe. Ha elírtuk az adatot van lehetőségünk szerkeszteni vagy törölni azt. Mindkét esetben a listázásnál az összes adat megjelenik, és nem tűnik túl zsúfoltnak az oldal, cserébe informatív a felhasználó számára.



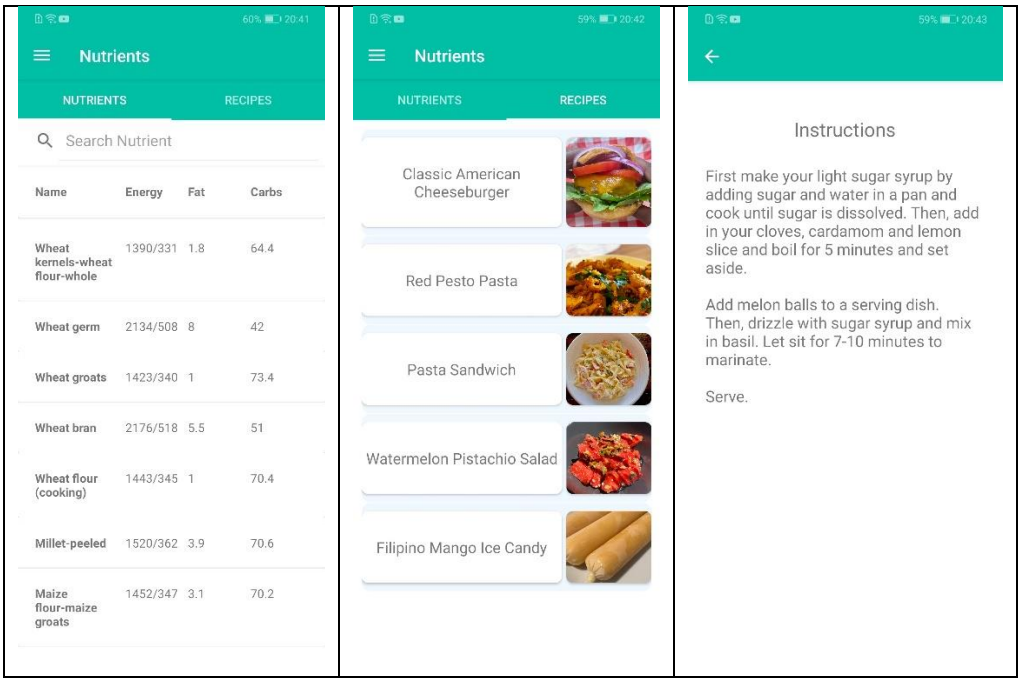
34. ábra: Mérések és Sporttevékenységek oldal

Új elem felvitelénél mindkét esetben kellően nagy beviteli mezők jelennek meg az oldalon ahhoz, hogy gyorsan felvihessük tevékenységünket. A mérési idő automatikusan beáll, de természetesen lehetőség van ezt módosítani a felületen.



35. ábra: Új értékek felvitele az Aktivitás és Mérés oldalon

A „Nutrients” menüpont alatt élelmiszerek / összetevők tápanyagtartalmát találjuk, valamint recepteket. Az első lap a tápanyagokat tartalmazza, melyek főbb jellemzői annak neve, energia, zsír és szénhidrát mennyiség tartalom egy egységnyi (100 g vagy 100 ml) adagban. A lista – a gyógyszerekhez hasonlóan – az alkalmazás első megnyitását követően töltődik le a készülékre, ezt követően nincs szükség internetre az adatok megjelenítéséhez. Folyamatos gépelés közben a lista minden gomblenyomás után frissül, szem előtt tartva a user experience-t. A receptek lapon 5 random recept jelenik meg. Ezek a rapidapi.com egy nyilvános api-járól származnak, melyre lehetőség van szűrést is beállítani, hogy az alkalmazás például bizonyos szénhidrát mennyiséget tartalmazó ételeket kínáljon fel. A receptekre nyomva megjelenik annak alkészítési módja.



36. ábra: Tápanyagok oldal, Receptek lista és recept leírás

A „Statistics” oldal egy másik főbb funkciója az alkalmazásnak, mely megjeleníti az eszközön eltárolt adatokból generált statisztikákat, melyek diagramokon jelennek meg. A diagramok kirajzolásához a *microcharts* könyvtárat használtam, mivel ingyenes és viszonylag komplex ábrák kirajzolására van lehetőség. Ennek segítségével többféle chart közül választhatunk: vonaldiagram, pontdiagram, oszlopdiagram stb. Az első oldalon a mérésekkel kapcsolatos statisztikákat láthatunk. Checkbox-okkal – egyszerre többet is megjelölve – kiválaszthatjuk, hogy mely adatokra vagyunk kíváncsiak. Pl.: reggeli előtti és reggeli utáni vércukorértékeket szeretnénk látni. Itt van lehetőség a periódus megválasztására is, mely heti, vagy havi adatok megjelenítésére szolgál. Az opciók alatt kirajzolódik a diagram, mely különböző színnel jeleníti meg a kiválasztott mérések eredményeit. Ennek implementálására egy MultiLinesChart nevű osztályt használok, mely egy LineChart-ból származik le, némi módosítással, hogy egyszerre több adathalmaz értékeit jeleníthessük meg a kijelzőn. A ViewModel-ből elég csak a MultiLinesChartBuilder osztályt metódusait *fluent api* módszerrel hívni, mely tetszőleges adattal felépíti a kívánt diagramot. Alatta egy mérési átlagokat mutató diagram van, mely a heti, havi és háromhavi átlag vércukrot mutatja. Az oszlopok különböző színei indikálják, hogy a jelzett érték jó, vagy hosszú távon káros az egészségre. Végül pedig egy diagram arra, lássuk mennyi mérést történt egy hét alatta az adott napszakra. Az aktivitás lapon is szerepel egy diagram, amely egyszerre jeleníti meg, hogy mennyi kalóriát égettünk el egy nap, és ennek függvényében hogyan alakult aznap az átlag vércukorszint.

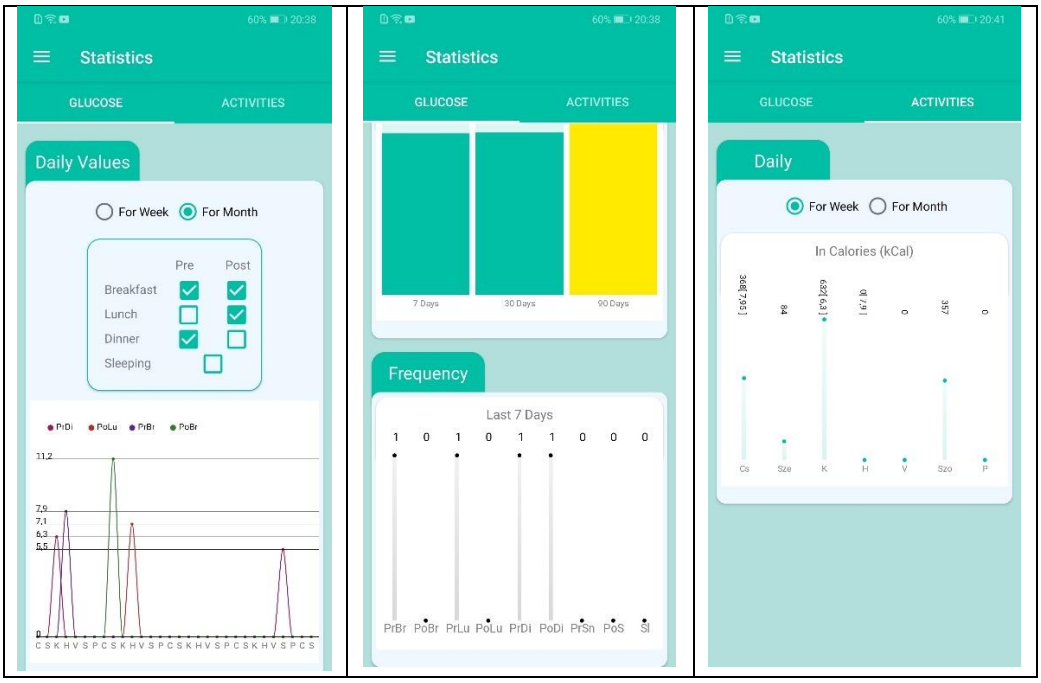
```
namespace DiabetesApplication.Helpers
{
    public class MultiLinesChart : LineChart
    {
        public IEnumerable<IEnumerable<ChartEntry>> Mult
        public List<string> LegendNames { get; set; }
        private bool initiated = false;
        private float multiLineMax;
        private float multiLineMin;
        private List<SKPoint> points = new List<SKPoint>

        private void Init()
        {
            if (initiated)
            {
                return;
            }

            initiated = true;
            foreach (List<ChartEntry> line in MultiLineE
            {
                foreach (ChartEntry entry in line)
                {
                    if (entry.Value > multiLineMax)
                    {
```

```
        public class MultiLinesChartBuilder
        {
            private List<string> legends;
            private List<float[]> entries;
            private string[] x;
            private float labelTextSize = 30f;
            private Orientation labelOrientation = Orientati
            private bool isAnimated = false;

            public MultiLinesChart Build()
            {

                if (IsValid())
                {
                    List<List<ChartEntry>> chartEntries = ne

                    int yCount = 0;
                    foreach (var element in entries)
                    {
                        int idx = 0;
                        List<ChartEntry> entry = new List<Ch
                        foreach (var data in element)
                        {
                            entry.Add(new ChartEntry(data)
                            {
                                Color = Constants.SKColors.E
                                ValueLabel = $"{data}",
                                Label = x[idx]
```

37. ábra: MultiLinesChart és MultiLinesChartBuilder osztályok

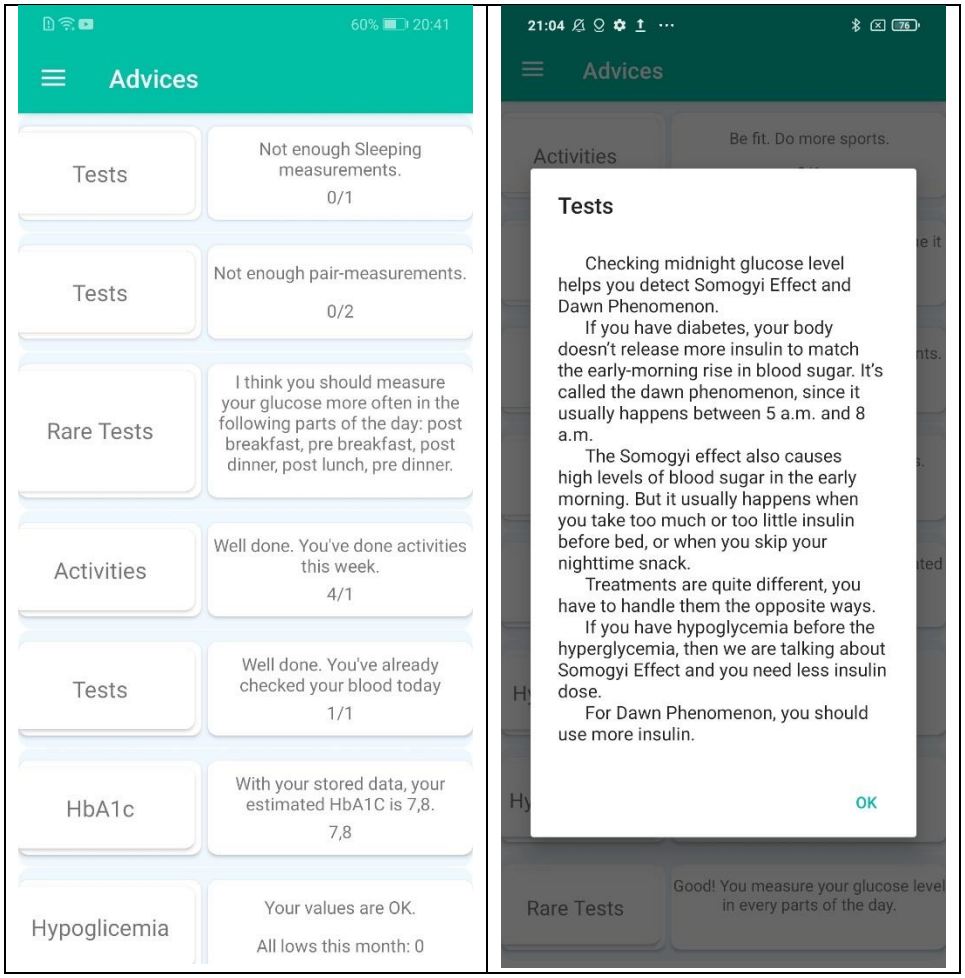
```
private Task BuildMultiLineGlucoseChart()
{
    MultiLinesChart = new MultiLinesChart.MultiLinesChartBuilder()
        .SetEntries(Y)
        .SetX(DaysGChart)
        .SetLegends(Legends)
        .Build();

    return Task.CompletedTask;
}
```

38. ábra: MultiLinesChartBuilder hívás a VM-ből



39. ábra: Statisztikák oldal

Végére maradt az „Advices” oldal, amely a tanácsok megjelenítéséért felel. Itt olyan értesítésekkel szolgál a program, melyet az irodalomkutatásban is taglaltam. Igyekeztem több az egészséget befolyásoló faktort találni, melyek karbantartásával a beteg állapota kiegyensúlyozott és színvonalas maradhat. A tanácsok a listában aszerint vannak rendezve, hogy a felhasználó eleget tesz-e bizonyos feltételeknek. Ebből kifolyólag a lista elején szerepelnek azok az elemek, amelyeknek a beteg nem tesz eleget (keveset sportol, nincs elég párban mérés, elhanyagolja az éjszakai vércukor mérést). Ide tartozik a hajnali- és somogyi effektus is, amik kevésbé észrevehetően befolyásolják a vércukrot. A listaelemekre koppintva felugró ablakban egy súgó jelenik meg, hogy az adott tanács miért fontos egy diabéteszes számára.



40. ábra: Tanácsadás oldal és tanács részletei

7.2.3. ViewModel

Itt szerepelnek azok az adatmezők, amelyeket a felhasználó a UI (View) használatával interakcióba lép. Minden mező írásakor lefut az OnPropertyChanged() esemény, amely kivált egyfajta viselkedést az interakciókra. Így oldottam meg például azt is, hogy új aktivitás felvitelekor az „időtartam” változtatásakor a rendszer – figyelembevéve a páciens súlyát, valamint egy service segítségével – kiszámolja és

megjeleníti az elégetett kalóriát. Itt, a View Modelben jönnek létre az egyes diagramok is, melyek egy külön statisztikákra dedikált servicet használnak az adatok megjelenítésére.

Ezekben az osztályokban a service-k Dependency Injection használatával kerülnek be a konstruktorba. Így időt és erőfrrást spórolunk meg, hisz nem kell mindig minden egyes servicet újra meg újra konstruálni. Két konstruktort kell létrehozni minden VM-ben, mivel alapértelmezetten a View-k a VM-ek üres konstruktorát keresik. Ezért a ServiceLocator-ral példányosított paramétereket tartalmazó konstruktort az üresből hívom meg. A locator megkeresi nekünk a servicek példányát és injektálja a konstruktorba.

```
namespace DiabetesApplication.ViewModels
{
    public class SamplesViewModel : BaseViewModel
    {
        private IApiService apiService;
        private IFetchDataService fetchService;
        private ISampleDataService sampleService;


        public SamplesViewModel(IApiService apiService, IFetchDataSer
        {
            this.apiService = apiService;
            this.fetchService = fetchService;
            this.sampleService = sampleService;
        }
         public SamplesViewModel() : this(
             ServiceLocator.Current.GetInstance<IApiService>(),
             ServiceLocator.Current.GetInstance<IFetchDataService>(),
             ServiceLocator.Current.GetInstance<ISampleDataService>()
             )
         {
         }
```

41. ábra: ViewModel

7.2.4. Services

Ez a réteg alkotja a mobilalkalmazás üzleti logikáját. Itt végződnek a számítások és ez az a réteg, ami a szerverrel kommunikál. Lényegében négy féle servicet hozok létre. Kell a tárolásra egy DataService, egy az adatok szinkronizálásáért felelő FetchService, üzleti logikáért felelős servicek, melyek a StatisticsService és AdviceService valamint a kliens-szerver közti kommunikációért felelős ApiService. Mivel az alkalmazást lehetőség van offline módon is használni, ezért a DataService a helyi adattárolást SQLite adatbázis segítségével valósítja meg. Ezt a service-t a FetchService is használja, ami – figyelve az offline/online kapcsolót – online módra váltás esetén az adatokat először felküldi az szerver adatbázisába, majd a távoli adatokat lekéri a szervertől, így megkapva a felhasználó összes adatát tud a páciens naprakész maradni. Az kliens a szerverrel az ApiService-en keresztül kommunikál JWT használatával. Minden egyes kérés mellé kerül a token is, ezáltal fogja tudni a szerver-oldali authorizációért és authentikációért

felelős middleware megmondani, hogy jogosultak vagyunk-e a végpont elérésére vagy sem. A két business logikáért felelős szervice szintén az eszközön eltárolt adatokból táplálkozik. A statisztikáért felelős osztály metódusai összegyűjtik az adatokat, majd továbbadják a ViewModel számára, illetve a tanácsadó komponens számára, aki a kapott adatokból összeállítja a listát, amit az AdvicesViewModel kap meg és biztosítja az elérést az ezt megjelenítő oldal számára.

8. TESZTELÉS

8.1. Módszertan

Az alkalmazás megfelelő működésének teszteléséhez egység tesztelést választottam. Unit tesztek írásához több keretrendszer is a rendelkezésünkre áll, ilyenek az *MSTest*, az *NUnit* és *XUnit*. Mindhárom open-source, és mindegyikkel lehetőség van a tesztek párhuzamos futtatására. A választásom az XUnit-ra esett, mivel a kezdő adatok inicializálásához elég a konstruktort használni, mivel a framework minden tesztesethez új példányt készít. Valamint lehetőség van a tesztek kontextusát megosztani tesztek között úgynevezett *Class Fixture*-ök segítségével, melyhez a hivatalos dokumentációban részletes leírást is találni. A nevezéktannal kapcsolatban sokféle gyakorlat létezik. A projektemben a *Given-When-Then* sablont használtam a tesztmetódusok elnevezésére, amely jól leírja, hogy adott körülmények között bizonyos műveletek végrehajtása után milyen eredményt várunk. Például: a „Given valid credentials When login button pressed Then user is authenticated” tisztán leírja, hogy egy teszteset leírásánál mit várunk el a programtól.

```
1 reference | 0 changes | 0 authors, 0 changes
public class SampleControllerTest
{
    private Mock<ISampleRepository> _sampleRepository;
    private SampleController ?_controller;

    0 references | 0 changes | 0 authors, 0 changes
    public SampleControllerTest() => _sampleRepository = new Mock<ISampleRepository>();

    [Fact]
    | 0 references | 0 changes | 0 authors, 0 changes
    public void Given_Authentication_When_TryToGetAllSamples_Then_SamplesReturn()
    {
        var user = new IdentityUser("user");
        var id = user.Id;

        // Arrange
        var userManagerMock = GetUserManagerMock<IdentityUser>();
        userManagerMock.Setup(u => u.FindByIdAsync(It.IsAny<String>())).Returns(Task.FromResult(user));
        var roleManagerMock = GetRoleManagerMock<IdentityRole>().Object;
        List<Sample> samples = new List<Sample>()
        {
            new Sample()
            {
                Id="some_id",
                CorrectionAmmount=4,
                MeasureType="Post-Dinner",
                OwnerId = id,
                Synced=false,
                Timestamp=DateTime.Now,
                Value=6
            }
        };
        _sampleRepository.Setup(repo => repo.GetAll()).Returns(samples.AsQueryable());
        _controller = new SampleController(userManagerMock.Object, _sampleRepository.Object, null, null);
```

42. ábra: Unit test XUnit Framework használatával

8.2. Teszt eredmények

Az alkalmazás a fejlesztés végére minden teszten átment. A tesztesetek külön szerver-oldali és kliens-oldali esetekre bontásával az alábbi eredményeket kaptam a tesztelés során.

8.2.1. Szerver-oldali tesztek

SERVER-SIDE TESTS		
Category	Name	Passed
Controller	Given_NoAuthorization_When_TryToGetAllSamples_Then_UnauthorizedMessageReturns	True
	Given_CredentialsThatAlreadyExist_When_Register_Then_InvalidCredentialsMessageReturns	True
	Given_InvalidCredentials_When_Login_Then_UserDoesntExistMessageReturns	True
	Given_ValidCredentials_When_Login_Then_AccessTokenInResponseMessageReturns	True
	Given_ValidCredentials_When_Register_Then_UserStoredInDatabase	True
	Given_Authorization_When_TryToAddSample_Then_NewRecordAddedToDatabase	True
	Given_Authorization_When_TryToGetAllSamples_Then_SamplesReturn	True

Repository	When_TryToAddActivity_Then_NewRecordAddedToDatabase	True
	Given_ActivityID_When_TryToGetActivity_Then_ActivityReturns	True
	When_TryToGetAllActivites_Then_AllActivitiesOfUserReturns	True
	Given_ActivityID_When_TryToUpdateActivity_Then_RecordUpdatedInDatabase	True
	Given_ActivityID_When_TryToDeleteActivity_Then_RecordRemovedFromDatabase	True
	When_TryToAddSample_Then_NewRecordAddedToDatabase	True
	Given_SampleID_When_TryToGetSample_Then_SampleReturns	True
	When_TryToGetAllSampless_Then_AllSamplesOfUserReturns	True
	Given_SampleID_When_TryToUpdateSample_Then_RecordUpdatedInDatabase	True
	Given_SampleID_When_TryToDeleteSample_Then_RecordRemovedFromDatabase	True
	Given_InsulinID_When_TryToGetInsulin_Then_InsulinReturns	True
	When_TryToGetAllInsulins_Then_AllInsulinsOfUserReturns	True
	Given_PillID_When_TryToGetPill_Then_PillReturns	True
	When_TryToGetAllPills_Then_AllPillsOfUserReturns	True
	When_TryToGetAllIngredients_Then_AllIngredientsReturns	True
	Given_ProfileID_When_TryToGetProfile_Then_ProfileReturns	True
	Given_ProfileID_When_TryToUpdateProfile_Then_RecordUpdatedInDatabase	True
	When_TryToAddProfile_Then_RecordAddedToDatabase	True
Email Service	When_TryToSendEmails_Then_EmailsSentToUsers	True
	When_EmailsSent_Then_BodyAppearsInEmail	True
	When_EmailsSent_Then_SubjectAppearsInEmail	True

8.2.2. Kliens-oldali tesztek

CLIENT-SIDE TESTS		
Category	Name	Passed
Login / Register	Given_ValidCredentials_When_LoginButtonPressed_Then_LoadingSymbolAppears	True
	When_Authenticated_Then_RedirectedToProfilePage	True
	When_OfflineModePressed_Then_RedirectedToProfilePage_And_OfflineModeIsActive	True
Profile Page	Given_NewUser_When_ProfilePageOpened_Then_AllFieldsAreEmpty	True
	Given_NewUserAfterOldUser_When_ProfilePageOpened_Then_AllFieldsAreEmpty	True
	When_SaveButtonPressed_Then_ProfileSavedToDatabaseMessageAppears	True
	When_CancelButtonPressed_Then_AllFieldsSetToPreviousState	True
	Given_SavedUserToDatabase_When_ProfilePageOpened_Then_UsersDataFetchedFromDatabaseAppear	True
	When_CarbohydratesSet_Then_SumOfCarbohydratesRefreshes	True
Samples Page	When_NavigatingToSamplesPage_Then_AllSamplesAppear	True
	Given_OfflineMode_When_NavigatingToSamplesPage_Then_AllStoredSamplesAppear	True
	When_NewSampleAdded_Then_NewSampleAppearsInSamplesList	True
New Sample Page	When_AddButtonPressed_AndNavigatedBackToSamplesPage_Then_NewSampleAppearsInSamplesList	True
	When_CancelButtonPressed_Then_RedirectedToSamplesPage	True
	When_TappedOnValueEntry_And_InappropriateCharacterTyped_Then_CharacterDoesntAppear	True
Activities Page	When_NavigatingToActivitiesPage_Then_AllActivitiesAppear	True
	Given_OfflineMode_When_NavigatingToActivitiesPage_Then_AllStoredActivitiesAppear	True
	When_NewActivityAdded_Then_NewActivityAppearsInActivitiesList	True
New Activity Page	When_AddButtonPressed_AndNavigatedBackToActivitiesPage_Then_NewActivityAppearsInActivitiesList	True
	When_CancelButtonPressed_Then_RedirectedToActivitiesPage	True
	When_TypingInTimeEntry_Then_CaloriesAutomaticallyCalculated	True
Statistics Page	When_NewSampleAdded_And_StatisticsPageOpened_Then_NewStatiscticsCalculated	True
	When_HighValuesCalculated_Then_ColorOfUIComponentsChange	True
	When_MultipleCheckboxesTicked_Then_MultipleDataAppear	True
Advices Page	When_NewNewSampleAdded_And_AdvicesPageOpened_Then_NewStatisticsAdvicesGenerated	True

Nutrients Page	When_TypingInSearchEntry_Then_FilteredListDisplayed	True
	When_RecipesTabOpened_Then_RecipesDisplayed	True

9. ÉRTÉKELÉS

9.1. Konklúzió

Az általam készített alkalmazást méréssel értékelni ugyan nem lehet – hiszen az alkalmazás célja nem a gyorsaság volt –, viszont funkcionalitási szempontból már igen. Mivel a cél az volt, hogy a dolgozatom elején említett alkalmazásokból merítve – azok ötvözeteként – létrehozzak egy mobilappot, ezért sorra végig véve a use-case-eket lenne érdemes megvizsgálni, hogy mi az, ami sikerült, vagy mi, ami kevésbé. Összevetve a saját eredményeimet a többi gyártó termékével fog kiderülni, hogyan sikerült az alkalmazás.

A naplózás volt talán a rendszer legfontosabb alapeleme, hiszen enélkül nem tudunk statisztikákat generálni vagy tanácsadást végezni. A felület eléggé letisztultra sikeredett, de színek segítik vizualizálni az adatokat a mérések historikus megjelenítésénél. Mivel a navigálás egyszerű az oldalakon és nincsenek eldugva a fontosabb funkciók különböző menüpontok alá, így gyorsan el tudunk jutni a vércukornaplózóra vagy bármely más oldalra. A diagramok átláthatóra és informatívra sikerültek, ezek mellett fontos szempont volt, hogy egyszerre minél több adatot tudjon megjeleníteni, hiszen összevetve egy adathalmazt egy másikkal – amely hatással lehet az előzőre – van értelme igazán ábrákat megjeleníteni a felhasználónak, amiből következtetéseket tud levonni, és melyekből tanulhat. A receptek megjelenítése szintén egy jó funkció véleményem szerint, hiszen cukorbetegként gyakran szembe találkozhat az ember a kérdéssel, hogy „Milyen cukorban/lisztben szegényebb ételt egyek a mai nap, amit a héten még nem ettem?”, és ezzel akár ötletet is meríthet magának.

9.2. Nehézségek

A szakdolgozatom írása közben több nehézség is akadt. Ilyen volt az a probléma, hogy meg kellett oldani, hogy a kliens (telefon) és a szerver (laptop) valóban tudjon egymással kommunikálni, hiszen a tanúsítványok kezelése fejfájással tud járni. Végül annál a megoldásnál maradtam, hogy egy Conveyor nevű .NET-es pluginnal hidalom át a problémát, mivel ez távoli elérést biztosít a saját, lokális IIS Express webszerverhez. Ezzel lényegében egy publikus url-ről el tudjuk érni a lokálisan futó ASP.NET API-unkat, mellyel megspóroltam a certificate-ek okozta fejfájást. Nehézség volt továbbá az is, hogy miképpen fogok tudni megjeleníteni több grafikát egyetlen diagramon, de hála a micocharts könyvtárnak, ezt sikerült megoldanom. Nehézség volt továbbá, hogy hogyan lehetne minél személyre szabottabb tanácsokkal ellátni a felhasználót, és erre úgy gondolom, több nyilvánosság számára nem, vagy csak nehezen elérhető kutatásokra lett

volna szükségem ahhoz, hogy ezt egy még felsőbb szinten implementálhassam a programban.

Ezek ellenére sikerült a specifikációban megfogalmazott feladatokat az elvártaknak megfelelően teljesítenem és úgy vélem, hasznos alkalmazás lett ez egy cukorbeteg ember számára.

10. TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK

Az alkalmazás továbbfejleszthetősége lehetőséget nyújt arra, hogy szélesebb célközönség számára legyen használható egy alkalmazás. Több funkció is van, amellyel szeretném a jövőben kibővíteni a programot. Fontos lenne, ha az alkalmazást nem csak a páciensek, de maguk az orvosok is használhatnák és ezáltal egy olyan platform jönne létre, mely leegyszerűsíti az orvos-beteg kommunikációt. Dokumentumkezeléssel is bővülhetne az alkalmazás, így diabetológus által felírt javaslatok is felkerülhetnének ide. Az orvosi papírok megőrzése nehéz feladat szokott lenni, így viszont ezen sem kellene többet aggódni.

Továbbá hasznos lenne, ha a beteg által jegyzett adatokat exportálni / importálni lehetne az alkalmazásba. Bluetooth-os vércukormérő híján nem tudtam kipróbálni milyen lenne, ha a gépről az adatokat átlehetne küldeni a telefonra, de úgy vélem sajnos ez egy túlságosan is zárt rendszer ahhoz, hogy ki tudjam próbálni, ha szert tennék egy ilyen eszközre. Azonban, a kiexportált adatokat lehetőségünk lenne kontrollra magunkkal vinni.

Jelenleg az alkalmazás nem képes push értesítések küldésére, viszont hasznos lenne, ha különböző napszakokban értesítések érkeznének arról, hogyan tartsuk karban vércukrunkat. Ezeket egy beállítás menüben lehetne konfigurálni kedvünk szerint.

Továbbfejlesztési lehetőség lenne, hogy a felhasználó szerveren tárolt adatait összevetve más felhasználók adataival tudjunk következtetéseket vonni. Megvizsgálva más páciensek adatait tanácsadásra is lehetne ezt használni. Esetleg elérhetővé lehetne tenni mások mérési / sportolási adatait oly módon, hogy lássuk, mennyire vagyunk „jó cukorbetegek”. Úgy gondolom ez jó hatással lenne a betegséghez való hozzáállásra.

11. ÖSSZEFOGLALÁS

Szakdolgozatom témája egy olyan Androidos alkalmazás készítése, mely a piacon levő hasonló szoftverekből ötletet merítve hasznos funkciókkal látja el felhasználóját, és segítségére lesz a mindennapokban azáltal, hogy értékes statisztikát nyújt számára és személyreszabott tanácsokkal látja el őt. Tekintve, hogy a világon több, mint félmilliárd cukorbeteg él, valamint az informatikai technológiai háttér fejlődése egyre nagyobb ütemben történik, vált számomra egy törekvéssé egy hasznos mobilalkalmazás fejlesztése. A dolgozatom elején ismertetem magát a diabétesz jelenségét a világon, és

hogy miért fontos egy támogató alkalmazás fejlesztése. Szó van a mobilalkalmazásokról és hogy hasonló témában elég sok szoftver létezik, melyek többnyire ugyan hasonlítanak, mégis vannak funkciók, amikben eltérnek egymástól. Én ezeket a funkciókat keresem és igyekszem felhasználni az alkalmazásomban, hogy lehetőleg minél több funkciót valósíthassak meg, és ne csak egy sablonos mobilalkalmazás legyen a végeredmény. Fontos megemlíteni a már használt módszereket, hogy milyen architektúrális, illetve adatbázis megvalósításokat használnak a ma is működő alkalmazásokban. Emellett nem szabad kihagyni a tanácsadásra való piaci példákat sem, mely a készülő app fontos feladata lesz. Ezt követően specifikálom az alkalmazás elvégzendő feladatait nagy vonalakban, a használati eseteket, melyeket felépítését UML diagramok segítségével ábrázolom. A rendszertervben a szerver-kliens architektúra főbb részei és azok működése kerül bemutatásra. A kettő közti kapcsolatért pedig egy szerveroldali API a felelős, mely a publikus elérésű lokális végpontokon keresztül teszi elérhetővé a kéréseket fogadó kontrollereket a kliens számára. Bár a kliensen helyben eltároljuk az adatokat, fontos, hogy a felhasználók adatai biztonságban, a szerveren is el legyenek tárolva egy adatbázisban. Ez biztosítja az adatok elérését több készülékről is. A megvalósítás részben kifejtésre kerül, hogy mely technológiák mellett döntöttem a tervezés során. Az alkalmazás backend-jéül szolgáló ASP.NET Core alkalmazás Model-View-Controller tervezési mintán alapul, a kliensoldali Xamarin.Forms applikációé pedig Model-View-ViewModel mintára épül. Az authentikációt és authorizációt a keretrendszer ASP.NET Core Identity végzi el. Továbbá, implementálásra kerül egy Mail Service is, melyhez a frontendet Angular Framework használatával TypeScript nyelven írom. implementációban már a konkrét megoldás szerepel, milyen technikát, megközelítést használtam az egyes feladatok elvégzésére. A programkódot a szerveroldalra, valamint az androidos alkalmazásra Visual Studio 2022 segítségével .NET 6 keretrendszerrel, az Angular frontendet Visual Studio Code IDE-ben, 16-os Node verzióval végeztem. A Xamarin.Forms sok hasznos könyvtárat tartalmaz, melynek végeredményét a telefonnal készült képernyőképekkel igyekeztem demonstrálni. A teszteléshez segítségemre volt a .NET XUnit framework-je, valamint a végpontok teszteléséhez Swagger UI-t használtam. Igyekeztem minél több „edge-case” szituációt is letesztelni, hogy minél stabilabban fusson a program. Végeredményben sikerült a feladatban kiírt és a specifikációban definiált alkalmazást megtervezni és megvalósítani, melynek köszönhetően létrejött az alkalmazás, mely segíti egyszerűbbé, kényelmesebbé és tudatosabbá tenni a diabéteszesen hétköznapi életét.

12. SUMMARY

The subject of my thesis is to build an Android application that provides its users useful functions by gathering ideas from similar software on the market, and it will help diabetics in their everyday life by providing them valuable statistics and personal advice. Given the fact that there are more than half a billion diabetics in the world and the

improvement of the IT sector, the development of a useful mobile application has become an aspiration for me. At the beginning of my thesis, I describe the phenomenon of diabetes in the world and why it is important to develop a supportive application. In this section I say some words about mobile applications and that there are quite a lot of software on a similar topic which are mostly similar but there are some functions in which they differ from each other. I search for these functions and try to use them in my application so that I can implement as many functions as possible and build an application that is not just a mobile application template. It is important to mention the methods that are already used on the market, and what architectural and database implementations are used in the applications that are still working today. In addition, we should not just walk by today's guidance methods in informatics, since it will be an important task of the upcoming app. After that, I specify the tasks to be performed by the application in broader terms, the use-cases which I presented with the help of UML diagrams. The main parts of the server-client architecture and their operation are presented in the system plan. A server-side API is responsible for the communication between the two, which makes the controllers – that receive requests – available to the client through publicly accessible local endpoints. Although we store the data locally on the client, it is important that the users' data is stored in a secured database on the server as well. This ensures data access from multiple devices and persistence. The implementation part is where I explained which technologies I chose during the design phase. The ASP.NET Core application that serves as the backend of the application is based on Model-View-Controller design pattern, and the client-side Xamarin.Forms application uses Model-View-ViewModel design pattern. Authentication and authorization are performed by the ASP.NET Core Identity framework. Furthermore, a Mail Service is implemented, for which I write the frontend in TypeScript language using the Angular Framework. The implementation already includes the exact solution, which techniques and approaches I used to perform each task. I wrote the program code for the server side and the Android application using Visual Studio 2022 with the .NET 6 framework, and the Angular frontend in Visual Studio Code IDE with Node version 16. Xamarin.Forms contains many useful libraries which result I tried to demonstrate with the screenshots taken with the phone itself. The .NET XUnit framework helped me with the unit testing, and I used Swagger UI to test the endpoints of the API. I tried to test as many "edge-case" situations as possible so that the program runs as stable as possible. As a result of my thesis, I managed to design and implement the application described in the task and which was defined in the specification. Thanks to the application that has been created, it can help people with the disease make their everyday-life simpler, more comfortable and more conscious.

13. FELHASZNÁLT IRODALOM:

[1] Prof. dr. Jermendy, Gy.: IDF DIABETES ATLAS – 2019 (kilencedik kiadás) A diabetes prevalenciája,
(https://diabet.hu/tarsasag/diabetes/hirek.aspx?&nid=98084&t=1),
utoljára megtekintve: 2022. 12. 10.

[2] American Diabetes Association: Diagnosis and Classification of Diabetes Mellitus,
(https://care.diabetesjournals.org/content/diacare/28/suppl_1/s37.full.pdf),
utoljára megtekintve: 2022. 12. 12.

[3] Ferri, F. F.: Ferri's Clinical Advisor 2022. Fellow of the American College of Physicians, 2021

[4] Cukorbetegközpont: Somogyi hatás, hajnali jelenség - cukorbetegként nem árt ismerni, (https://www.cukorbetegkozpont.hu/hireink/somogyi-hatas-hajnali-jelenseg-cukorbetegkent-nem-art-ismerni),
utoljára megtekintve: 2022. 12. 12.

[5] Callahan, A., Bedosky, L.: 16 Apps for Managing Diabetes: Blood Glucose Trackers, Food and Exercise Logs, and More,
(https://www.everydayhealth.com/hs/type-2-diabetes-care/diabetes-apps/),
utoljára megtekintve: 2022. 01. 19.

[6] forDiabetes, (https://fordiabetes.app),
utoljára megtekintve: 2022. 12. 07.

[7] Diabetes:M, (https://www.diabetes-m.com),
utoljára megtekintve: 2022. 12. 07.

[8] Healthy Diabetes,
(https://play.google.com/store/search?q=healthy%20diabetes&c=apps&hl=en&gl=US),
utoljára megtekintve: 2020. 12. 07.

[9] Glucose Buddy, (https://www.diabetes-m.com),
utoljára megtekintve: 2022. 12. 07.

[10] Peterson, R.: SQL vs NoSQL: What's the Difference Between SQL and NoSQL,
(https://www.guru99.com/sql-vs-nosql.html),
utoljára megtekintve: 2022. 12. 12.

[11] Smallcombe M.: SQL vs NoSQL: 5 Critical Differences,
(https://www.xplenty.com/blog/the-sql-vs-nosql-difference/),
utoljára megtekintve: 2022. 12. 10.

[12] Anderson, B., Nicholson, B.: SQL vs. NoSQL Databases: What's the Difference?,
(https://www.ibm.com/cloud/blog/sql-vs-nosql),
utoljára megtekintve: 2022. 12. 10.

[13] Choudhary, J. C., Genovese, S., Reach, G.:Blood Glucose Pattern Management in Diabetes: Creating Order from Disorder. Journal of Diabetes Science and Technology, Vol. 7, 2013, pp. 1575-1584

[14] Mitra, A.: Diabetes and Stress: A Review. Studies on Ethno-Medicine, Vol. 2, 2008, pp. 131-135

[15] Dr. Porochnavecz, M.: Inzulinok típusai-pro és kontra, (https://www.cukorbetegkozpont.hu/hireink/inzulinok-tipusai-pro-es-kontra), utoljára megtekintve: 2022. 12. 10.

[16] IOSR-JCE: Client-Server Model. Journal of Computer Engineering, Volume 16, Issue 1, 2014, pp. 67-71

[17] Li, Y., Manoharan, S.: A performance comparison of SQL and NoSQL databases, (https://ieeexplore.ieee.org/abstract/document/6625441), utoljára megtekintve: 2022. 12. 12.

[18] Bhatia, R.: NoSQL vs SQL — Which Database Type is Better For Big Data Applications, (https://analyticsindiamag.com/nosql-vs-sql-database-type-better-big-data-applications/), utoljára megtekintve: 2022. 12. 10.

[19] Microsoft: What is ASP.NET?, (https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet), utoljára megtekintve: 2022. 12. 12.

14. MELLÉKLETEK

14.1. Sample Model

```
using ApiBase.Data;
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;

namespace ApiBase.Models
{
   public class Sample : IEntity<Sample>
   {
      [Key]
      public string Id { get; set; }
      public float Value { get; set; }
      public string OwnerId { get; set; }
      public string MeasureType { get; set; }
      public int CorrectionAmmount { get; set; }
      public DateTime Timestamp { get; set; }
      public bool Synced { get; set; }

      internal void CreateUID()
      {
         Id = Guid.NewGuid().ToString();
      }
```

```
        public void CopyFrom(Sample another)
        {
            OwnerId = another.OwnerId;
            MeasureType = another.MeasureType;
            Synced = another.Synced;
            Value = another.Value;
            CorrectionAmmount = another.CorrectionAmmount;
            Id = another.Id;
        }
        public Sample()
        {
            Id = Guid.NewGuid().ToString();
        }
    }
}
```

14.2. Sample Repository

```
using ApiBase.Data;
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;

namespace ApiBase.Models
{
    public class Sample : IEntity<Sample>
    {
        [Key]
```

```
        public string Id { get; set; }
        public float Value { get; set; }
        public string OwnerId { get; set; }
        public string MeasureType { get; set; }
        public int CorrectionAmmount { get; set; }
        public DateTime Timestamp { get; set; }
        public bool Synced { get; set; }
        internal void CreateUID()
        {
            Id = Guid.NewGuid().ToString();
        }
        public void CopyFrom(Sample another)
        {
            OwnerId = another.OwnerId;
            MeasureType = another.MeasureType;
            Synced = another.Synced;
            Value = another.Value;
            CorrectionAmmount = another.CorrectionAmmount;
            Id = another.Id;
        }
        public Sample()
        {
            Id = Guid.NewGuid().ToString();
        }
    }
}
```

14.3. Sample Controller kódrészlet

```
namespace ApiBase.Controllers
```

```
{
    [ApiController]
    [Route("[controller]")]
    public class ProfileController : ControllerBase
    {
        protected readonly UserManager<IdentityUser> _userManager;
        protected readonly IProfileRepository _repository;
        protected ILogger<SampleController> _logger;
        protected readonly IHubContext<EventHub> _hub;

        public ProfileController(UserManager<IdentityUser> userManager, IProfileRepository repository, ILogger<SampleController> logger, IHubContext<EventHub> hub)
        {
            _userManager = userManager;
            _repository = repository;
            _logger = logger;
            _hub = hub;
        }

        [HttpGet("{id}")]
        [Authorize]
        public JsonResult GetMyProfile(string id)
        {
            var myself = CurrentUserId();
            return new JsonResult(_repository.GetAll().FirstOrDefault(t => t.OwnerId == myself));
        } …
```